FINITE ELEMENT MODEL OF
SHAPE AND DENSITY ADAPTATION
IN ENGINEERED BONE

THESIS

Presented in Partial Fulfillment of the Requirements for
the Degree Master of Science in the Graduate
School of The Ohio State University

By
Rebecca M. Jansen, B.S.

* * * * *

The Ohio State University
2008

Master’s Examination Committee:
Professor Richard T. Hart, Advisor
Professor Alan S. Litsky

Approved by

Advisor
Biomedical Engineering Graduate Program
ABSTRACT

A finite element method is combined with control theory and used to model bone’s shape and density adaptations. The model is implemented using Simulink (The MathWorks Inc., Natick, MA) along with functions from MATLAB (The MathWorks Inc., Natick, MA) and COMSOL (COMSOL Inc., Burlington, MA). Regulators of shape and density adaptations are user-definable as a function of time or as a function of a mechanical signal such as strain. The described technique is used to simulate the adaptations in a tissue engineered bone graft, or bone flap. First, the functionality of each of the remodeling rules is demonstrated. Then, shape and density remodeling is combined to replicate the bone flap’s adaptations and compared to experimental data. Finally, the model is used to theorize how the bone flap may behave when reimplanted during a maxilla reconstruction.
This work is dedicated to my parents,

Mark and Kathleen Jansen.
ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Richard Hart for his wonderful teaching and mentoring. I am very grateful for all the time he has spent, the support he provided, and the encouragement he has given.

I would also like to thank Dr. Mike Miller, who provided the inspiration and experimental data for this project and was kind enough to meet with me to discuss this project. Thanks also goes to Dr. Alan Litsky, for his time in serving on my thesis committee.

Thank you to fellow graduate students Xiaoyue Ma and Francis Sheer for their help with brainstorming and troubleshooting. Special acknowledgements go to Drs. Xia Liu and Michael Roberts, whose work in bone adaptation modeling formed the foundation for this thesis.

I am grateful to my parents and family, for all their love, support and encouragement. I especially appreciate of all the sacrifices my parents have made in order to provide me with the best education possible.

Finally, I would like to thank my fiancé David for his love, patience, and encouragement to always to follow my heart.
VITA

September 4, 1981 ........................................... Born – St. Louis, Missouri

2005 .......................................................... B.S. Biomedical Engineering, Marquette University, Milwaukee, Wisconsin

January – August 2006 ................................... Graduate Teaching Assistant, Tulane University, New Orleans, Louisiana

September 2006 – December 2008 ................. Graduate Research Assistant, The Ohio State University, Columbus, Ohio

FIELDS OF STUDY

Major Field: Biomedical Engineering
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Dedication</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>iv</td>
</tr>
<tr>
<td>Vita</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xiii</td>
</tr>
<tr>
<td>List of Equations</td>
<td>xix</td>
</tr>
</tbody>
</table>

## Chapter 1  Introduction

1.1 Motivation ................................................................................. 1

1.2 Bone Function and Structure............................................................ 2

1.3 Bone Composition and Formation.................................................... 4

1.4 Bone Remodeling ............................................................................. 5

1.5 Fracture Healing ............................................................................ 6

1.6 Bone-Graft Materials ..................................................................... 8

1.7 Craniofacial Reconstruction Considerations ..................................... 11

1.8 Bone flap experiments .................................................................... 11
1.9 Previous modeling of bone adaptation ........................................... 15

Chapter 2 Materials and Methods .................................................. 21
2.1 Overview and Motivation ............................................................. 21
2.2 Software .................................................................................. 21
2.3 Pre-Simulation Steps .................................................................. 22
  2.3.1 Remodeling Rules ................................................................. 23
  2.3.2 Creation, Storage and Use of Control Points ......................... 25
  2.3.3 FEM Set-Up ......................................................................... 31
2.4 Model Simulation ........................................................................ 38
  2.4.1 S-function Subroutines Setup, DoPostPropSetup, InitConditions, & Update 41
  2.4.2 S-function Subroutine Output ................................................. 42
    2.4.2.1 Creating and Solving the fem Structure .......................... 42
    2.4.2.2 Mechanical Signal Extraction for Shape Adaptation .......... 45
    2.4.2.3 Density Adaptation ......................................................... 46
  2.4.3 MATLAB Functions get localaxial & get unitnormal .............. 51
  2.4.4 Shape Adaptation ................................................................. 52
  2.4.5 Growth ............................................................................... 57
  2.4.6 Applying the Shape Changes ............................................... 59
2.5 Post-Processing .......................................................................... 59
2.6 Methods for Simulation E .............................................................................. 60

Chapter 3 Results ............................................................................................... 66
3.1 Simulation A ............................................................................................... 66
3.2 Simulation B ............................................................................................... 70
3.3 Simulation C ............................................................................................... 74
3.4 Simulation D ............................................................................................... 79

Chapter 4 Discussion ......................................................................................... 91
4.1 Comments on the Results ........................................................................... 91
4.2 Validity of the Results ................................................................................ 95
4.3 Limitations of the Model ............................................................................ 97
4.4 Modeling Advances .................................................................................... 102
4.5 Future Studies and Improvements ............................................................... 103

Chapter 5 Conclusion ......................................................................................... 106

Appendix MATLAB M-File Functions and Subroutines ........................................ 107
A.1 Pre-Simulation Routines ............................................................................. 109
A.1.1 M-file Script: filesforblock.m .............................................................. 109
A.1.2 M-file Function: readcontourpts.m .......................................................... 111
A.1.3 M-file Function: readdatafile.m ............................................................. 113
A.1.4 M-file Function: creategeom.m ............................................................. 115
A.1.5 M-file Function: createcentfile.m .......................................................... 117

A.2 Simulation Routines .................................................................................. 119
A.2.1 Level 2 M-file S-function: msfun_bone.m ........................................... 119
A.2.2 M-file Function: creategeom.m ............................................................. 129
A.2.3 M-file Function: Dinterpolation.m ......................................................... 129
A.2.4 M-file Function: intercepts.m ................................................................. 131
A.2.4.1 M-file Subfunction: findcontrolpts.m .............................................. 134
A.2.4.2 M-file Subfunction: findtangents.m .................................................. 136
A.2.4.3 M-file Subfunction: drawclosedspline.m ......................................... 137
A.2.4.4 M-file Subfunction: contoursegintercept.m ...................................... 139
A.2.4.5 M-file Subfunction: perpdist3d.m ..................................................... 141
A.2.4.6 M-file Subroutine: samplebspline.m ................................................. 143
A.2.5 Simulink Blocks: get localaxial & get unitnormal ............................... 145
A.2.6 Simulink Subroutine: Shape Adapt...................................................... 145
A.2.6.1 Simulink Blocks for Mechanostat Function ........................................ 146
A.2.6.2 M-file Function: slopes.m ................................................................. 146
A.2.6.3 M-file Function: calcspeed.m ......................................................... 147
A.2.7 Simulink Subroutine: Grow ................................................................. 149

A.2.8 M-file Function: deltaXYZ.m ............................................................ 149

Bibliography ................................................................................................. 151
LIST OF TABLES

Table 1.1: Comparison of properties of autografts and allografts. Redrawn from Greenwald et al. (2001) ................................................................. 9

Table 1.2: Eight different density to modulus relationships. (Oden et al., 1993) ....... 18

Table 2.1: Summary of simulations and a brief description of each ......................... 25
Table 2.2: Parameters used in filesforblock.m to create control point files .............. 29
Table 2.3: Variable inputs for model ........................................................................ 30
Table 2.4: Initial material properties for each simulation ........................................ 34
Table 2.5: Function expression for dentomod(rho_sld), where rho_sld is in kg/m$^3$ and the output is in Pa ............................................................... 35
Table 2.6: Free mesh parameters for the finite element mesh .................................. 37
Table 2.7: Parameters for mechanically driven density adaptation used in this implementation .................................................................... 51

Table 3.1: Summary of parameters and settings for Simulation A .......................... 67
Table 3.2: Summary of parameters and settings for Simulation B ............................ 71
Table 3.3: Summary of parameters and settings for Simulation C ............................ 75
Table 3.4: Summary of the parameters and settings for Simulation D. ..........................80
Table 3.5: Summary of the parameters and settings for Simulation E..........................84

Table A.1: List of user-defined MATLAB functions where (s) – script, (F) – function, (SF) – S-function. .................................................................................................................. 108

Table A.2: Definition of MATLAB functions get localaxial and get unitnormals. ......145
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Diagram of a fracture callus. Redrawn from Martin et al. (2004).</td>
</tr>
<tr>
<td>1.2</td>
<td>Location of bone flap chamber, attached to the periosteum of a rib in sheep model. Redrawn from Cheng et al. (2005).</td>
</tr>
<tr>
<td>1.3</td>
<td>Schematic of the feedback process involved in bone adaptation. (Hart, 1983; Hart, Davy and Heiple, 1984)</td>
</tr>
<tr>
<td>1.4</td>
<td>A shape remodeling rule that converts a mechanical signal, S, into a bone formation or resorption rate, V. (Roberts and Hart, 2005)</td>
</tr>
<tr>
<td>2.1</td>
<td>Points are used to define the perimeter of the sliced geometry.</td>
</tr>
<tr>
<td>2.2</td>
<td>An input file containing the coordinates of a single slice’s control points.</td>
</tr>
<tr>
<td>2.3</td>
<td>Flowchart for Section 2.3.2 with steps to create, store and read control points.</td>
</tr>
<tr>
<td>2.4</td>
<td>Flowchart for Section 2.3.3, displaying steps to set-up the finite element model.</td>
</tr>
<tr>
<td>2.5</td>
<td>A) The control points for all slices. B) Continuous splines connect the control points of each slice. C) The splines are lofted to create solid geometry.</td>
</tr>
</tbody>
</table>
Figure 2.6: Plot of density ($\rho_{sld}$) in kg m$^{-3}$ versus Young’s modulus in Pa, as defined by $dentomod(\rho_{sld})$.

Figure 2.7: The meshed geometry used in Simulations A – D.

Figure 2.8: Simulink model, $BoneFeedback.mdl$, which controls shape and density adaptation.

Figure 2.9: The Simulink model $BoneFeedback.mdl$ with the Level-2 M-file S-function $msfun\_bone.m$ highlighted.

Figure 2.10: Flowchart for Section 2.4.2, showing steps to set-up the s-function $msfun\_bone.m$.

Figure 2.11: Graph showing volume fraction by week of bone flap grown in MBG-filled chamber. Data from Cheng et al. (2005).

Figure 2.13: Graph showing structural density by week of bone flap grown in MBG-filled chamber. Data from Cheng et al. (2005).

Figure 2.15: The Simulink model $BoneFeedback.mdl$ with MATLAB functions $get localaxial$ and $get unitnormal$ highlighted.

Figure 2.14: Flowchart showing steps needed to set-up $Shape Adapt$ and $Grow$.

Figure 2.17: The Simulink model $BoneFeedback.mdl$ with the subroutine, $Shape Adapt$, highlighted and expanded. $Shape Adapt$ performs the mechanically driven shape adaptation. The bracketed blocks implement the mechanostat function.
Figure 2.16: Graph showing total tissue volume (TTV) by week of bone flap grown in MBG-filled chamber. Data from Cheng et al. (2005) ........................................................56

Figure 2.17: Graph of the mechanostat function used to relate strain to the rate of bone apposition or resorption. The areas where the apposition rate is 0 are often referred to as the “lazy zone” ..........................................................57

Figure 2.18: Simulink model BoneFeedback.mdl with the subroutine, Grow, highlighted and expanded. Grow performs the non-mechanically driven shape adaptation ..................................................................................58

Figure 2.19: The Simulink model BoneFeedback.mdl with the simout to Workspace block highlighted. The block saves the Simulink signal at that location to the MATLAB Workspace ..................................................................................61

Figure 2.20: Initial geometry for Simulation E ....................................................................................................................62

Figure 2.21: Bite load applied to the top surface of the block is distributed along the x-direction of the block ........................................................................................................63

Figure 2.22: Boundary conditions for Simulation E are applied to the right and left top edges. The left top edge is fixed in x-, y- and z-directions and shown in green. The right top edge is fixed in y and blue. ........................................................................64

Figure 2.23: Initial meshed geometry in Simulation E ...........................................................................................................65
Figure 3.1: Plot of structural density by week in simulation A, following a remodeling rule based on linear trendline through data from Cheng et al. (2005)..........................68

Figure 3.2: Results from Simulation A, showing a cross-section of the block and plotting density for week 0 (top left), 3 (top right), 6 (mid left), 9 (mid right), 12 (bottom left) and 24 (bottom right)........................................................................................................69

Figure 3.3: Plot of structural density by week according to Cowin’s remodeling rule (1986; 1992) in the special case, like Simulation B, when no strain is present........72

Figure 3.4: Results from Simulation B showing a cross-section of the block and plotting density for week 0 (top left), 3 (top right), 6 (mid left), 9 (mid right), 12 (bottom left) and 24 (bottom right)........................................................................................................73

Figure 3.5: The initial geometry (left) and the geometry after 9 weeks (right) in Simulation C........................................................................................................................................74

Figure 3.6: Results from Simulation C showing a cross-section of the block through a y-z plane for week 0 (top left), 3 (top right), 6 (bottom left), and 9 (bottom right).......76

Figure 3.7: Results from Simulation C showing a cross-section of the block through an x-z plane for week 0 (far left), 3 (mid left), 6 (mid right), and 9 (far right)............77

Figure 3.8: Plot comparing the measured total tissue volume (TTV) by week of bone flap grown in MBG-filled chambers (Cheng et al., 2005), plotted with circles, and the geometric volume during finite element Simulation C, plotted with squares. Note
that, although not specifically measured, TTV at time zero is approximately 4 cm$^3$

since the chamber was filled with MBG.

Figure 3.9: Results from Simulation D showing a cross-section of the block through a y-z plane and plotting density for week 0 (top left), 3 (top right), 6 (bottom left), and 9 (bottom right).

Figure 3.10: Results from Simulation D showing a cross-section of the block through an x-z plane and plotting density for week 0 (top left), 3 (top right), 6 (bottom left), and 9 (bottom right).

Figure 3.11: Results from Simulation E showing the change in shape at week 0 (top left), week 3 (top right), week 6 (bottom left) and week 9 (bottom right).

Figure 3.12: Results from Simulation E showing a slice plot of density at week 0 (top) and week 3 (bottom).

Figure 3.13: Results from Simulation E showing a slice plot of density at week 6 (top), and week 9 (bottom).

Figure 3.14: Result from Simulation E plotting the trace of the strain tensor on the deformed geometry at week 0. Deformation has a scale factor of 1. The trace of the strain tensor is the mechanical input for density remodeling.

Figure 3.15: Results from Simulation E plotting the trace of the strain tensor onto the deformed geometry at week 3 (top), week 6 (middle), and week 9 (bottom).
Deformation has a scale factor of 1. The trace of the strain tensor is the mechanical input for density remodeling.

Figure 4.1: As control points are moved inward, corners may develop, leading to the inability to identify a unique tangent vector.

Figure 4.2: Moving the control points too far inward will cause the spline to overlap itself, and the simulation will fail.
# LIST OF EQUATIONS

<table>
<thead>
<tr>
<th>Equation</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equation 1.1</td>
<td>17</td>
</tr>
<tr>
<td>Equation 1.2</td>
<td>17</td>
</tr>
<tr>
<td>Equation 2.1</td>
<td>46</td>
</tr>
<tr>
<td>Equation 2.2</td>
<td>47</td>
</tr>
<tr>
<td>Equation 2.3</td>
<td>49</td>
</tr>
<tr>
<td>Equation 2.4</td>
<td>49</td>
</tr>
<tr>
<td>Equation 2.5</td>
<td>55</td>
</tr>
<tr>
<td>Equation 2.6</td>
<td>55</td>
</tr>
</tbody>
</table>

xix
CHAPTER 1

INTRODUCTION

1.1 Motivation

Bone is a structural material, but unlike traditional structural materials, bone is a living tissue, capable of adapting to its environment. The mass and geometry of bone is greatly determined by its loading environment. A change in bone’s loading environment can cause the bone tissue to adapt, possibly changing the shape and density of the bone. Understanding how bone adapts and grows is essential to developing methods to repair bone defects and treatments for bone diseases.

Bone grafting is a common technique currently used to repair bone defects (Greenwald et al., 2001). However, the many drawbacks to bone graft surgery have led scientists and engineers to attempt to grow tissue engineered bone grafts, called bone flaps. Their goal is to grow bone either in vitro or in vivo as an alternative to the autografts and allografts that are currently available for reconstructive surgery. Because the structure of bone is so strongly tied to its function and loading, success in creating tissue engineered bone grafts may depend on understanding how the grafts are loaded during development.
Developing a computer model to simulate and analyze the stresses, strains, geometry alterations and density changes is possible using finite element analysis. Advances in computational power and methods now enable the development of bone adaptation models. Finite element analysis is combined with control theory to apply the appropriate loading to bone geometry and simulate how the shape and density of the bone adapts with the strain environment. This technique may be useful in analyzing bone flaps. Creating computer models using experimental data as input may allow researchers to predict the graft’s behavior and, in turn, develop ways to control the graft’s growth and material properties.

1.2 Bone Function and Structure

One of the main functions of bone, in addition to serving as a mineral reservoir, is to provide structural support to the body (Goodship and Cunningham, 2001). Bones are often grouped by their shape and function. For example, long bones, such as those in the limbs, work as mechanical levers in conjunction with muscles to enable movement. Flat bones, like the skull and pelvis, provide a protective barrier for underlying tissues and organs. The structural composition of bone greatly depends on its function. This concept will be explored further in Section 1.4.

There are two types of bone tissue: cortical and trabecular (also called cancellous). Cortical bone comprises the shafts of long bones, the exteriors of some flat bones, and the thin layers around vertebra. Trabecular bone is located in the ends of long bones and composes most of the volume of flat bones and vertebra. The two types of
tissues have very different structures. Cortical bone is compact and dense with a porosity of about 5-10%, whereas trabecular bone has a spongy appearance with a porosity of 75-95% (Martin et al., 2004).

A membrane called the periosteum envelops bone’s outer surface. It is composed of an outer fibrous layer and an inner cambium layer. The cambium layer is highly cellular and has a high proportion of osteoblasts (Martin et al., 2004). Both bone apposition and resorption can occur at the periosteal surface. Drift, a term describing the phenomenon that expands the ribs and pelvis as one ages, is a result of apposition occurring along the periosteal surface of one side of the bone and resorption occurring on the opposite side. Bone growth is largely controlled by the cells in the periosteum. However, the role that mechanical forces play in periosteal bone growth is unknown (Orwoll, 2003). The periosteum also plays an important role in both bone remodeling and in fracture healing.

The shafts of long bones are hollow tubes. The interior surface, referred to as the endosteal surface, is covered with bone lining cells. Like the periosteum, these cells may play a role in bone adaptation. But unlike the periosteum, the bone lining cells do not assist with fracture healing (Martin et al., 2004).

The shape and density of bone is determined by both its strain environment and genetics. Chalmers and Ray (1962) showed that in the absence of mechanical loading, bones developed towards their normal physiologic shape, indicating a genetic role in the modeling process. They transplanted the femurs from fetal mice into the spleens of mice from the same strain, producing an isograft. The femurs continued to grow and ossify as
in normal femoral development, although the overall size was noticeably smaller. Even without mechanical loading, the bones’ shapes were easily recognizable as those of a femur, and they had many of the same bony landmarks as normal femurs. This experiment showed it possible for bone to grow in unconventional areas, although the mechanical properties of the resulting bone may be remarkably different than a functionally loaded bone.

1.3 Bone Composition and Formation

Bone is essentially a composite material made mostly of collagen, hydroxyapatite mineral, and bone cells (Jee, 2001). There are four major types of bone cells. Osteoblasts are mononuclear, bone-forming cells, producing osteoid, the organic component of the bone matrix. Osteoclasts are multinucleated cells that resorb bone. Osteocytes are former osteoblasts that have been buried in osteoid. It has been proposed that the osteocytes may sense strain and communicate strain information to surface cells via cellular processes that connect them (Jee, 2001). Bone lining cells are flattened cells that cover bone surfaces that are not undergoing remodeling. They are thought to play a role in mineral homeostasis maintenance and may also sense functional strain.

There are two methods of bone formation – endochondral ossification and intramembranous ossification (Jee, 2001). Endochondral ossification occurs when cartilage is calcified and then replaced by bone. This type of ossification forms the long bones during embryonic development and also plays a role in fracture healing (see Section 1.5). On the other hand, intramembranous ossification is direct bone formation
and is responsible for the embryo’s development of flat bones. Intramembranous ossification also takes place along the periosteum during shape changes, which often referred to as “modeling”.

1.4 Bone Remodeling

In 1892, Wolff published observations that pioneered bone adaptation research. He noted that, at steady state, the trabeculae, or struts, of cancellous bone are oriented in the same directions as the local principle directions of stress. This observation led him to believe that bone remodeling follows mathematical rules based on the stresses in the bone (Wolff, 1892; 1986). Wolff introduced the idea that bone shape is a result of bone function, or functional adaptation. In more recent years, researchers have argued that strain and strain rate, not stress, are probably the signals that initiates bone remodeling (Turner, 1998). However, the exact mechanism that transduces the strain into a biological signal for bone cells to either resorb or lay down bone is largely unknown. What is known is that mechanical loading, or lack thereof, causes the bone cells to respond by either resorbing bone tissue or depositing it.

Bone is a tissue that is constantly being turned over and remodeled based on its functional loading. The term “remodeling” describes how basic multicellular units made of osteoblasts and osteoclasts are continually renewing bone tissue by resorbing and laying down bone (Martin et al., 2004). In contrast, bone “modeling” refers to a change in the overall shape of the bone. Bone volume increases when osteoblasts are more active than osteoclasts. When osteoclasts are more active than osteoblasts, bone volume
decreases. Bone modeling appears to alter the volume and density of bone in order to reach the optimal mass needed to carry the loading it sees (Goodship and Cunningham, 2001). When loading is reduced below normal usage, bone tissue undergoes net resorption. When loading exceeds normal loading, net bone apposition occurs. Net bone apposition and resorption can occur either locally or systemically. For example, systemic bone loss is seen during space flight when bones are no longer exposed to gravity forces and during long-term bed rest (Goodship and Cunningham, 2001). Orthopedic implants and fixation devices can induce localized bone loss by carrying the load usually carried by the bone tissue (Goodship and Cunningham, 2001).

1.5 Fracture Healing

Bone, like most structural materials, will fracture when loaded beyond its capacity. Usually the body is able to repair fractures without assistance via either primary or secondary fracture healing. In primary fracture healing, the fracture gap ossifies via intramembranous bone formation without forming a fracture callus. However, the more common method of bone repair is secondary fracture healing (Prendergast and van der Meulen, 2001).

Normal secondary fracture healing is self-controlled and can be divided into three stages (Carter et al., 1998). During the first stage, the wound is flooded with pluripotent tissue, which has the ability to differentiate into cartilage, bone, or fibrous tissue. This tissue forms a fracture callus (see Figure 1.1) that functions to stabilize the fracture. The second stage involves endochondral ossification of the cartilage in the
fracture callus. During the final stage, both endochondral and intramembranous bone formation occurs. The length of time needed to heal a fracture varies by location. Ribs and other bones in the torso can heal in only a few weeks, whereas bone of the appendicular skeleton may take several months (Martin et al., 2004).

Figure 1.1: Diagram of a fracture callus. Redrawn from Martin et al. (2004).

Despite bones’ ability to heal, there are situations, such as congenital defects, traumatic injury, or tumor resection, which leaves the bone with permanent defects that will not heal on their own. Surgical reconstruction is often needed to repair such defects. A large volume of replacement tissue is sometimes required for reconstruction of these defects.

Currently, the most common technique for reconstruction consists of harvesting autologous bone from donor sites like the fibula or iliac crest, reshaping the harvested
bone, and transplanting it into the site of the defect. However, this technique has many limitations: availability of donor sites; damage, pain and possible infection at the donor site; and the need to reshape harvested bone to fit the shape of the defect (Miller et al., 1996). Integration of the bone into the defect and possible resorption of the graft is also a concern (Zuk, 2008).

1.6 Bone-Graft Materials

Bone-graft materials can be evaluated by measuring its structural strength and assessment of how well the material promotes osteogenesis, osteoinduction, and osteoconduction. Osteogenesis occurs when new bone is synthesized by cells or cellular components that are already present in the graft material before transplantation. Osteoinduction promotes bone formation by actively recruiting mesenchymal stem cells from host tissues that surround the graft. Osteoinduction is often assisted by the presence of bone morphogenic proteins (BMPs) that are present in the graft materials. Osteoconduction is a passive process that allows blood vessel and bone formation due to its porous structure. All three of these processes are important processes of bone formation (Greenwald et al. 2001).

Autografts are tissue grafts taken from the patient’s own body. Because it’s the patient’s own tissue, autografts eliminate the possibility of immune system rejections. Also, bone autografts are living tissue and capable of osteogenesis, osteoinduction and osteoconduction. However, autografts require harvest sites, which are limited, cause
additional pain, can become infected, and can damage surrounding tissues (Miller et al., 1996).

Allografts are tissue grafts from other members of the same species, usually cadavers. Allografts have the advantage of not requiring bone harvesting from the patient, thus eliminating donor-site morbidity and the possibility of infection. However, since the graft is from another person, the tissue must be irradiated to eliminate the possible transmission of infectious agents. This irradiation kills any living cells, leaving the graft with no ability for osteogenesis and very little, if any, osteoinduction. The irradiation, and subsequent freezing, also reduces the graft’s structural strength. Table 1.1 compares the properties of autografts and allografts.

<table>
<thead>
<tr>
<th>Bone Graft</th>
<th>Structural Strength</th>
<th>Osteoconduction</th>
<th>Osteoinduction</th>
<th>Osteogenesis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Autograft</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cancellous</td>
<td>No</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>Cortical</td>
<td>+++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td><strong>Allograft</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Cancellous</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frozen</td>
<td>No</td>
<td>++</td>
<td>+</td>
<td>No</td>
</tr>
<tr>
<td>Freeze-Dry</td>
<td>No</td>
<td>++</td>
<td>+</td>
<td>No</td>
</tr>
<tr>
<td><em>Cortical</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frozen</td>
<td>+++</td>
<td>+</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Freeze-Dry</td>
<td>+</td>
<td>+</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 1.1: Comparison of properties of autografts and allografts. Redrawn from Greenwald et al. (2001).
Other options for tissue replacement include xenografts, which are currently not approved for use in the US, and bone graft substitutes. Xenografts are tissue harvested from another species. They are capable of osteogenesis, osteoinduction and osteoconduction, but can incite immunity and inflammatory responses. Bone graft substitutes are also commercially available. They often are in powdered form, which limits their structural strength. Because they are not made of living tissue, bone graft substitutes are usually only osteoconductive. Bone morphogenic proteins are sometimes added to encourage osteoinduction (Greenwald et al., 2001).

Of the available grafts, autografts offer the best structural properties. In addition, autografts allow for the possibility of vascularized grafts. Proper vascularization of grafts is essential to their survival and incorporation into the surrounding tissue; lack of proper vascularization promotes chondrogenesis instead of osteogenesis (Carter et al., 1998).

The number of bone graft procedures performed in the United States continues to increase. While the number of musculoskeletal tissue donors has also increased, there is still a shortage of allograft material (Greenwald et al., 2001). The demonstrated superiority of allografts and autografts over bone-graft substitute materials provides strong motivation to look into tissue-engineered grafts with similar bone forming properties, but without the associated donor-site morbidity.
1.7 Craniofacial Reconstruction Considerations

There are almost 600,000 bone grafts completed in the United States each year; of these, 6% are craniofacial reconstructions (Zuk, 2008). Craniofacial reconstruction presents different challenges than reconstruction of the appendicular skeleton. The aesthetics of reconstruction in the facial area is a much larger concern than in other locations. The reconstruction’s success or lack thereof can be the difference between social acceptance and alienation, especially for children (Zuk, 2008).

There are also functional considerations that are only associated with mid-facial reconstructions. The proper alignment of teeth is dependent on structural stability of the maxilla and mandible. Damage to the maxilla can also affect drainage of the middle ear and lead to an increased occurrence of ear infections. Facial deformations, improper alignment of teeth, and increased incidence of ear infections can negatively influence speech (Zuk, 2008).

1.8 Bone flap experiments

In response to these limitations, Miller et al. (1996), developed a technique to grow bone flaps using the periosteum. A sheep model was chosen to demonstrate feasibility, and four fully-grown sheep where each implanted with four chambers. The chambers had a profile measuring 10 x 40 mm and were closed on five sides. Half the chambers had a depth of 5 mm; the others had a depth of 10 mm. The chambers were made of poly-methyl methacrylate (PMMA) with a cuff of poly-tetrafluoroethylene (PTFE) around the open base. While under general anesthesia, the sheep’s periosteum
was opened, a 6-cm segment of rib bone was removed, and the chambers where implanted at alternating levels. Two shallow and two deep chambers were implanted in each sheep. One of each was implanted empty (the control), and the others were filled with morcellized corticocancellous bone graft (MBG) made from the removed bone segment. The chambers were sutured to the periosteum with the open side of the chamber facing the cambium layer (see Figure 1.2). The chambers were harvested after 6, 9 and 13 weeks. The contents of the chamber underwent histological analysis. The empty chambers showed dense fibrous tissue, but no bone. However, the MBG-filled chambers contained osteocytes and a network of blood vessels. The shape of the bone flaps after 6 and 9 weeks matched the chamber’s interior dimensions. However, after 13 weeks, the shape was less clearly defined, indicating that some of the bone had probably been resorbed.
Cheng et al. (2005) expanded upon the work done by Miller et al (1996). Twenty sheep were each implanted with an empty chamber (control), an MBG-filled chamber, and a chamber filled with “deactivated” MBG (deMBG). The deMBG is MBG that had been autoclaved to denature the bioactive proteins. The chambers were harvested after 3, 6, 9, 12 and 24 weeks. The volume of the contents of the chamber was measured. Then the tissue block was digitally scanned to record the tissue shape and underwent histological analysis. The empty chambers contained only fibrovascular tissue. The MBG-filled chambers showed active endochondral bone formation. The deMBG-filled chambers showed little bone formation. Both the MBG- and deMBG-filled chambers
had significantly larger the total tissue volumes than the empty chambers. The authors observed that, over time, the total amount of bone in deMBG- and MBG-filled chambers increases while the total tissue volume decreases. The authors acknowledge that the model is limited by the lack of forces applied to the chambers during bone flap development.

Brey et al. (2007) attempted to determine if the bone flaps formed better when chambers were attached to periosteum or when attached to muscles fascia. Twenty sheep were implanted with two chambers each. All chambers contained MBG. One chamber per animal was attached to the periosteum of the rib as previously stated. The other chamber was implanted on the latissimus dorsi muscle with the open side of the chamber facing the muscle fascia. The chambers were harvested at 3, 6, 9, 12 and 24 weeks. The contents of the chamber were digitally scanned to obtain the dimensions and then sectioned for histological analysis. The results showed no significant differences between the chambers implanted on the periosteum and fascia in the volume of tissue harvested. The periosteum-based chamber showed an increase in the amount of calcified tissue over the five time periods, whereas amount of calcified tissue in the fascia-based chamber remained constant.

Brey et al. (2007) observed that the main loss of tissue was a result of a decreased height and that tissue resorption was not uniform. They concluded that the periosteum was significantly better for bone formation, indicating that the periosteum’s vascularity may give it an advantage in bone formation. Brey et al. (2007) also theorized that further study was need to determine proper harvesting time and suggested that local forces in the
area where the bone flap is reimplemented may help in adapting to mature bone. Herring and Ochareon (2005) even suggest “under-engineering” engineered tissue to allow adaptive remodeling to occur.

1.9 Previous modeling of bone adaptation

Bone adaptation can be analyzed as a physiological control process, as proposed by Hart and coworkers (Hart, Davy and Heiple, 1984; Hart, 2001). Figure 1.3 shows a schematic of the feedback process of bone. Bone formation and resorption occurs as a result of mechanical loading via the strain remodeling potential as well as non-mechanical signals such as hormonal, metabolic and genetic factors. The cellular activity that results in bone remodeling can take two forms: adaptation of geometric properties and changes in material properties.
Cowin (1986; 1992) suggests the idea of a remodeling equilibrium (RE), a set of conditions where no orientation or density remodeling takes place. Using the idea, Cowin develops a constitutive model to describe how the density and trabecular architecture of bone adapts with time. The model compares current levels of strain to the RE strain and uses the difference to create a rate of change for trabecular orientation and volume fraction. The model is mathematically difficult to implement except in simplified cases where one assumes that the trabecular orientation and density are independent. Then the change in density can be expressed as:
\[
\dot{e} = (f_1 + f_2 e) (\text{tr}(E) - \text{tr}(E^\circ))
\]  

Equation 1.1

where \( \dot{e} \) is the time rate of change of volume fraction of bone, \( e \) is the volume fraction of bone, \( f_1 \) and \( f_2 \) are constants, \( \text{tr}(E) \) is the trace of the strain matrix, and \( \text{tr}(E^\circ) \) is the trace of the RE strain.

There have been many attempts to relate bone’s density to its elastic modulus (Carter and Hayes, 1977; Rice et al., 1988; Snyder and Schneider, 1991; Hodgkinson and Currey, 1992; Kabel et al., 1999). The proposed relationships are based on a wide range of experimental data using various methods, bone samples, and assumptions. There is still debate on which relationship is best, and there are significant differences between the relationships (Oden et al., 1993). Table 1.2 shows eight different density to modulus relationships.

Oden (1994) selected a piecewise definition using functions proposed by Rice et al. (1988) and Snyder and Schneider (1991):

\[
E = 0.06 + 0.9 \rho^2 \text{ GPa for } 0.0 < \rho \leq 1.0
\]

\[
E = 10^{0.59} \rho^{2.39} \text{ GPa for } 1.0 < \rho \leq 2.0
\]

Equation 1.2

where \( \rho \) is structural density in g/cm\(^3\). These functions were selected by Oden based on the large number of both cancellous and cortical bone experimental samples used to determine density to modulus relationship.
Table 1.2: Eight different density to modulus relationships. (Oden et al., 1993)
A shape adaptation approach is proposed by Roberts and Hart (2005). A finite element model (FEM) is created using a geometry whose perimeter is defined by closed B-splines. The geometry is constructed using Patran (MacNeal-Schwendler Co., Costa Mesa, CA). The FEM is solved using ABAQUS (Hibbitt, Karlsson, and Sorenson, Inc., Pawtucket, RI), the shape change is generated using a series of Fortran 90 programs, and the results are visualized using Patran. The geometry’s shape is adapted using a mechanostat-type remodeling rule (see Figure 1.4). A mechanical signal is used to move the splines a distance proportional to the mechanical signal in a direction along the outward normals. The adjusted splines define the new geometry, which is remeshed before the FEM is resolved. The technique is applied to idealized and actual geometries of long bones. This approach to shape adaptation is successful in modeling large geometry changes. Liu et al. (2005) adapted this contour based shape adaptation to run using FEMLAB (now COMSOL Multiphysics (COMSOL Inc., Burlington, MA)), MATLAB (The MathWorks Inc., Natick, MA), and Simulink (The MathWorks Inc., Natick, MA).
Figure 1.4: A shape remodeling rule that converts a mechanical signal, $S$, into a bone formation or resorption rate, $V$. (Roberts and Hart, 2005)
2.1 Overview and Motivation

This chapter presents a method for using control theory and finite element analysis to simulate shape and density adaptations in bone. This method is used to carry out five different simulations (lettered A through E) inspired by the work done by Miller and coworkers (Miller et al., 1996; Cheng et al., 2005; Brey et al., 2007) in growing bone flaps using an ovine model. Miller’s current technique for growing bone flaps results in much of the bone that is formed being resorbed either before it is harvested or shortly after it is reimplanted.

The only materials used in the creation of this model are three software packages, described below. The methods for developing the model can be divided into three parts: pre-simulation, model simulation, and post-processing.

2.2 Software

A combination of three packaged software programs are employed in the development of this model. MATLAB (Version R2008a, The MathWorks Inc., Natick, MA), a numerical coding program that is widely used in engineering research, is the
foundation of the model. MATLAB commands can be executed from both the command line and from m-files containing functions or scripts. Both techniques are employed in the methods below.

The finite element analysis (FEA) in the model is completed using COMSOL Multiphysics (Version 3.4, COMSOL Inc., Burlington, MA). COMSOL was chosen over other FEA packages for its ability to simultaneously solve multiple physics problems. For example, one is able to analyze structural, thermal and fluid interactions in a single model. In addition, because COMSOL was developed out of the partial differential equations (PDE) toolbox within MATLAB, MATLAB functions and operations can be used directly in COMSOL models, and COMSOL commands can be incorporated into m-file functions.

Simulink (Version R2008a, The MathWorks Inc., Natick, MA) is a graphical programming tool that can be used to implement dynamic control loops. MATLAB functions, including COMSOL operations, can be called and run in Simulink simulations. This ability allows finite element analysis to be completed within a control loop.

2.3 Pre-Simulation Steps

Inputs for the model must be assembled before the simulation runs. The first input into the model is geometry information given in the form of control points. The simulation uses the control points to generate the geometry and to control the shape adaptation during the simulation.
The model uses a finite element method to analyze the stresses and strains that serve as input for the control processes of adaptation. Therefore, the next step sets-up boundary and loading conditions for the finite element model. Initial material properties are also defined during this step.

The remodeling rules used to control adaptation are adjusted based on the theory or hypothesis to be tested or simulated. Because researchers have not reached a consensus on the principal mechanical signals that control bone adaptation, this model allows the user to change the signals and alter the mathematical function that controls the adaptation.

### 2.3.1 Remodeling Rules

Both the geometry and the material properties undergo adaptation during the model simulation. There are four paths to adaptation:

- **mechanically driven shape change**
- **mechanically driven density change**
- **non-mechanically driven shape change**
- **non-mechanically driven density change**

Each of these is controlled by an independent adaptation formula. The implementation of these formulas will be discussed later in this chapter in Section 2.4.2.3, Section 2.4.4 and Section 2.4.5.

Mechanically driven adaptations are designed to simulate how the bone changes with overuse or disuse (see Section 1.3). Because the controlling signals are not yet
known, the procedure allows the user to select one and then analyze the consequences of the choices. To set-up the mechanically driven adaptations, the user must first select which mechanical signal (e.g. axial stress, principle strain, strain energy, etc.) will control how the shape or density changes. Then, the user must decide how that signal will alter the model. For mechanically driven shape adaptation, axial strains are used as the mechanical signal in these examples. For mechanically driven density adaptation, the trace of the strain tensor is used in these examples.

Non-mechanically driven signals are designed to simulate how bone may be affected by stimuli such as hormones or wound response mechanisms. These adaptations are independent of any mechanical loading and change only with time. To set-up the non-mechanically driven adaptations, the user determines how the shape or density will change based on the shape or density at the previous time step.

This method of geometry development and modeling of shape adaptations was first developed by Roberts and coworkers (Roberts and Hart, 2005). Later, Liu and coworkers (Liu and Hart, 2005; Liu et al., 2005) adapted the shape adaptation model to the currently used software. Liu and coworkers (Liu and Hart, 2005; Liu et al., 2005) also developed a 2-D shape and density adaptation model, and a 3-D density adaptation model. In this thesis, their methods were modified to allow 3-D modeling of both shape and density adaptations in the same simulation.

Simulation A demonstrates the non-mechanically driven density adaptation. Mechanically driven density adaptation is demonstrated in Simulation B. Simulation C shows mechanically driven shape adaptation. The mechanically driven shape and density...
adaptations are combined in Simulations D and E. The behavior of the bone flap during
growth in the chamber is modeled in Simulation D. Simulation E presents how,
theoretically, the bone flap will behave when loaded after being harvested and used in the
reconstruction of the maxilla. These simulations are summarized in Table 2.1.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Demonstrates density adaptation as a function of time.</td>
</tr>
<tr>
<td>B</td>
<td>Demonstrates density adaptation as a function of strain</td>
</tr>
<tr>
<td>C</td>
<td>Demonstrates shape adaptation as a function of strain</td>
</tr>
<tr>
<td>D</td>
<td>Models bone flap development using strain-driven shape and density adaptation</td>
</tr>
<tr>
<td>E</td>
<td>Predicts bone flap response to loading after reimplantation</td>
</tr>
</tbody>
</table>

Table 2.1: Summary of simulations and a brief description of each.

2.3.2 Creation, Storage and Use of Control Points

Imaging modalities, such as CT and MRI, often produce a series of two-
dimensional (2-D) images of sections, or slices, of the three-dimensional (3-D) geometry.
These slices are then digitally reconstructed to form the 3-D geometry. The same
concept is used to construct the geometry used in the bone adaptation model.

The geometry to be analyzed is sectioned through a given x-y plane. Points are
used to outline the perimeter of the sliced geometry. Figure 2.1 shows how a rectangular
block may be sliced and illustrates how control points can be used to define the sliced shape.

Figure 2.1: Points are used to define the perimeter of the sliced geometry.

To fully capture the shape of the geometry to be analyzed, the solid must be sectioned numerous times. The x-, y-, and z-coordinates of the control points are stored in separate text files for each slice. See Figure 2.2 for an example of how the text file is formatted. Each line contains the coordinates for a single point, with the x-, y- and z-coordinates listed in order and separated by commas. Note that because the points for a single slice all belong to the same x-y plane, all of the z-coordinates in a single text file
are the same. These text files will serve as the input to the model, and the control points will be used to reconstruct the 3-D geometry and to control the shape adaptation.

![Image of control points](image.png)

Figure 2.2: An input file containing the coordinates of a single slice’s control points.

In the present implementation, the number of slices used is variable, but the slices must be spaced evenly. Because the reconstruction of the 3-D geometry is dependent on the control points, the number of slices used should be enough to accurately define the geometry. Hence, a regular rectangular block will not need as many slices as an irregularly shaped piece of bone. However, one should remember that while each slice
used increases the number of control points, the volume of data to be managed also increases. Using too many slices may slow computation time considerably.

The number of points used to outline the sliced geometry is also variable. However, for this implementation, the number of control points in a slice must be the same for all slices. Again, enough points should be used so as to accurately outline the sections, but not so many that computations are hampered.

All of the text files for a given geometry are named with the same prefix. The text files are numbered sequentially in order to keep the slice data that they contain in the proper order. The numbers may be incremented by any value, but the increment must be standard. One suggestion is to increment by the distance between the slices, so that the numbering corresponds to the z-coordinate in the file.

Creating geometry text files for a rectangular block geometry is relatively easily. MATLAB m-file script filesforblock.m (see Section A.1.1) uses eight user-defined parameters to create text files for any sized rectangular block. This is the first step shown in the flowchart of Figure 2.3. The initial geometries for Simulations A – D were produced using the parameters shown in Table 2.2. This produces a 4 cm by 1 cm by 1 cm rectangular block that is similar to the size of the chamber used in bone flap development. The construction of the geometry for Simulation E will be discussed separately in Section 2.6.

Text files for other geometry shapes can be created manually by recording the x-, y-, and z-coordinates for each slice. Or, if analyzing a geometry based on CT or MRI
images, image-processing software can be used to define the perimeter of the geometry in each image slice.

Figure 2.3: Flowchart for Section 2.3.2 with steps to create, store and read control points.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Value used in these examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>xnpts</td>
<td>number of control points along x-direction</td>
<td>21</td>
</tr>
<tr>
<td>ynpts</td>
<td>number of control points along y-direction</td>
<td>6</td>
</tr>
<tr>
<td>nslice</td>
<td>number of slices (number of points in z-direction)</td>
<td>6</td>
</tr>
<tr>
<td>width</td>
<td>width of the block, distance along x-axis</td>
<td>0.04</td>
</tr>
<tr>
<td>depth</td>
<td>depth of the block, distance along y-axis</td>
<td>0.01</td>
</tr>
<tr>
<td>height</td>
<td>height of the block, distance along z-axis</td>
<td>0.01</td>
</tr>
<tr>
<td>plotyn</td>
<td>create plot? 1=yes, 0=no</td>
<td>1</td>
</tr>
<tr>
<td>fname</td>
<td>character string for prefix of file names</td>
<td>‘bone’</td>
</tr>
</tbody>
</table>

Table 2.2: Parameters used in filesforblock.m to create control point files.
Before attempting to run any of the programs, each of the above stated variables must be declared in MATLAB’s command line. To produce the model that appears in the results in Chapter 3, the variable statement would be as follows:

```matlab
>> fileprefix='bone'; startslice=0000; slicestep=1; nslice=6;
```

These variables are the inputs to all of the proceeding functions. Table 2.3 gives a list of the variable names, type, and descriptions, as well as the value used to produce the results in this thesis.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Value used in results</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nslice</code></td>
<td>scalar</td>
<td>number of slices (i.e. the number of text file inputs)</td>
<td>6</td>
</tr>
<tr>
<td><code>startslice</code></td>
<td>scalar</td>
<td>number of the first text file or the lowest slice</td>
<td>0000</td>
</tr>
<tr>
<td><code>slicestep</code></td>
<td>scalar</td>
<td>increment used in the text file numbering</td>
<td>1</td>
</tr>
<tr>
<td><code>fileprefix</code></td>
<td>string</td>
<td>file name prefix</td>
<td>‘bone’</td>
</tr>
</tbody>
</table>

Table 2.3: Variable inputs for model

M-file function `readcontourpts.m` is used to read the control point coordinates in from text files into the MATLAB workspace:

```matlab
>> [XYZ1col, npts]=readcontourpts(fileprefix, startslice, slicestep, nslice);
```

This function utilizes `readdatafile.m`, an m-file function that reads a single text file and outputs it as a matrix. `readcontourpts.m` calls `readdatafile.m` for each slice and compiles
the output into a single large matrix called $XYZ1col$. The complete, commented code for these functions is found in Sections A.1.2 and A.1.3. The inputs for $readcontourpts.m$ are found in Table 2.3.

S-functions by nature require: 1) its input to be one-dimensional, and 2) for the input and output to be of the same length. Thus, the input to $msfun\_bone.m$, $XYZ1col$, lists the coordinates in a column vector. In addition, the end of the vector is padded with zeros to allow a larger column vector to be output from $msfun\_bone.m$.

### 2.3.3 FEM Set-Up

The steps to set-up the finite element model are outlined in the flowchart shown in Figure 2.4. The first step in setting up the finite element model is reconstructing the geometry. $XYZ1col$ is an input to the m-file function $creategeom.m$:

```matlab
>> g1=creategeom(XYZ1col(1:npts*3*nslice), npts, nslice);
```

Specifying elements “1:npts*3*nslice” in $XYZ1col$ denotes that only the actual coordinates, and not the zeros used to pad the matrix, should be input into the function. A plot of the control points is shown in Figure 2.5.A. The function $creategeom.m$ connects the control points in each slice with a closed, continuous, first- or second-degree spline, as shown in Figure 2.5.B. The splines are converted to solid 2-D objects and then lofted according to the z-coordinate to produce a solid geometry object, $g1$. Figure 2.5.C shows the solid geometry used as the initial geometry in Simulations A -- D.
Figure 2.4: Flowchart for Section 2.3.3, displaying steps to set-up the finite element model.
Figure 2.5: A) The control points for all slices. B) Continuous splines connect the control points of each slice. C) The splines are lofted to create solid geometry.
Geometry object $gI$ is then imported into COMSOL Multiphysics. A 3-D structural mechanics model is created using metric (SI) units, and a solid, stress-strain, static analysis is selected. The **Import** command from **File** menu imports $gI$ into the new model. One can then use the graphic user interface (GUI) of COMSOL to define initial material properties and apply loading and boundary conditions.

Material properties are defined in the **Subdomain Settings** dialog box under the **Physics** menu. For these simulations, structural density (also sometimes called apparent or bulk density) is used instead of material density. If the geometry’s initial density is constant, the value is simply entered into the dialog box. For more complex initial distributions, a MATLAB function or script may need to be employed. Table 2.4 gives the initial material properties for each of the simulations.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Young’s modulus, $E_{sld}$ (Pa)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dentomod($\rho_{sld}$)</td>
</tr>
<tr>
<td>Poisson’s ratio, $\nu_{sld}$</td>
<td>0.33</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial density, $\rho_{sld}$ (kg/m$^3$)</td>
<td>500</td>
<td>1100</td>
<td>800</td>
<td>1100</td>
<td>800</td>
</tr>
</tbody>
</table>

Table 2.4: Initial material properties for each simulation.

In these simulations, the elastic modulus, or Young’s modulus, is defined as a function of the structural density. As discussed in Section 1.9.1, there are many proposed relationships between density and modulus. These simulations use the functions selected by Oden (1994) as shown in **Equation 1.2**. This is incorporated into the model by
creating a user-defined function in COMSOL. In the **Functions** dialog under the **Options** menu, a piecewise density to modulus function is defined using density as its input. The function is named *dentomod* with input argument *rho_sld* (COMSOL’s variable name for density). Using a general function type and no extrapolation, the function is defined as in Table 2.5, where *rho_sld* is in kg/m$^3$ and the output is in Pascals (Pa). See Figure 2.6 for a plot of function *dentomod(rho_sld)*. This function is used in the Subdomain Settings dialog to define E, Young’s modulus for all simulations (see Table 2.4).

<table>
<thead>
<tr>
<th>$x_{\text{start}}$</th>
<th>$x_{\text{end}}$</th>
<th>$f(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1000</td>
<td>$(.06+.9*(\text{rho}_sld/1000)^2)*10^9$</td>
</tr>
<tr>
<td>1000</td>
<td>2000</td>
<td>$(10^{.59}*(\text{rho}_sld/1000)^{2.39})*10^9$</td>
</tr>
</tbody>
</table>

Table 2.5: Function expression for *dentomod(rho_sld)*, where *rho_sld* is in kg/m$^3$ and the output is in Pa.

The initial boundary and loading conditions for this implementation are defined in the **Boundary Settings** dialog box under the **Physics** menu. For Simulations A – D, the top surface of the block is constrained in all directions (face 12 for this geometry) and no loading applied. The top surface represents the open side of the chamber that faces the periosteum in Miller’s experiments. Since the periosteum is supporting the block, the surface is assumed to be fixed. Because Simulations A – D model the bone flap while it
is still in the chamber, there is no loading applied. The boundary and loading conditions for Simulation E are discussed in Section 2.6.

Figure 2.6: Plot of density \((\text{rho}\_\text{sl}d)\) in kg m\(^{-3}\) versus Young’s modulus in Pa, as defined by \(\text{dentomod(rho}\_\text{sl}d)\).

The finite element mesh is defined using the **Free Mesh Parameters** dialog under the **Mesh** menu. For these simulations, a custom mesh was developed using the parameters shown in Table 2.6. The meshed geometry used in Simulations A – D is
displayed in Figure 2.7. These parameters were chosen to create a mesh where the number of elements produced is enough to obtain accurate results, but not so large to cause an unreasonably high computation time.

<table>
<thead>
<tr>
<th>Mesh Parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. element size scaling factor</td>
<td>0.8</td>
</tr>
<tr>
<td>Element growth rate</td>
<td>1.45</td>
</tr>
<tr>
<td>Mesh curvature factor</td>
<td>0.8</td>
</tr>
<tr>
<td>Mesh curvature cutoff</td>
<td>0.05</td>
</tr>
<tr>
<td>Resolution of narrow regions</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 2.6: Free mesh parameters for the finite element mesh.

Figure 2.7: The meshed geometry used in Simulations A – D.
After meshing, the finite element problem is ready to be solved. The COMSOL model is saved using the model m-file file format (e.g. SimAinit.m). A structure containing the various aspects of the finite element problem, is exported to MATLAB using the **Export, FEM Structure as ‘fem’** command under the **File** menu.

### 2.4 Model Simulation

The Simulink model *BoneFeedback.mdl* (see Figure 2.8) controls the simulation of the shape and density adaptation. The model reconstructs the 3-D geometry, creates and solves the *fem* file that carries out the finite element analysis (FEA), and processes the results from the FEA to determine how the shape and density will adapt in the analysis cycle. This process repeats for each time step specified by the user. The length of the simulation is specified by indicating a start and stop time in **Configuration Parameters** under the **Simulation** menu. The simulation begins by selecting **Start** from the **Simulation** menu.
Figure 2.8: Simulink model, *BoneFeedback.mdl*, which controls shape and density adaptation.

The model is actually a series of connected blocks, each of which represents a different function. The model input is *XYZ1col*, a column vector that contains the coordinates of the control points. *XYZ1col* is generated by the m-file function *readcontourpts.m* in the pre-simulation steps (see Section 2.3.2). The matrix of control points is fed into the workhorse of the simulation, *msfun_bone.m*, a level-2 m-file s-function (see Figure 2.9). Level-2 m-file s-functions are user-defined Simulink blocks;
the main purposes of this s-function are to set-up and solve the finite element analysis.

S-functions are constrained by a very specific structure. The s-function first issues a setup command, which in turn calls four “registered methods” or subroutines: first DoPostPropSetup, second InitConditions, third Output, and lastly Update. The setup command, DoPostPropSetup, InitConditions and Update subroutines are administrative type functions and will be discussed in the following section. Output contains the bulk of the s-function commands and will be discussed separately in Section 2.4.2.

Figure 2.9: The Simulink model BoneFeedback.mdl with the Level-2 M-file S-function msfun_bone.m highlighted.
2.4.1 S-function Subroutines Setup, DoPostPropSetup, InitConditions, & Update

The parameters of the s-function *msfun_bone.m* are defined during the block set-up (see Section A.2.1, lines 25-56). The number and dimensions of the input and output ports are declared. In this s-function, there is one input and one output port, and the dimensions of both ports are inherited (meaning that the port size is dimensioned to match the input). Other properties of the input and output ports are also inherited. The number of dialog parameters for the s-function is also declared during this section. For *msfun_bone.m*, there are 3 dialog parameters: *npts*, the number of control points; *nslice*, the number of slices in the geometry; and the initial value of time, set at 0. The last part of the set-up specifies the function’s sampling time. For this project, the initial time is zero (0), and time is incremented by one (1) week. After the set-up is complete the function calls the subroutines.

The first subroutine, *DoPostPropSetup*, creates a *Dwork* vector (see Section A.2.1, lines 59-71). A *Dwork* vector creates a discrete variable that can be incremented in the *Update* subroutine. The variable created in this application is used to track the current time. The variable functions similarly to a global variable in that it is accessible in all of the other subroutines, allowing the current time to be called upon during the *Outputs* subroutine. The following subroutine, *InitConditions*, sets the initial value of the *Dwork* vector to the third input parameter of *msfun_bone.m* (see Section A.2.1, lines 74-81).
After completing \textit{DoPostPropSetup} and \textit{InitConditions}, the Outputs subroutine is run, followed by \textit{Update}. The \textit{Update} subroutine simply increments the \textit{Dwork} vector by the time step, which, in this case, is 1 (see Section A.2.1, lines 456-462). On subsequent time steps, the s-function will skip \textit{DoPostPropSetup} and \textit{InitConditions}, and only the Outputs and Update routines will be called.

\textbf{2.4.2 S-function Subroutine Output}

The Output subroutine first assigns variable names to the dialog parameters (\textit{npts} and \textit{nslice}), \textit{Dwork} vector (\textit{t}), and input (\textit{u}) of \textit{msfun\_bone.m}. The zero padding of the input is removed and the input is reshaped into a 2-D matrix of x-, y-, and z-coordinates. The function then begins to construct the finite element problem. After solving the finite element problem, the mechanical signal is extracted from the finite element results, and density adaptation is performed. These steps are outlined in the flowchart shown in Figure 2.10.

\textbf{2.4.2.1 Creating and Solving the fem Structure}

The various aspects of the finite element problem and solution are compiled into a MATLAB structure named \textit{fem}. To set-up and solve a finite element problem in MATLAB, the fem structure must be populated. The majority of the code used to produce the \textit{fem} structure in \textit{msfun\_bone.m} is taken from the model m-file produced by COMSOL (such as SimAinit.m).
Figure 2.10: Flowchart for Section 2.4.2, showing steps to set-up the s-function msfun_bone.m.

The first portion of the model m-file simply identifies which version of COMSOL is being used. This section can be copied directly from the model m-file as shown in Lines 141-149 of msfun_bone.m (see Section A.2.1).

The next portion of the model m-file creates the geometry. However, the geometry needs to be recreated during each time step of the simulation based on the input
to s-function `msfun_bone.m`. To do that, m-file function `creategeom.m` is called instead of using the model m-file code (see Section A.2.1, lines 151-162).

The analyzed geometry, mesh parameters and application mode of the fem structure are all assigned after the geometry segment in the model m-file. These portions can be copied from the model m-file into `msfun_bone.m` as in lines 165-189 (see Section A.2.1).

The following portion of the model m-file defines the boundary and loading conditions. COMSOL assigns the boundary and loading conditions according to the face (or edge or vertex) number. Because the geometry is reconstructed at each time step, the number may or may not remain the same throughout the simulation. Therefore, this portion cannot be copied from the model m-file. To ensure that the boundary and loading conditions are applied to the correct locations at each and every time step, the surface numbers of interest are reevaluated at each iteration of the s-function (see Section A.2.1, lines 191-232). For Simulations A – D, the top surface of the block, where \( z = 0.01 \), is held fixed, so `msfun_bone.m` finds all surfaces whose centroids lie within the plane where \( z = 0.01 \). These surfaces are assigned a boundary condition of 2, meaning fixed, and all other faces are assigned a boundary condition of 1, indicating free. (see Section A.2.1, lines 239-250).

The equations, functions, ordinary differential equations (ODE), units, and solving commands (see Section A.2.1, lines 253-294) are all copied from the model m-file into `msfun_bone.m`, with only one alteration. The equations section of the fem
structure specifies the material properties, so when performing simulations where density remodeling is desired, the density must be specified as:

\[
equ.rho = \text{Dinterpolation}(x,y,z);
\]

The \textit{Dinterpolation.m} function will be discussed in Section 2.4.2.3. If a constant density is desired, the density equation can be left as in the model m-file.

The function portion of the \textit{fem} structure creates the \textit{dentomod} function discussed in Section 2.3.3. The ODE and units part of the \textit{fem} structure indicates that the problem is given in SI units. Finally, the solve command starts the linear solvers used for static finite element problems. The solution is then stored as part of the \textit{fem} structure. The \textit{fem} is converted back to a COMSOL Multiphysics file (*.mph) and saved to the current MATLAB directory with the current time in the name (e.g. Outfem4.mph for time step 4). This allows the finite element problem at each time step to be opened later for post-processing.

### 2.4.2.2 Mechanical Signal Extraction for Shape Adaptation

In order to carry out the mechanically driven shape adaptation, the desired mechanical signals must be extracted from the \textit{fem} structure. Because s-functions limit the inputs and outputs to certain dimensions, the \textit{fem} structure cannot be passed to another block in the Simulink model \textit{BoneFeedback.mdl}. Thus, the mechanical signal extraction is included as part of \textit{msfun_bone.m} (see Section A.2.1, lines 298-366).

This implementation uses local axial strains to motivate remodeling. Because the shape adaptation procedure changes the position of the control points, the only locations where the local axial strain must be evaluated are at the control points. First, the normal
and shear strains in the global coordinate system are interpolated from the fem structure at each of the geometry’s control points. Then the m-file function intercepts.m (see Section A.2.4) is called to determine the local coordinate system and the unit normals at all of the control points. The local coordinate system and the global strains are used to find the local axial strains. Finally, the local axial strains and the unit normals are reshaped into column vectors and concatenated to form the single column vector that is output from msfun_bone.m.

2.4.2.3 Density Adaptation

Density adaptation also requires use of the fem structure and thus is carried out within msfun_bone.m (see Section A.2.1, lines 370-441). The density adaptation in this application is based on the remodeling equilibrium ideas developed by Cowin (1986; 1992) (see Section 1.9). The density adaptation follows the following equations:

\[
\dot{\rho} = (\rho_m f_1 + f_2 \rho) \left( \text{abs} \left( \text{tr}(E) \right) - \text{tr}(E^o) \right)
\]

\[
\rho_{t+1} = \rho_t + \dot{\rho}
\]

Equation 2.1

where \( \dot{\rho} \) is the time rate of change of density, \( \rho_m \) is the material density of cortical bone, \( f_1 \) and \( f_2 \) are constants, \( \rho \) is the structural density, \( \text{tr}(E) \) is the trace of the strain matrix, \( \text{tr}(E^o) \) is the trace of the RE strain, and subscript \( t \) indicates the current time step. Unlike shape adaptation, density adaptation alters the density at every element, not just the control points. Thus, to find the change in density, the normal and shear strains and the current density values are interpolated from the fem structure at the vertices of all mesh
elements, along with the x-, y-, and z-coordinates of the vertices. The coordinates, strains, and densities are averaged across each element to locate the element’s centroid and to approximate strain tensor and density at the centroid. The function then calculates the trace of the strain tensor and the rate of change of density according to Equation 2.1. The change in density and the current density value are combined to determine the density for the next time step. For simulations where mechanically driven shape adaptation is not desired, lines 397-408 in msfun_bone.m are commented out.

Density adaptation can also be specified as a function of time. In this implementation, the current density is simply incremented by a constant value to determine the density at the following time step:

\[ \rho_{t+1} = \rho_t + C \]  

Equation 2.2

where \( \rho \) is the structural density, \( C \) is a constant, and subscript \( t \) indicates the current time step. To remove density adaptation as a function of time in a given simulation, lines 410-412 are commented out.

Once the density for the next time step has been calculated (either using using Cowin’s function or density as a function of time or both), the new density must be stored in such a way as to be accessible in the next time step. Therefore, every element’s centroid coordinates and the new density at the centroid are printed and stored in a text file, \textit{centroid.txt}, for use in the following time step. Then, in the next time step, when the geometry has been remeshed, the function \textit{Dinterpolation.m} (see Section A.2.3) is used to interpolate the density onto the centroids of the new elements. For each new centroid, \textit{Dinterpolation.m} finds the closest old centroid in the text file and assigns the new
centroid the same density value. Because centroid.txt is rewritten at every time step, a
second file is generated with the same data and is named using the current time (e.g.
centroid4.txt for time step 4). This allows the centroid and density data to be recorded
and called upon during post-processing.

During the first time step, the centroid.txt file does not exist since there was no
previous time step. Instead, the MATLAB m-file function createcentfile.m is called
before running the simulation:

```matlab
>> createcentfile(fem);
```

The fem structure exported by COMSOL containing the initial conditions is input into the
function. The createcentfile.m function then uses the fem structure to create a centroid.txt
file using the initial mesh and density values that can be used during the first time step.

Density adaptation as a function of time is demonstrated in Simulation A. The
density increment per time step was chosen using data from Cheng et al. (2005) (see
Section 1.8). Cheng et al. measured the total bone volume (TBV) by week and the total
tissue volume (TTV) by week during bone flap growth. This data combined can provide
the volume fraction of bone by week, which is plotted in Figure 2.11.
Figure 2.11: Graph showing volume fraction by week of bone flap grown in MBG-filled chamber. Data from Cheng et al. (2005).

With the volume fraction of bone by week, the structural density can be calculated. The equation for structural density, $\rho$, in terms of volume fraction, $e$, is:

$$\rho = \rho_m e$$  

Equation 2.3

The structural density by week is plotted in Figure 2.12. A linear trendline was fitted through the data to produce a linear equation and correlation of:

$$\rho = 24.383t + 498.97$$

$$R^2 = 0.9467$$

Equation 2.4
where density is measured in kg/m$^3$ and $t$ is measured in weeks (wk). Therefore, for Simulation A, the initial density was set approximately to the $y$-intercept of the trendline, and the density change per week was set to 24 kg/m$^3$/wk.

![Graph showing structural density by week of bone flap grown in MBG-filled chamber. Data from Cheng et al. (2005).](image)

Figure 2.12: Graph showing structural density by week of bone flap grown in MBG-filled chamber. Data from Cheng et al. (2005).

Mechanically driven density adaptation is employed in Simulations B, D and E. Because the forces seen by the ribs are relatively small, the remodeling equilibrium strain for rib bone is assumed to be 150 $\mu\varepsilon$. The parameters of Equation 2.1 that were used for this implementation are given in Table 2.7. The constants $f_1$ and $f_2$ were selected such that the changes in density would follow the density changes of Simulation A in reverse.
This models how the bone density in the bone flap experiments (Miller et al., 1996; Cheng et al., 2005; Brey et al., 2007) declines after peaking sometime between week 9 and 12.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>material density of cortical bone, $\rho_m$</td>
<td>1830 kg/m$^3$</td>
</tr>
<tr>
<td>constant, $f_1$</td>
<td>4.37 /wk</td>
</tr>
<tr>
<td>constant, $f_2$</td>
<td>-80 /wk</td>
</tr>
<tr>
<td>trace of RE strain, $\text{tr}(E^\circ)$</td>
<td>0.000450</td>
</tr>
</tbody>
</table>

Table 2.7: Parameters for mechanically driven density adaptation used in this implementation.

### 2.4.3 MATLAB Functions *get localaxial* & *get unitnormal*

Because the s-function’s output is limited to the same dimensions as the function’s input, the two outputs from *msfun_bone.m*, the local axial strains and the unit normals, must be concatenated into a single column vector. Two MATLAB function blocks, *get localaxial* and *get unitnormal*, are then used to separate the concatenated vector into individual vectors containing the local axial strains and the unit normals, respectively (see Figure 2.13). The details of the blocks’ definitions can be found in Section A.2.5.
2.4.4 Shape Adaptation

The Simulink blocks for shape adaptation are grouped into a Simulink subroutine called \textit{Shape Adapt} (see Figure 2.15). The subroutine has two inputs: the local axial strains and the unit normals. The axial strain signal is put through a mechanostat function that relates the strain to a bone apposition or resorption rate. The mechanostat theory is explained in Section 1.9.2, and its implementation using the Simulink blocks is explained.
in detail in Section A.2.6.1. The gain block titled *Mech Shape Adapt* turns the mechanically driven shape adaptation on and off. To include the adaptation in the simulation the block is set to 1, and to exclude it from in the simulation, the block is set to 0. The steps required to set-up shape adaptation in *Shape Adapt* are shown in the flowchart of Figure 2.14.

![Flowchart showing steps needed to set-up Shape Adapt and Grow.](image)

Figure 2.14: Flowchart showing steps needed to set-up *Shape Adapt* and *Grow*.

Once the axial strain signal has been converted into a bone apposition/resorption rate, the signal is concatenated with the unit normals before being passed to MATLAB m-file function *calcspeed.m*. This function multiplies the bone formation rate by the unit normals to determine the distance that the control points will be moved in the direction of the unit normal vector. A positive bone formation rate will move the control point along the geometry’s outward normal, whereas a negative bone formation rate will move the control point in the opposite direction. The function *calcspeed.m* outputs the difference
in the x-, y-, and z-coordinates between the control points of the original geometry and
the mechanically driven shape adapted geometry.

Figure 2.15: The Simunlink model *BoneFeedback.mdl* with the subroutine, *Shape Adapt,*
highlighted and expanded. *Shape Adapt* performs the mechanically driven shape
adaptation. The bracketed blocks implement the mechanostat function.
Mechanically driven shape adaptation is used in Simulations C, D and E. To construct the mechanostat function for this application, the maximum rate of bone apposition is assumed to be equivalent to the maximum rate of bone resorption. The maximum rate of bone resorption was calculated using data of bone flaps grown in MBG-filled chambers measured by Cheng et al. (2005). Figure 2.16 shows a plot of the total tissue volume (TTV) in the chamber by week (t). A linear trendline was fit through the data and generated the following linear equation and correlation:

\[
\text{TTV} = -0.12t + 3.86
\]

\[
R^2 = 0.89
\]

where TTV is in cm$^3$, and $t$ is in weeks. Thus, approximately 0.12 cm$^3$ of bone is lost per week, assuming the resorption is linear. Because the bone flap sees no loading, it is assumed that this rate of resorption is the maximum. Considering that the initial volume of the block is 4 cm$^3$, this gives a $V_r$ value of 1.63 %/yr by calculating:

\[
V_r = \frac{\Delta \text{TTV}}{\text{initial volume}} = \frac{-0.12 \text{ cm}^3}{4 \text{ cm}^3} = -0.031 \frac{\%}{\text{wk}} = -1.63 \frac{\%}{\text{yr}}
\]

This value is comparable to the values of human bone formation rates given by Martin et al. (2004). Thus, the maximum resorption and apposition rate is 1.63 %/yr, based on the assumption that maximum resorption and apposition rates are equivalent.
Figure 2.16: Graph showing total tissue volume (TTV) by week of bone flap grown in MBG-filled chamber. Data from Cheng et al. (2005).

However, because this model adapts the shape by altering the position of the geometry’s perimeter, the $V_r$ in units of %/time is not immediately useful. $V_r$ can be converted into units of distance/time by dividing it by the block’s surface area (10 cm$^2$), resulting in a maximum resorption (and apposition) rate of -0.012 cm/wk.

The “lazy zone” was established from 100 $\mu$ε to 200 $\mu$ε using the assumption for the remodeling equilibrium strain of 150 $\mu$ε that was stated in Section 2.4.2.3. The final piece of information needed to construct the mechanostat function is when the bone
formation rates saturate. Because the remodeling equilibrium is relatively low to begin with, it is assumed that the resorption rate does not saturate at low strains. The bone apposition rate is assumed to saturate at 3000 µε, which is the upper limit of physiologic strain levels (Martin et al., 2004). The above calculations and assumptions produce the mechanostat function shown in Figure 2.17.

Figure 2.17: Graph of the mechanostat function used to relate strain to the rate of bone apposition or resorption. The areas where the apposition rate is 0 are often referred to as the “lazy zone”.

2.4.5 Growth

The Simulink block for the non-mechanically driven shape adaptation is grouped into a Simulink subroutine called Grow (see Figure 2.18). The unit normal vectors are
input into the subroutine where they are multiplied by a specified gain (see Figure 2.14). Therefore, the output of the subroutine is the difference in the x-, y-, and z-coordinates between the control points of the original geometry and the non-mechanically driven shape adapted geometry. The non-mechanically driven shape adaptation was not used during these simulations.

Figure 2.18: Simulink model *BoneFeedback.mdl* with the subroutine, *Grow*, highlighted and expanded. *Grow* performs the non-mechanically driven shape adaptation.
2.4.6 Applying the Shape Changes

There are several more steps that BoneFeedback.mdl must take to incorporate the shape adaptations from the mechanically and non-mechanically driven shape adaptations into the geometry of the next time step. First, the outputs from subroutines Shape Adapt and Grow are added together to obtain the total difference in the position of the control points between the initial geometry and the new, shape-adapted geometry.

The output is then passed to an integration block. This block is required because this is a discrete model. For models where the time step is set to 1, the input and output of the block is the same.

The next block is MATLAB m-file function deltaxyz.m (see Section A.2.8). The function reshapes the matrix input so that the coordinate differences are arranged in the same manner as XYZ1col. Finally, an addition block completes the loop by combining the output matrix from deltaxyz.m with XYZ1col to calculate the x-, y-, and z-coordinates of the new geometry’s control points. The new coordinate values are then input into s-function msfun_bone.m to start the next time step. The simulation ends with the time reaches the value set in Simulink’s Configuration Parameters dialog box under the Simulation menu.

2.5 Post-Processing

The finite element problem at each time step is saved as a COMSOL Multiphysics file. Once the simulation is complete these files can be opened and analyzed using COMSOL. When the files are opened, COMSOL will reevaluate any MATLAB
functions used to define material properties. This means that *Dinterpolation.m* will be used to find the density. In order to be sure that *Dinterpolation.m* opens the *centroid.txt* file from the correct time step, line 22 in the function (see Section A.2.3) should be edited before the *.mph* file is opened. The file name should be changed to the corresponding recorded data file (e.g. ‘centroid4.txt for time step 4) so the *Dinterpolation.m* opens it instead of the *centroid.txt*, which has been rewritten at each time step. The COMSOL finite element file can then be opened as normal and analyzed using any of COMSOL’s many post-processing tools.

### 2.6 Methods for Simulation E

Simulation E models the hypothetical situation of how the engineered bone flap may behave after it has been harvested from the chamber at 9 weeks and used in a reconstruction of the maxilla. Because Simulation D models the bone flap’s development in the chamber, the results at time step 9 from Simulation D are used as the initial conditions for Simulation E.

During the bone flap’s development, the overall volume of the block tends to decrease. Therefore, the initial geometry used during Simulation E is the final geometry from Simulation D, which undergoes shape adaptation due to lack of mechanical strain. The control points from Simulation D are saved during each time step by adding a *simout* to Workspace Simulink block just before the s-function (see Figure 2.19). The signal at this location is saved to the MATLAB Workspace under the variable name *simout*. Simulation E uses the control points from time step 9 in Simulation D as the input to the
The Simulink model *BoneFeedback.mdl* with the *simout* to Workspace block highlighted. The block saves the Simulink signal at that location to the MATLAB Workspace.
The initial material properties of Simulation E are the same as in the final step of Simulation D. To use the density from the final step of Simulation D, the centroid and density text file from time step 9 (centroid9.txt) is renamed centroid.txt to set it as the source for the initial density interpolation. This step replaces the calling of function createcentfile.m. The COMSOL function dentomod(rho_sld) is still used to calculate Young’s modulus based on density.

The main force of loading in the maxilla is from biting or chewing. The maximum resultant force in biting is 383 Newtons (N) for females and 545 N for males (Raadsheer et al., 1999). This simulation will use a resultant force of 450 Newtons (N) This force is applied in the negative z-direction to the top surface of the block shown in Figure 2.20.

The loading is distributed along the x-axis as shown in the plot in Figure 2.21. To produce this distribution in COMSOL, the $F_z$ force in the Boundary Settings window is entered in as: $-((x<.02)*(x*1125000)+(x>=.02)*(45000+(x*-1125000)))/.01$. 

Figure 2.20: Initial geometry for Simulation E.
Figure 2.21: Bite load applied to the top surface of the block is distributed along the x-direction of the block.

Boundary conditions are applied to the block’s top edges that run parallel to the y-axis (see Figure 2.22). One side is fixed in the x-, y-, and z-directions, and the other is fixed in only the y- and z-directions to allow deformation in the x-direction. To apply the boundary conditions within the s-function *msfun_bone*, the edge numbers where the boundary conditions are applied must be found. This is done using the same technique used to find the boundary surfaces in Simulations A – D (see Section 2.4.2.1 and Section A.2.1, lines 191-250).
Figure 2.22: Boundary conditions for Simulation E are applied to the right and left top edges. The left top edge is fixed in x-, y- and z-directions and shown in green. The right top edge is fixed in y and blue.

The geometry is meshed using the parameters described in Table 2.6. The initial mesh of the geometry can be seen in Figure 2.23. Finally, Simulation E remodels using mechanically driven shape and density adaptation. The shape remodeling rules used are discussed in Section 2.4.4. The density remodeling is discussed in Section 2.4.2.3.
Figure 2.23: Initial meshed geometry in Simulation E.
CHAPTER 3

RESULTS

3.1 Simulation A

In the first simulation, density is regulated as a function of time, and no shape adaptation is included. The simulation runs for 24 time steps, representing 24 weeks. The initial density is set at 500 kg/m$^3$; the density is then changed using Equation 2.2. At the final step, the density reaches 1076 kg/m$^3$. Table 3.1 summarizes the parameters and settings used to perform Simulation A. Figure 3.1 shows a plot of how density adapts as a function of time. The result of the simulation is displayed in Figure 3.2, which presents cross-sections of the block through the y-z plane with density plotted. Because there is no shape remodeling included, there is no geometry change during the simulation.
<table>
<thead>
<tr>
<th><strong>Simulation A</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial Geometry</strong></td>
<td>4 cm by 1 cm by 1 cm rectangular block</td>
</tr>
<tr>
<td><strong>Physics</strong></td>
<td></td>
</tr>
<tr>
<td>Boundary Conditions</td>
<td>top surface fixed in x-, y-, &amp; z-directions</td>
</tr>
<tr>
<td>Loading Conditions</td>
<td>no loading</td>
</tr>
<tr>
<td><strong>Material Properties</strong></td>
<td></td>
</tr>
<tr>
<td>Initial density (kg/m$^3$)</td>
<td>500</td>
</tr>
<tr>
<td>Density (rho_sld) (kg/m$^3$)</td>
<td>Dinterpolation(x, y, z)</td>
</tr>
<tr>
<td>Young's modulus (E_sld) (Pa)</td>
<td>dentomod(rho_sld)</td>
</tr>
<tr>
<td>Poisson's Ratio</td>
<td>0.33</td>
</tr>
<tr>
<td><strong>Mesh Parameters</strong></td>
<td>see Table 2.4</td>
</tr>
<tr>
<td><strong>Mesh Statistics for Initial Step</strong></td>
<td></td>
</tr>
<tr>
<td>Number of elements</td>
<td>3475</td>
</tr>
<tr>
<td>Number of degrees of freedom</td>
<td>17910</td>
</tr>
<tr>
<td><strong>Remodeling Rules</strong></td>
<td></td>
</tr>
<tr>
<td>Mechanically driven shape change</td>
<td>N/A (gain = 0.0)</td>
</tr>
<tr>
<td>Mechanically driven density change</td>
<td>N/A (commented out)</td>
</tr>
<tr>
<td>Non-mechanically driven shape change</td>
<td>N/A (gain = 0.0)</td>
</tr>
<tr>
<td>Non-mechanically driven density change</td>
<td>see Equation 2.2, where C=24</td>
</tr>
</tbody>
</table>

Table 3.1: Summary of parameters and settings for Simulation A.
Figure 3.1: Plot of structural density by week in simulation A, following a remodeling rule based on linear trendline through data from Cheng et al. (2005).
Figure 3.2: Results from Simulation A, showing a cross-section of the block and plotting density for week 0 (top left), 3 (top right), 6 (mid left), 9 (mid right), 12 (bottom left) and 24 (bottom right).
### 3.2 Simulation B

Simulation B includes mechanically driven density change and no shape adaptation. The simulation runs through 24 time steps, representing 24 weeks. The initial density is set at 1100 kg/m$^3$. The density change follows the rule in Equation 2.1. Table 3.2 summarizes the parameters and settings used during this simulation. The graph in Figure 3.3 shows how the adaptation rule presented in Equation 2.1 adapts the structural density when no strain is present. The results of the simulation are displayed in Figure 3.4, which presents cross-sections of the block through the y-z plane with density plotted. Because there is no shape remodeling included, there is no geometry change during the simulation.
<table>
<thead>
<tr>
<th>Simulation B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial Geometry</strong></td>
</tr>
</tbody>
</table>

**Physics**

| Boundary Conditions | top surface fixed in x-, y-, & z-directions |
| Loading Conditions | no loading |

**Material Properties**

| Initial density (kg/m$^3$) | 1100 |
| Density (rho_sld) (kg/m$^3$) | Dinterpolation(x, y, z) |
| Young's modulus (E_sld) (Pa) | dentomod(rho_sld) |
| Poisson's Ratio | 0.33 |

**Mesh Parameters**

see Table 2.4

**Mesh Statistics for Initial Step**

| Number of elements | 3475 |
| Number of degrees of freedom | 17910 |

**Remodeling Rules**

| Mechanically driven shape change | N/A (gain = 0.0) |
| Mechanically driven density change | see Equation 2.1 and parameters in Table 2.7 |
| Non-mechanically driven shape change | N/A (gain = 0.0) |
| Non-mechanically driven density change | N/A (commented out) |

Table 3.2: Summary of parameters and settings for Simulation B.
Figure 3.3: Plot of structural density by week according to Cowin’s remodeling rule (1986; 1992) in the special case, like Simulation B, when no strain is present.
Figure 3.4: Results from Simulation B showing a cross-section of the block and plotting density for week 0 (top left), 3 (top right), 6 (mid left), 9 (mid right), 12 (bottom left) and 24 (bottom right).
3.3 Simulation C

Simulation C demonstrates the mechanically driven shape adaptation (see Figure 3.5). The block’s shape is adapted using the mechanostat function discussed in Section 2.4.4 and shown in Figure 2.17. Table 3.3 summarizes the parameters and settings used to create the simulation. Figure 3.6 and Figure 3.7 illustrate how the block’s cross-sectional shape along a y-z plane and an x-z plane, respectively, changes during the adaptation. Finally, the plot in Figure 3.8 compares the block’s geometric volume during the simulation with the total tissue volume (TTV) as measured by Cheng et al. (2005). Density remains constant during the simulation since density changes are not included.

While the simulation was set to run for 24 time step, representing 24 weeks, problems with the geometry construction and collapsing surface caused the simulation to stop after 11 time steps. See Section 4.3 for a description of the collapsing surfaces limitation.

Figure 3.5: The initial geometry (left) and the geometry after 9 weeks (right) in Simulation C.
## Simulation C

<table>
<thead>
<tr>
<th><strong>Initial Geometry</strong></th>
<th>4 cm by 1 cm by 1 cm rectangular block</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Physics</strong></td>
<td></td>
</tr>
<tr>
<td>Boundary Conditions</td>
<td>top surface fixed in x-, y-, &amp; z-directions</td>
</tr>
<tr>
<td>Loading Conditions</td>
<td>no loading</td>
</tr>
<tr>
<td><strong>Material Properties</strong></td>
<td></td>
</tr>
<tr>
<td>Initial density (kg/m^3)</td>
<td>800</td>
</tr>
<tr>
<td>Density (rho_sld) (kg/m^3)</td>
<td>800</td>
</tr>
<tr>
<td>Young's modulus (E_sld) (Pa)</td>
<td>dentomod(rho_sld)</td>
</tr>
<tr>
<td>Poisson's Ratio</td>
<td>0.33</td>
</tr>
<tr>
<td><strong>Mesh Parameters</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>see Table 2.4</td>
</tr>
<tr>
<td><strong>Mesh Statistics for Initial Step</strong></td>
<td></td>
</tr>
<tr>
<td>Number of elements</td>
<td>3475</td>
</tr>
<tr>
<td>Number of degrees of freedom</td>
<td>17910</td>
</tr>
<tr>
<td><strong>Remodeling Rules</strong></td>
<td></td>
</tr>
<tr>
<td>Mechanically driven shape change</td>
<td>Mechnostat Function, see Figure 2.17</td>
</tr>
<tr>
<td>Mechanically driven density change</td>
<td>N/A (commented out)</td>
</tr>
<tr>
<td>Non-mechanically driven shape change</td>
<td>N/A (gain = 0.0)</td>
</tr>
<tr>
<td>Non-mechanically driven density change</td>
<td>N/A (commented out)</td>
</tr>
</tbody>
</table>

Table 3.3: Summary of parameters and settings for Simulation C.
Figure 3.6: Results from Simulation C showing a cross-section of the block through a y-z plane for week 0 (top left), 3 (top right), 6 (bottom left), and 9 (bottom right).
Figure 3.7: Results from Simulation C showing a cross-section of the block through an x-z plane for week 0 (far left), 3 (mid left), 6 (mid right), and 9 (far right).
Figure 3.8: Plot comparing the measured total tissue volume (TTV) by week of bone flap grown in MBG-filled chambers (Cheng et al., 2005), plotted with circles, and the geometric volume during finite element Simulation C, plotted with squares. Note that, although not specifically measured, TTV at time zero is approximately 4 cm$^3$ since the chamber was filled with MBG.
3.4 Simulation D

Simulation D represents the bone flap’s behavior during development within the chamber. Both mechanically driven shape and density adaptations are included. The parameters and settings used in the simulation are summarized in Table 3.4. Figure 3.9 shows the shape changes in cross-sections of the block through a y-z plane. Figure 3.10 shows the changes in the cross-sections through an x-z plane. The density at each time step is also plotted in Figure 3.9 and Figure 3.10. Again, the simulation was set to run for 24 time steps, representing 24 weeks, but problems with geometry construction and collapsing surfaces caused the simulation to stop after 11 time steps. See Section 4.3 for a description of the collapsing surfaces limitation.
<table>
<thead>
<tr>
<th><strong>Simulation D</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial Geometry</strong></td>
</tr>
<tr>
<td><strong>Physics</strong></td>
</tr>
<tr>
<td>Boundary Conditions</td>
</tr>
<tr>
<td>Loading Conditions</td>
</tr>
<tr>
<td><strong>Material Properties</strong></td>
</tr>
<tr>
<td>Initial density (kg/m^3)</td>
</tr>
<tr>
<td>Density (rho_sld) (kg/m^3)</td>
</tr>
<tr>
<td>Young's modulus (E_sld) (Pa)</td>
</tr>
<tr>
<td>Poisson's Ratio</td>
</tr>
<tr>
<td><strong>Mesh Parameters</strong></td>
</tr>
<tr>
<td><strong>Mesh Statistics for Initial Step</strong></td>
</tr>
<tr>
<td>Number of elements</td>
</tr>
<tr>
<td>Number of degrees of freedom</td>
</tr>
<tr>
<td><strong>Remodeling Rules</strong></td>
</tr>
<tr>
<td>Mechanically driven shape change</td>
</tr>
<tr>
<td>Mechanically driven density change</td>
</tr>
<tr>
<td>Non-mechanically driven shape change</td>
</tr>
<tr>
<td>Non-mechanically driven density change</td>
</tr>
</tbody>
</table>

Table 3.4: Summary of the parameters and settings for Simulation D.
Figure 3.9: Results from Simulation D showing a cross-section of the block through a y-z plane and plotting density for week 0 (top left), 3 (top right), 6 (bottom left), and 9 (bottom right).
Figure 3.10: Results from Simulation D showing a cross-section of the block through an x-z plane and plotting density for week 0 (top left), 3 (top right), 6 (bottom left), and 9 (bottom right).
3.5 Simulation E

Simulation E predicts how the bone flap may theoretically behave if it were harvested after 9 weeks and used in the reconstruction of the maxilla. The results from time step 9 of Simulation D are used as this simulation’s initial conditions. Both mechanically driven shape and density adaptations are included. This simulation encounters the same collapsing surfaces problem seen in Simulations C and D, causing the simulation to stop after 9 time steps, representing 9 weeks. See Section 4.3 for a description of the collapsing surfaces limitation. Table 3.5 summarizes the parameters and settings used to produce Simulation E. Figure 3.11 displays a 3-D view of the geometry as the shape is adapted. Slice plots showing density during the simulation are presented in Figure 3.12 and Figure 3.13. Finally, the trace of the strain tensor is plotted on the deformed geometry in Figure 3.14 and Figure 3.15. Because the strains in the initial time step are two orders of magnitude larger than the strains in later weeks, the plots are displayed separately and use different color scales.
<table>
<thead>
<tr>
<th><strong>Simulation E</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial Geometry</strong></td>
</tr>
<tr>
<td><strong>Physics</strong></td>
</tr>
<tr>
<td>Boundary Conditions</td>
</tr>
<tr>
<td>Loading Conditions</td>
</tr>
<tr>
<td><strong>Material Properties</strong></td>
</tr>
<tr>
<td>Initial density (kg/m^3)</td>
</tr>
<tr>
<td>Density (rho_sld) (kg/m^3)</td>
</tr>
<tr>
<td>Young's modulus (E_sld) (Pa)</td>
</tr>
<tr>
<td>Poisson's Ratio</td>
</tr>
<tr>
<td><strong>Mesh Parameters</strong></td>
</tr>
<tr>
<td><strong>Mesh Statistics for Initial Step</strong></td>
</tr>
<tr>
<td>Number of elements</td>
</tr>
<tr>
<td>Number of degrees of freedom</td>
</tr>
<tr>
<td><strong>Remodeling Rules</strong></td>
</tr>
<tr>
<td>Mechanically driven shape change</td>
</tr>
<tr>
<td>Mechanically driven density change</td>
</tr>
<tr>
<td>Non-mechanically driven shape change</td>
</tr>
<tr>
<td>Non-mechanically driven density change</td>
</tr>
</tbody>
</table>

Table 3.5: Summary of the parameters and settings for Simulation E.
Figure 3.11: Results from Simulation E showing the change in shape at week 0 (top left), week 3 (top right), week 6 (bottom left) and week 9 (bottom right).
Figure 3.12: Results from Simulation E showing a slice plot of density at week 0 (top) and week 3 (bottom).
Figure 3.13: Results from Simulation E showing a slice plot of density at week 6 (top), and week 9 (bottom).
Figure 3.14: Result from Simulation E plotting the trace of the strain tensor on the deformed geometry at week 0. Deformation has a scale factor of 1. The trace of the strain tensor is the mechanical input for density remodeling.
Trace of Strain Tensor at Week 3

Trace of Strain Tensor at Week 6

Trace of Strain Tensor at Week 9

Figure 3.15: (captioned on next page.)
Figure 3.15: (shown on previous page) Results from Simulation E plotting the trace of the strain tensor onto the deformed geometry at week 3 (top), week 6 (middle), and week 9 (bottom). Deformation has a scale factor of 1. The trace of the strain tensor is the mechanical input for density remodeling.
CHAPTER 4

DISCUSSION

4.1 Comments on the Results

The finite element modeling technique presented in this thesis facilitates four means of adaptation that can be used alone or in any combination. Simulations A through C are purposefully designed to be simple to demonstrate that these techniques function as expected. Simulations D and E use the technique in an actual application.

Simulations A and B show that both of the density adaptation rules are very robust. The change in density was simulated for 24 weeks without problem and produced the expected results. Simulation A is assigned a low uniform initial density. Because the density in Simulation A is governed as a function of time, the density increases at a uniform rate, and the density remains uniform at each time step of the simulation (see Figure 3.1 and Figure 3.2).

The bone flap’s initial increase in bone density is most likely a wound healing response. This simulation attempts to model how density may change during a wound healing response. In a wound healing situation, the bone density increases due to a cascade of cellular events, such as an increase in bone morphogenic proteins, which can be described as function of time.

91
Simulation B is given a high initial density, which again is uniform throughout the geometry. The density adaptation is a function of the current density and the trace of the strain tensor, a mechanical signal that changes based on the geometry’s loading. However, since no forces are applied, there are no strains, and the trace of the strain tensor equals zero at every time step. Therefore, the rate of density change during Simulation B is actually only a non-linear function of the current density. At each time step, the density is uniform throughout the geometry, but the change in density at each time step varies (see Figure 3.3 and Figure 3.4).

Mechanically driven shape adaptation is demonstrated during Simulation C. The geometry begins as a rectangular block. The shape adaptation follows the mechanostat function using local axial strain as the driving signal. Because the block is not loaded, the geometry is adapted by moving the control points inward to create a smaller perimeter, and thus a shrinking volume (see Figure 3.5). This simulation ran successfully for 11 time steps before running into problems constructing the geometry due to collapsing surfaces. This issue is discussed in greater detail in Section 4.3.

Simulations B and C try to model how the bone flap is affected by bone resorption caused by disuse. When the bone flap chamber is first implanted, new bone is formed caused by a wound response. As the wound response ends, the bone in the chamber remains unloaded, and the bone in the chamber is resorbed. Simulation B replicates bone resorption by simulating the decreasing bone density, while Simulation C attempts to model bone resorption by shrinking the overall volume (see Figure 3.8). In
reality however, the bone resorption is a combination of lowered density and reduced volume.

Simulation D employs both mechanically driven shape and density adaptation to develop a more accurate representation of bone resorption. The simulation again runs for 11 time steps before encountering the same problem with collapsing surfaces as in Simulation C. As there is still no loading applied and the strains are zero, the result of Simulation D is simply a superposition of results from Simulations B and C (see Figure 3.9 and Figure 3.10). In models with loading, the strains will vary with the changes in shape and density, and thus both the shape and density adaptation will be dependent upon each other.

To demonstrate this interplay between shape and density adaptation, Simulation E presents a hypothetical situation where the bone flap is harvested from the chamber and used to reconstruct the maxilla bone. The initial geometry and material properties for Simulation E should represent the bone after harvesting from the chamber after 9 or 12 weeks. The geometry and material properties of Simulation D at 9 weeks were used to do this.

The loading on a normal maxilla comes primarily from biting and chewing, so the bone flap is loaded with a face load in the z-direction that approximates the bite force. To represent how the bone flap may be constrained when implanted, the edges on the ends of the bone flap were fixed. Ideally, the faces or surfaces on the ends of the bone flap would be used for fixation instead; however, difficulties in applying boundary
conditions to faces undergoing shape adaptation prevent it. This limitation is discussed further in Section 4.3.

The simulation runs for 9 time steps, representing 9 weeks, before stopping due to problems with collapsing surfaces. Initially, the applied force produces large deformations and strain magnitudes that are larger than the RE strain (see Figure 3.14). Because the strain magnitude varies over the geometry, the rates of change for both density and shape vary as well. This produces a non-uniform density distribution and an irregularly shaped geometry. Furthermore, because the shape and density remodeling rates are proportional to the strain, the large strains produce sizeable changes between time steps 0 and 1. While excessively high strains will saturate the shape adaptation rate, there is no saturation built into the density adaptation rule. This may result in density increases that are larger than are physiologically possible.

The strain levels are two orders of magnitude lower after the initial time step. Overall, the density generally increases from the initial 818 kg/m$^3$ to within the 900-1600 kg/m$^3$ range (see Figure 3.12 and Figure 3.13). The volume of the block also increases as the control points are general moved outward (see Figure 3.11). However, the strain levels are fairly low on the unfixed edges at the ends of the block because the surface is free to displace instead of deforming. The low strains move the control points inward and result in volume loss in that region. The simulation stops after 9 time steps because the inward moving control points cause surfaces to collapse again.
4.2 Validity of the Results

When performing computational analyses, the results must be validated with experimental data. Access to experimental data relevant to the applications in this thesis is limited to the published data on bone flap growth (Miller et al., 1996; Cheng et al., 2005; Brey et al., 2007). The simulation parameters were selected to model the behavior of bone flaps grown by Cheng et al. (2005) in chambers filled with autologous morselized bone graft (MBG). Available data includes total tissue volume (TTV), total bone volume (TBV), and tissue cross-sectional area at 3-week intervals.

From the TTV and TBV data, the structural density can be calculated and used to approximate the parameters used the density adaptation rule. The structural density calculations assumed that any bone present had a density equal to that of cortical bone (1,830 kg/m$^3$) and that areas of non-bone tissue did not contribute to the structural density. Actual measurements of bone flap structural density and the material density of the bone within the flap may lead to more accurate density remodeling rules.

In these simulations, the modulus of elasticity, or Young’s modulus, is input as a function of structural density. Researchers have proposed many functions to describe the relationship between density and Young’s modulus, but none are widely agreed upon. The function used in these simulations was chosen because of the large number and wide range of experimental samples it was based upon. The selected function is able to approximate Young’s modulus for a wide range of densities. To obtain more accurate results, mechanical testing of the bone flaps would be needed. Measurement of the
material density of the bone tissue in the bone flaps, as suggested above, would also result in better estimations of the bone flap’s Young’s modulus.

The parameters used in the mechanically driven shape remodeling rule were also derived using data from Cheng et al. (2005). The change in TTV was used to determine the maximum bone resorption rate under the assumption that volume change in the x- and y-directions was uniform. Dimensions of the bone flap at each time step, instead of just cross-sectional areas, would be needed to validate this assumption. In addition, this modeling technique is not capable of producing shape adaptation in the z-direction. This limitation of the model produces the difference in TTV and geometric volume seen in Figure 3.8, as the computational model does not capture the bone flap’s diminishing z-dimension seen during experimental studies.

The adaptation models presented here assume that the material is a continuum. However, in reality, the bone flap is composite of a variety of tissues and materials, which probably include blood, extracellular fluid, bone tissue, and fibrous tissue. These materials will be distributed differently and be of different proportion in each bone flap. The material properties of each material will contribute to the overall properties of the bone flap in ways that are not examined in this thesis. These simulations also assume that the bone flaps are isotropic, meaning that the material properties are the same in all directions. This assumption is likely untrue, given the composite natural of the bone flap. However, assuming isotropy simplifies equations and reduces the number of material properties to be determined. An orthotropic description may be more accurate since bone, as a material, has been shown to be orthotropic. However, because the bone flap is
not solid bone, an orthotropic description may not be entirely accurate either. Further mechanical testing of the bone flaps may help to establish which material description is the best representation.

The simulations in this thesis also assume that the bone flap deforms elastically. Cortical bone is considered an elastic material, however cancellous bone consists of pores often filled with fluid and is best modeled as a viscoelastic material. The presence of a liquid portion in the composite of materials that make up the bone flap would indicate that the bone flap would be more accurately modeled as a viscoelastic material.

Viscoelastic models are cumbersome in COMSOL Version 3.4 and are computationally intensive. Mechanical testing of the bone flap could determine if it could be reasonable modeled as an elastic tissue, or if too much accuracy would be sacrificed for the sake of simplicity. Since only Simulation E experiences any deformation, it is the only simulation affected by this assumption.

4.3 Limitations of the Model

The computational techniques used in these simulations have several limitations, including the control point method of geometry construction and significant computation time. The control points are essential to the geometry construction and shape remodeling. Unfortunately, the use of the control points in this modeling method creates two weaknesses that cause the simulations to crash.

During the function creategeom.m, splines connect the control points of each slice. Unfortunately, the method of geometry construction and shape adaptation was
originally developed for tubular long bones (Roberts and Hart, 2005), and continuous splines do not capture corners well, producing rounded corners instead. When the control points are spaced far apart, the radius of the rounded corner can be rather large. Closer spacing between control points reduces the radius of the corner but does not eliminate it. So while the splines create a good representation of the cylindrical bone shafts that were previously modeled using this technique (Liu and Hart, 2005; Liu et al., 2005; Roberts and Hart, 2005), the splines only approximate the rectangular block geometry in this application.

Still, the current method of shape adaptation is dependent upon the splines and the rounded corners they create. The shape adaptation technique relies on knowing the unit normal vector. The unit normal vector is calculated using the tangent of each control point. Finding a unique tangent vector requires a continuous surface, meaning a unique tangent vector cannot be identified at a corner. Corners can develop in the geometry as the control points are moved inward (see Figure 4.1). When the tangents cannot be found, the unit normal is output from s-function msfun_bone.m as NaN, or “not a number”. This produces errors in the calculation of Simulink subroutines Shape Adapt and Grow and causes the simulation to abort.
Figure 4.1: As control points are moved inward, corners may develop, leading to the inability to identify a unique tangent vector.

The control point method of geometry construction can also lead to problems with collapsing surfaces. The splines connect the control points in a specified order. When the control points are moved a particular distance inward, the spline connecting the points may overlap itself (see Figure 4.2). This creates problems during geometry construction. Function `creategeom.m` converts the splines to solid objects and calls a loft function to connect the solid object of each slice to form a solid 3-D geometry object. The loft function requires each solid object to contain the same number of surfaces. When the overlapping spline is converted to a solid object, the looped segment is removed, and the number of surfaces on that solid object is reduced. An error is reported stating that solids with differing surfaces cannot be lofted, and the simulation aborts. In summary, too large of a shape adaptation will collapse a surface and cause the simulation to fail. Note however, that this problem does not exist for geometries without sharp edges.
Moving the control points only small distances at each time step may allow the simulation to run longer, but if the points continue to move inward, eventually the spline will begin to overlap. If the control points are spaced further apart, the distance the points can move inward before causing problems is larger. However, spacing the control points further apart also creates a larger radius on the rounded corner.

This method of modeling requires a significant amount of computation time. At each time step, the geometry must be re-generated and re-meshed. The geometry is input into a new finite element problem that must be solved. While the time to complete a single time step varies, mostly due to the number of degrees of freedom, the solution time on average is between one to two minutes (running on an iMac 2.16 GHz Intel Core 2 Duo). However, shape adaptation often creates additional curvature in the geometry and the meshing algorithm increases the mesh density in these locations (compare Figure 2.7 and Figure 2.23). The difference in the mesh density can significant raise the number of degrees of freedom (e.g. from 17,910 in Simulations A – D to 61,602 in Simulation E).
This translates into longer solving times (4-5 minutes per time step) for geometries with large shape adaptations.

The m-file function *Dinterpolation.m*, which is used to interpolate density, also increases the computation time. The function must refer to a text file to obtain the stored density data. In post-processing, the file name must be changed within the function before opening the COSMOL file for analysis. This process is rather cumbersome, but it allows density to be interpolated onto geometries that may grow beyond a specified grid.

Another limitation of the model is that the last slice of the geometry does not remodel. In this application, the last slice represented the surface attached to the periosteum. Because the periosteum supports the bone flap and supplies its nutrients, resorption along this surface is not substantial, so the limitation does not greatly affect the results. However, limiting the remodeling of the last slice limits the application of this technique in other scenarios.

COMSOL identifies which surfaces and edges are to receive boundary and loading conditions by assigning each surface and edge a number. The numbering system is embedded as an algorithm within the COMSOL solvers and is not predictable from one model to the next. In this modeling technique, the remodeled geometry must be reconstructed at every time step, which also changes the surface and edge numbering. Therefore, the surfaces and edges that are to be loaded or fixed must be redetermined at each iteration. To do this, the coordinates of each surface and edge are compared to the locations to be loaded or fixed, thereby requiring any loaded or fixed surface or edge to be stationary in at least one direction. This requirement prohibits the surfaces on the ends...
of the bone flap geometry to be fixed during Simulation E. The surfaces are able to undergo shape adaptation, meaning that the location of the surface may change at any time step.

4.4 Modeling Advances

The technique presented here combines density and shape adaptation in the same 3-D model. Changes in either shape or density affect the strain magnitudes, and changes in the strain magnitudes affect shape and density remodeling. This creates interesting interactions between shape, density and strain levels. The previous generation of this modeling technique (Liu and Hart, 2005; Liu et al., 2005) enabled 3-D shape adaptation or 3-D density adaptation. However, combined density and shape adaptation was limited to 2-D.

The COMSOL software includes several interpolation functions, but each require the data to be interpolation to be in a grid format. These functions work well for shapes smaller than the grid volume or area, but for geometries that extend past (or grow past) the grid, the interpolation becomes a less reliable extrapolation. The m-file function Dinterpolation.m was developed to circumvent this issue. Dinterpolation.m uses the centroids from the previous geometry as the source of interpolation data. This eliminates the grid requirement and allows the size of the interpolation source to change with each time step. Currently, Dinterpolation.m uses a closest-value type interpolation to assign a density value to each vertex in the mesh.
In addition, the level-2 m-file s-function in this version of the Simulink model was converted from a level-1 s-function used by Liu (Liu and Hart, 2005; Liu et al., 2005). While MATLAB still runs level-1 s-functions, there is no longer much support for the format. Also, while the syntax between the s-functions is different, their operations are nearly identical.

The m-file function \textit{slopes.m} replaced a standard matrix gain Simulink block. The new function enables different strain relationships for bone resorption and apposition rates (i.e. the slopes of the mechanostat function can be different).

### 4.5 Future Studies and Improvements

Simulations A – D all model the bone flap’s behavior inside the PMMA chamber. In experimental studies, the chamber walls limit the bone flap’s volume expansion, and the bone flap’s shape is molded by the chamber. However, the computational model does not include a way to constrain the volume expansion of the geometry. While this is not an issue given the simulations presented here, modeling a bone flap while in the chamber using shape adaptation which causes the volume will produce incorrect results if the volume is allow to expand beyond the chamber dimensions. Including constraints to represent the chamber could introduce solid-solid interactions and would be an interesting model to pursue in future studies.

A new method to construct the geometry is needed to eliminate the geometry limitations discussed in Section 4.3. The problems with collapsing surfaces and corners are embedded in the current method. Future models also should include a way to enable
shape adaptation in the z-direction. Ideally, the new method would allow remodeling along each surface to be turned on and off in order to better control shape adaptation.

The ability to turn shape adaptation of a specific surface on or off would allow the geometry to be sliced in different directions. In these examples, the geometry is sliced along x-y plane in order to provide a non-adapting top surface to place boundary and loading conditions. If the geometry is sliced along the x-z or y-z planes, the surface representing the periosteum contains control points and will adapt. However, if the modeling method included a way to turn shape adaptation of a specific surface on or off, the shape adaptation on the surface representing the periosteum could simply be turned off. In addition, slicing along the x-z or y-z planes may enable shape adaptation in the z-direction.

Future models may also consider an Eulerian approach instead of the current Lagrangian one. Using control points to physically shrink the geometry during shape adaptation creates problems with collapsing surfaces and in identifying necessary tangent vectors. Rather than using control points to direct shape adaptation, it may be possible to describe the volume loss using material properties. Perhaps the disappearing volume can be assigned a density of zero (or close to it) while the actual analyzed geometry size remains the same. This would create a sort of density front that would represent the shape changes without the computational problems of the current shape adaptation method.

More accurate models could be created with more detailed data from the bone flaps. Precise dimensions from the bone flap would enhance the shape adaptation
simulations. Mechanical testing of the bone flap’s composition and deformation behavior may provide insight on which material model is best suited. Mechanical testing could also determine the actual material constants to input into the model. Furthermore, understanding how the bone flap’s mechanical properties change week by week while in the chamber would enable more precise density and shape adaptation, and may generate an awareness of when or if density adaptation should saturate.

Finally, future experimental studies might focus on how harvested bone flaps respond to loading. A substantial bone flap will not develop without some type of mechanical loading, be it during growth in the chamber or after harvesting. Studying the shape and density changes that occur in a loaded bone flap is essential to validating models of the harvested bone flap, like Simulation E.
CHAPTER 5

CONCLUSION

The finite element modeling method presented in this thesis is a powerful tool that can be used to simulation the shape and density remodeling that occurs during the development of bone flaps. Simulating bone flap behavior provides insight into possible ways to make this experimental procedure a successful alternative to autografts and allografts. Since the bone that is formed in the chamber is most likely caused by a wound response, the best time to harvest the bone flap would be as the wound response is ending and before resorption begins. Loading must be applied to the bone flap in order to increase the overall quality of the bone.

The five simulations presented demonstrate that the model is capable of reproducing the shape and density changes of the bone flap in the chamber. However, geometry issues, some of which cause the simulations to crash, limit the model’s application. In addition, more mechanical testing of the bone flaps is necessary to provide better input parameters to the model. Still, more accurate material parameters and a more adaptable geometry construction may lead to the ability to not just simulate bone flap behavior, but to predict it.
APPENDIX A

MATLAB M-FILE FUNCTIONS AND SUBROUTINES
This appendix contains all the user-defined functions used in this thesis and is divided into two parts: Pre-Simulation Routines and Simulation Routines. Table A.1 displays a list of the user-defined MATLAB functions and a brief description of each.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filesforblock</td>
<td>(s) Creates text files containing control points for geometry</td>
</tr>
<tr>
<td>readcontourpts</td>
<td>(F) Reads text files containing control points for geometry</td>
</tr>
<tr>
<td>readdatafile</td>
<td>(F) Reads individual text file containing control points of a single slice</td>
</tr>
<tr>
<td>creategeom</td>
<td>(F) Uses control points to construct solid geometry object</td>
</tr>
<tr>
<td>createcentfile</td>
<td>(F) Creates text file containing initial centroid and density data</td>
</tr>
<tr>
<td>msfun_bone</td>
<td>(SF) Creates and solves the finite element problem, extracts mechanical data, and performs density adaptation</td>
</tr>
<tr>
<td>Dinterpolation</td>
<td>(F) Interpolates density over geometry using centroid.txt file</td>
</tr>
<tr>
<td>intercepts</td>
<td>(F) Finds unit normal, unit tangent, and axial vectors at control points</td>
</tr>
<tr>
<td>findcontrolpts</td>
<td>(F) Finds coordinates of control points using splines</td>
</tr>
<tr>
<td>findtangents</td>
<td>(F) Finds unit tangents of control points</td>
</tr>
<tr>
<td>drawclosedspline</td>
<td>(F) Draws closed spline through control points</td>
</tr>
<tr>
<td>contoursegintercept</td>
<td>(F) Finds points on next spline closest to each control point</td>
</tr>
<tr>
<td>perpdist3d</td>
<td>(F) Constructs vector perpendicular to tangent to closed point</td>
</tr>
<tr>
<td>samplebspline</td>
<td>(F) Finds next set of control point coordinates along spline</td>
</tr>
<tr>
<td>slopes</td>
<td>(F) Defines slopes of mechanostat function</td>
</tr>
<tr>
<td>calcspeed</td>
<td>(F) Determines distance to move control points for shape adaptation</td>
</tr>
<tr>
<td>deltaXYZ</td>
<td>(F) Prepares shape adaptation matrix to line up with control point input</td>
</tr>
</tbody>
</table>

Table A.1: List of user-defined MATLAB functions where (s) – script, (F) – function, (SF) – S-function.
A.1 Pre-Simulation Routines

This section contains m-file functions and scripts used to set-up the simulations.

Some of the functions (such as creategeom.m) are used again during the simulations.

A.1.1 M-file Script: filesforblock.m

The m-file script filesforblock.m creates the geometry text files for a rectangular block. The section labeled “Control Variables” contains the parameters to define the number of control points along the x- and y-directions, the number of slices to create, the dimensions of the block, the name of the files, and an option to plot.

```matlab
%filesforblock.m
% This script creates the data files needed for BoneFeedback.mdl Creates an array of points that are used to create a block. The points are compiled in matrix XYZacross and then distributed into data files, each containing a slice of the geometry. The size of the block is controlled using height, width and depth. The density of data points is controlled using xnpts, ynpts, and nslice. Plotting of the points is controlled with plotyn. The program produces nslice-number of data files. File name is determined by character string fname. The block's lower front left corner is always positioned at the origin, (0,0,0).
% Written by R. Jansen (2008).

clear all

% -------- Control Variables -------- %
xnpts=21; % number of data points along x-direction
ynpts=6; % number of data points along y-direction
nslice=6; % number of slices (number of points in z-dir)
width=.04; % width of the block, distance along x-axis
depth=.01; % depth of the block, distance along y-axis
```
height=.01; % height of the block, distance along z-axis
plotyn=1; % 1=create plot, 0=do not create plot
fname='bone'; % character string for prefix of file name

% how far apart points are in the x-direction
xspace=width/(xnpts-1);
% how far apart points are in the y-direction
yspace=depth/(ynpts-1);
% how far apart points are in the z-direction
zspace=height/(nslice-1);
% the number of points in one slice/data file
npts=xnpts*2+ynpts*2-4;
% XYZacross will contain all data points
XYZacross=zeros(npts,3,nslice);

% Because the block is uniform, the x- and y-coordinates % will be the same on all slices and only the z-coordinate % will increment.
for i=1:nslice
    % fill in the x-coordinates
    XYZacross(xnpts:xnpts+ynpts-1,1,i)=width;
    XYZacross(xnpts*2+ynpts-2:npts,1,i)=0;
    for j=0:xnpts-1
        XYZacross(j+1,1,i)=j*xspace;
        XYZacross(j+xnpts+ynpts-1,1,i)=width-j*xspace;
    end

    % fill in the y-coordinates
    XYZacross(1:xnpts,2,i)=0;
    XYZacross(xnpts+ynpts-1:xnpts*2+ynpts*2-2,2,i)=depth;
    for k=0:ynpts-1
        XYZacross(k+xnpts,2,i)=k*yspace;
        XYZacross(k+xnpts*2+ynpts-2,2,i)=depth-k*yspace;
    end

    % fill in z-coordinates
    XYZacross(:,3,i)=((i-1)*zspace);

    % plot points if desired
    if plotyn==1
        hold on
        plot3(XYZacross(:,1,i), XYZacross(:,2,i), ...
             XYZacross(:,3,i))
        grid on
    end

A.1.2 M-file Function: readcontourpts.m

Before the simulation BoneFeedback.mdl can be run in Simulink, MATLAB M-file function readcontourpts.m must be used to read the control point coordinates from stored text files. This function uses readdatafile.m (see Section A.1.3).

```matlab
function [XYZ1col, npts]=readcontourpts(fileprefix,...
    startslice, slicestep, nslice)
%readcontourpts.m
% This function reads in the x, y, z coordinates from a
% series of data files. Each data file contains
% coordinates for a single slice of a geometry object.
% The data file names should all have the same prefix,
% followed by a numerical indicator of the slice number
% with constant incrementing (see below for example).
% The points on each slice can be connected with splines,
% then each slice’s spline can be combined to recreate the
% geometry object. This is done in creategeom.m
```
This function uses user created function: readdatafile.m

Inputs are: fileprefix, the character string of the file name; startslice, the file suffix that begins the series of data files; slicethick, the increment between the data file suffixes; and nslice, the total number of files in the series of data files.

For example, the data files might be tube_0000, tube_0100, and tube_0200. Then the inputs would be as follows: fileprefix='tube_', startslice=0000, slicestep=100, nslice=3.

Outputs are: XYZ1col, a (npts*4*(nslice-1))-by-1 array where the x, y, and z coordinates of the (npts*3) data points are listed in a single column (i.e. [x1;y1;z1;x2;y2;z2;..]), followed by a series of zeros to fill the rest of the matrix (see below); and npts, the number of points in a single data file (containing one slice).

Written by R. Jansen (2008).

Clear variables

for j=0:nslice-1  % for each slice/data file
    % Call readdatafile function, and put output in matrix % named XYZacross. XYZacross is a 3D % npts-by-3-by-nslice matrix.
    [XYZacross(:,:,j+1),npts]=readdatafile([fileprefix ...
        num2str(startslice+(j*slicestep), '%04.0f')));

    % Rearrange points into XYZdown, a 3-by-npts-by-nslice sized matrix.
    XYZdown(:,:,j+1)=XYZacross(:,:,j+1)';
end

% Reshape XYZdown into a 1-dimensional (vector) array
% because level-1 s-functions will only accept 1-dimensional inputs
XYZ1col=XYZdown( : );

% Because the input and output sizes must match, zeros
% must be added to end input matrix to make it as long as
% the needed output matrix. Current input matrix length is
% npts*3*nslice. Output matrix must be of length
% npts*4*(nslice-1). Thus to make the input length equal
% the output length, add a single column zero matrix that
% is of length: ( npts*4*(nslice-1) - npts*3*nslice ) =
% npts*(nslice-4)
XYZ1col=[XYZ1col; zeros(npts*(nslice-4),1)];

A.1.3 M-file Function: readdatafile.m

Function readcontourpts.m uses subroutine readdatafile.m to read individual data
files. The geometry points are then available in MATLAB’s workspace for use by
Simulink.

function [xyzacross, npts] = readdatafile(inputfile)
%readdatafile.m
% This function reads in the x, y, z coordinate points
% from a single data file. The data points can then be
% connected with a spline.
%
% This function is used within readcontourpts.m
%
% Input is: inputfile, the character string of the file
% name
%
% Outputs are: xyzacross, an npts-by-3 matrix of coordinate
% points with each point on a separate row with x, y, and
% z coordinates in the first, second and third columns
% respectively; and npts, the number of points in the data
% file
clear fid
fid=fopen(inputfile,'r'); % Inputfile is passed as input to the function

i=1; % i is counter for rows of data
A=fscanf(fid,'%f, %f, %f
',3); % Scans first line of file

clear xyzacross
while length(A)>0 % Read until end of file
    % Places x's in first column, y's in second column, 
    % z's in third.
    xyzacross(i,1)=A(1);
    xyzacross(i,2)=A(2);
    xyzacross(i,3)=A(3);
    i=i+1; % Increment counter
    A=fscanf(fid,'%f, %f, %f
',3); % Read next line
end

% Record the number of data points for later use
npts=i-1;

close(fid); % Close data file
A.1.4 M-file Function: *creategeom.m*

M-file function *creategeom.m* is used during both the pre-simulation set-up and during the simulation. The function creates a geometry object that can be imported into COMSOL and is used during the finite element analysis. The function also plots the geometry object in MATLAB.

```matlab
function [g1]=creategeom(XYZ1col, npts, nslice)
%creategeom.m
% This function takes a 1D matrix of coordinate data from
% a series of sections and reshapes it. Then it connects
% the points of each section with a spline. Finally, the
% stack of splines are lofted to create a solid geometry
% that can be imported into COMSOL.
%
% This function is used within: msfun_bone.m
%
% Inputs are: XYZ1col, a (3*npts*nslice)-by-1 array where
% all data point's x, y, and z coordinates are listed in a
% single column (i.e. [x1;y1;z1;x2;y2;z2;..]); npts, the
% number of points in each data file; and nslice, the
% number of data files in the series
%
% Output is: g1, a geometry object capable of being
% imported into COMSOL
%
% Adapted by R. Jansen (2008) from import_tube2.m written
% by Xia Liu (2006).
%
% Reshape 1-dimensional XYZ1col back into a 3D matrix that
% is 3-by-npts-by-nslice
XYZdown=reshape(XYZ1col, 3, npts, nslice);
%
% Define vertices match vector
clear t11;
t11 = cell(1,nslice);
%
% clear c;
c=cell(1, nslice);
```
for i=1:nslice

    % Take all x and y coordinates from a single slice and
    % place in 2-by-npts sized matrix
    xy=XYZdown(1:2,:,i);

    % Connect points in matrix xy with spline and use
    % solid2 to convert from 2D curve to 2D solid
    c{i}=geomspline(xy, 'splinemethod', 'uniform', ...
        'closed', 'on');
    c{i}=solid2(c{i});
    geom=c{i};

    % Define the matching vertices. The 1st and 2nd points
    % (as in the data file, not the spline curve) of each
    % slice are used to define the vertex matching pair,
    % i.e. the 1st point on the 1st curve will be matched
    % with that on the 2nd curve, and the 2nd point on the
    % 1st curve will match with 2nd point on the 2nd
    % curve, etc. A while loop is designed to find the
    % point numbers.
    j=0;
    ipt1=0;
    ipt2=0;
    pt1=XYZdown(npts*3*(i-1)+1:npts*3*(i-1)+2);
    pt2=XYZdown(npts*3*(i-1)+4:npts*3*(i-1)+5);

    % Make sure that points are in the right order
    while ipt1*ipt2==0 && (j<=npts-1)
        j=j+1;
        [xx]=geominfo(c{i}, 'out', {'xx'}, 'par', {{j}} );
        xx{1}();
        magnitude1=sqrt(sum(((xx{1}()-pt1(:)).^2),2));
        magnitude2=sqrt(sum(((xx{1}()-pt2(:)).^2),2));
        if magnitude1<0.00001
            ipt1=j;
        elseif magnitude2<0.00001
            ipt2=j;
        end
    end

t1{i}=[ipt1;ipt2];

    % All z-values in a single file should be the same
    z(i)=XYZdown(3,1,i);
% Loop on each slice/section/data file
81
% Define loft parameter
82 v=z(2:nslice)-z(1:nslice-1);
83
% Use loft to create solid geometry
84 g1=loft(c, 'loftvtxpair',tl1,'LoftSecPos', {v, [], []});
85
% Plot the geometry
86 geomplot(g1)

A.1.5 M-file Function: createcentfile.m

M-file function createcentfile.m creates a text file containing the coordinates of each element’s centroid and the averaged density for that element. The file is used to create an initial text file during set-up and is rewritten during each time step of the simulation to store the density that will be used in the following time step by Dinterpolation.m (see Section A.2.3).

function createcentfile(fem)
%createcentfile.m
% This function creates a text file containing the coordinates for the centroid of each element in the mesh, and the value for density at the centroid.
% The text file is later used by Dinterpolation.m.
% This function is called during msfun_bone.m.
% Input is: fem, the finite element structure.
% Written by R. Jansen, 2008.

% get current data from fem file
output=posteval(fem,'rho_sld','refine',1);

% find number of elements
nelements=length(output.p)/4;

% out1.p contains x,y,z data for each node
nodes=reshape(output.p,3,4,nelements);
% out.d contains E value at each node
datapt=reshape(output.d,1,4,nelements);

% for each element, find the centroid and the centroid's
density by averaging
for j=1:nelements
elemcntr(j,:)=mean(nodes(:,:,j),2)';
rho(j,1)=mean(datapt(:,:,j),2)';
end

centroid=[elemcntr rho]; %compile centroid information

% print centroid information to file
fid=fopen('centroid.txt', 'wt');
for i=1:4
    for j=1:nelements
        fprintf(fid, '%f ', centroid(j,i));
    end
    fprintf(fid, '\n');
end
fclose(fid);

% Uncomment this section to create a plot of mesh and
centroids. WARNING: Mesh plot takes a LONG time and is
not recommended to be used during simulations.
hold on
for j=1:nelements
    % Plots the mesh
    plot3([nodes(1,1,j),nodes(1,2,j)],[nodes(2,1,j),nodes(2,2,j)],[nodes(3,1,j),nodes(3,2,j)]);
    plot3([nodes(1,3,j),nodes(1,2,j)],[nodes(2,3,j),nodes(2,2,j)],[nodes(3,3,j),nodes(3,2,j)]);
    plot3([nodes(1,4,j),nodes(1,2,j)],[nodes(2,4,j),nodes(2,2,j)],[nodes(3,4,j),nodes(3,2,j)]);
    plot3([nodes(1,1,j),nodes(1,3,j)],[nodes(2,1,j),nodes(2,3,j)],[nodes(3,1,j),nodes(3,3,j)]);
    plot3([nodes(1,1,j),nodes(1,4,j)],[nodes(2,1,j),nodes(2,4,j)],[nodes(3,1,j),nodes(3,4,j)]);
    plot3([nodes(1,4,j),nodes(1,3,j)],[nodes(2,4,j),nodes(2,3,j)],[nodes(3,4,j),nodes(3,3,j)]);
end

% Plots the centroids
scatter3(elemcntr(j,1),elemcntr(j,2),elemcntr(j,3))
end
axis square
A.2 Simulation Routines

The simulation is performed using Simulink. The first computation block in the Simulink program BoneFeedback.mdl is s-function msfun_bone.m. The s-function, its numerous custom subfunctions, and other MATLAB function using during the simulation are discussed in this section.

A.2.1 Level 2 M-file S-function: msfun_bone.m

A level-2 m-file s-function is a user defined Simulink block. The purpose of this s-function, msfun_bone.m, is to construct the finite element problem, solve it, and produce the results needed for output to the rest of the adaptation model. The block’s input is a column vector of the control points created by readcontourpts.m. For a section-by-section description of the function, see Sections 2.4.1 and 2.4.2. This s-function uses creategeom.m, Dinterpolation.m, and intercepts.m.

function msfun_bone(block)

% msfun_bone.m
% This is a level-2 M-file s-function used in
% BoneFeedback.mdl.
% This function is based off of the level-2 M-file
% s-function template msfuntmpl.m which can be found in
% the folder matlabroot/toolbox/simulink/blocks.
% Written by R. Jansen (2008).

% The setup method is used to setup the basic attributes
% of the S-function such as ports, parameters, etc. The
% setup function should only be altered if changing the
% form of the input or output.
% If the actions of the s-function need to be altered, changes should be made to Output function.

setup(block);
endfunction

function setup(block)

% Register number of ports
block.NumInputPorts = 1;
block.NumOutputPorts = 1;

% Set port size
block.InputPort(1).Dimensions = -1;
block.OutputPort(1).Dimensions = -1;

% Setup port properties to be dynamic
block.SetPreCompInpPortInfoToDynamic;
block.SetPreCompOutPortInfoToDynamic;

% Override input port properties from default
block.InputPort(1).DirectFeedthrough = true;

% Register number of parameters
block.NumDialogPrms = 3;

% Register sample times
block.SampleTimes = [1 0];

% Register methods
block.RegBlockMethod('PostPropagationSetup', @DoPostPropSetup);
block.RegBlockMethod('InitializeConditions', @InitConditions);
block.RegBlockMethod('Outputs', @Output);
block.RegBlockMethod('Update', @Update);
endfunction

function DoPostPropSetup(block)
% The function creates the Dwork vector that is used to monitor time (t).
% Setup Dwork
block.NumDworks = 1;
block.Dwork(1).Name = 't0';
block.Dwork(1).Dimensions = 1;
block.Dwork(1).DatatypeID = 0;
block.Dwork(1).Complexity = 'Real';
block.Dwork(1).UsedAsDiscState = true;

function InitConditions(block)
% Sets the initial time to the value of the third
% parameter.

% Initialize Dwork
block.Dwork(1).Data = block.DialogPrm(3).Data;

endfunction

function Output(block)
% This function creates geometry using function
% creategeom.m which needs inputs of XYZ, npts, and
% nslice. Both XYZ and npts are outputs from the function
% readcontourpts.m.
%
% The geometry is then used in an FEA simulation. Most of
% the code from the simulation is taken from the COMSOL
% Multiphysics Model M-file, generated by COMSOL 3.4
% (COMSOL 3.4.0.248, $Date: 2007/10/10 16:07:51 $)
%
% This function uses user created functions: creategeom.m,
% Dinterpolation.m and intercepts.m.
% Written by R. Jansen (2008).

% Assign variable names for the input parameters, time
% (Dwork vector), and the block input.
t = block.Dwork(1).Data;
npts = block.DialogPrm(1).Data;
nslice = block.DialogPrm(2).Data;
u = block.InputPort(1).Data;
global XYZrecord

% Reduce the size of the input matrix back to the appropriate size by removing all of the zeros that were added by readcontourpts.m (see that function for details as to why zeros were added).

newu=u(1:npts*3*nslice);

% Record the current x, y, and coordinate points being inputted into the column t of XYZrecord. XYZrecord will be a (3*npts*nslice)-by-t matrix
XYZrecord(:,t+1)=newu;

% Reshape 1-dimensional XYZdata back into a 3D matrix that is 3-by-npts-by-nslice
XYZdown=reshape(newu, 3, npts, nslice);

% --- INSTRUCTIONS ---
% To generate this section of code, creategeom.m can be run independently to create the geometry g1. g1 can then be imported with the GUI interface of COMSOL (File>Import>Geometry). Use the subdomain and boundary physics windows to assign material properties and boundary and loading conditions. After solving, save the model as an m-file. Copy the applicable parts into this section.

% --- CREATE FEM ---
%********************************************************
flclear fem
% COMSOL version
clear vrsn
vrsn.name = 'COMSOL 3.4';
vrsn.ext = '';
vrsn.major = 0;
vrsn.build = 248;
vrsn.rcs = '$Name: $';
vrsn.date = '$Date: 2007/10/10 16:07:51 $';
fem.version = vrsn;

% --- EDIT! ---
%********************************************************
% Delete the following lines:
% flbinaryfile='----.mphm';
%
% % Geometry
% g1=flbinary('g1','draw',flbinaryfile)

% Replace with the following:
% Geometry
clear g1

% g1=creategeom(newu, npts, nslice);
%******************************************************

% Analyzed geometry
clear s
s.objs={g1};
s.name={'g1'};
s.tags={'g1'};

fem.draw=struct('s',s);
fem.geom=geomcsg(fem);

% Initialize mesh
fem.mesh=meshinit(fem,
                  'hmaxfact',0.8,
                  'hcurve',0.8,
                  'hgrad',1.45,
                  'hcutoff',0.05,
                  'hnarrow',0.6);

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlSolid3';
appl.gporder = 4;
appl.cporder = 2;
appl.assignsuffix = '_sld';

%--- EDIT! ---
%******************************************************

% Add the following lines before the section for bnd to
% find surfaces for BC and LC:
% Find bottom and top faces to apply BC and LC
foundall=0;
iface=0;
ifacetop=[];
%ifacebtm=[];
error=0.0001; % adjust according to scale of geometry

% Define the loading surface and supporting surface.

% Surfaces to find must have constant z-coord.
zttop=.01; %EDIT!
%zbtm=0;

% Obtain number of faces in geometry
% nbs=number of boundary segments (faces)
nfaces=geominf(g1,'out',{'nbs'});

while (foundall==0) && (iface<nfaces)
    iface=iface+1;
    % Find center of face
    xyz_mid=flgeomfd(g1,iface,[0.5;0.5]);
    if abs(xyz_mid(3)-ztop)<error
        % Search for top face
        ifacetop=[ifacetop iface];
    elseif abs(xyz_mid(3)-zbtm)<error
        % Search for bottom face
        ifacebtm=[ifacebtm iface];
    end
    if length(ifacetop)==1 %&& length(ifacebtm)==1
        % Check to make sure you have all the faces
        % For rectangular block, only 1 face
        foundall=1;
    end
end

%**********************************************
clear bnd
bnd.Hy = {0,1};
bnd.Hx = {0,1};
bnd.Hz = {0,1};

%--- EDIT! ---
%******************************************************************************
% Delete:
% bnd.ind=[1,1,... etc];
% Replace with construction of bnd.ind in terms of bottom
% and top faces:

% Set BC for top and bottom.  2=fixed, 1=free
bnd.ind=ones(1,nfaces);
bnd.ind(ifacetop)=2;
bnd.ind(ifacebtm)=1;

%******************************************************************************
appl.bnd = bnd;
clear equ
equ.rho = 'Dinterpolation(x,y,z)';
equ.E = 'dentomod(rho_sld)';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Functions
clear fcns
fcns{1}.type='piecewise';
fcs{1}.name='dentomod(rho_sld)';
fcs{1}.extmethod='none';
fcs{1}.subtype='general';
fcs{1}.expr={'.06+.9*(rho_sld/1000)^2)*10^9', ...
'(10^9.59*(rho_sld/1000)^2.39)*10^9'};
fcs{1}.intervals={'0','1000','2000'};
fem.functions = fcns;

% ODE Settings
clear ode
clear units;
units.basesystem = 'SI';
ode.units = units;
fem.ode=ode;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
'solcomp',{"w","u","v"}, ...
'outcomp',{"w","u","v"}, ...
'lin solver','spooles');

%******************************************************************************
% --- MECHANICAL SIGNAL DATA EXTRACTION ---
%=============================================================
% The following section of code extracts the mechanical
% signal from the model solution so that the results can
% be used to control adaptation or growth in the blocks
% following the s-function in the Simulink model
% BoneFeedback.mdl

% Calculate mechanical signal
for i=1:nslice-1 %the last slice doesn't remodel
    [ex ey ez exy eyz ezx]=postinterp(fem,'ex_sld',...
                        'ey_sld','ez_sld','exy_sld','eyz_sld',...
                        'exz_sld',XYZdown(:,i,:));
    mechsignal(:,:,i)=...
    [ex(:) ey(:) ez(:) exy(:) eyz(:) ezx(:)];
end

% Calculate the local coordinate system, including
% unit_normal vectors
[localsys, unitnormal]=intercepts(XYZdown, npts, nslice);

% Compute the local axial strain using the
% local coord sys found in intercepts
for i=1:nslice-1 %the last slice doesn't remodel
    for j=1:npts
        Atrans=reshape(localsys(j,:,i), 3,3);
        B(1,1)=mechsignal(j,1,i);
        B(2,2)=mechsignal(j,2,i);
        B(3,3)=mechsignal(j,3,i);
        B(1,2)=mechsignal(j,4,i);
        B(2,3)=mechsignal(j,5,i);
        B(1,3)=mechsignal(j,6,i);
        B(2,1)=B(1,2);
        B(3,2)=B(2,3);
        B(3,1)=B(1,3);
        localtensor=Atrans'*B*Atrans;
        %the local axial signal
        localaxial(j,i)=localtensor(2,2);
    end
end

% Filter the mechanical data
window=3;
npasses=1;
for i=1:nslice-1
    width=(window-1)/2;
sinpt=localaxial(:,i);
    for j=1:npasses
        datafront=sinpt(npts-width+1:npts);
dataend=sinpt(1:width);
dataexpanded=[ datafront; sinpt; dataend ];
datafiltered=filter...          (1/window*ones(1,window), 1, dataexpanded);
localaxialsingle=datafiltered...                         (window:length(datafiltered));
sinpt=localaxialsingle;
end
localaxial(:,i)=localaxialsingle;
end

%***********************************************************
% Use posteval to output density values at nodes
[output1, output2, output3, output4]=posteval(...
   (fem,'rho_sld','ex_sld','ey_sld','ez_sld','refine',1);
% Reshape data set into elements
nelements=length(output1.p)/4; %find number of elements
% output1.p contains x,y,z data for each node
nodes=reshape(output1.p,3,4,nelements);
% output1.d contains rho value at each node
datap1=reshape(output1.d,1,4,nelements);
% output2.d contains x normal strain value at each node
datap2=reshape(output2.d,1,4,nelements);
% output3.d contains y normal strain value at each node
datap3=reshape(output3.d,1,4,nelements);
% output4.d contains z normal strain value at each node
datap4=reshape(output4.d,1,4,nelements);
% For each element, find the centroid and the centroid's
for j=1:nelements
    elemcntr(j,:)=mean(nodes(:,j),2)';
rho(j,1)=mean(datapt1(:,:,j),2)'
traceE(j,1)=mean(datapt2(:,:,j),2)'+ ...
    mean(datapt3(:,:,j),2)'+ mean(datapt4(:,:,j),2)'

% Mechanically derived density remodeling rule
% constants
pm=1830; %material density for corticol bone kg/m3
f1=4.37; f2=-80; %constants
traceEo=.00045; %trace of RE strain
drho(j,1)=(pm*f1+f2*rho(j,1))*...
    (abs(traceE(j,1))-traceEo);
%formula from Cowin (1992)

rho(j,1)=rho(j,1)-drho(j,1);

if rho(j,1)<=0
    rho(j,1)=100;
end

% Non Mechanically driven densit remodeling rule
rho(j,1)=rho(j,1)+24;
%24/wk based on Cheng data, see plot

% Compile centroid information
centroid=[elemcntr rho];

% Store centroid information in file with time in name
centfilename=['centroid',num2str(t+1),'.txt'];
fid=fopen(centfilename, 'wt');
for i=1:4
    for j=1:nelements
        fprintf(fid, '%f ', centroid(j,i));
    end
    fprintf(fid, '\n');
end
fclose(fid); % close file

% Print centroid information to file for analysis in next step
centfilename='centroid.txt';
fid=fopen(centfilename, 'wt');
for i=1:4
    for j=1:nelements
        fprintf(fid, '%f ', centroid(j,i));
    end
    fprintf(fid, '\n');
end
end
fclose(fid); % close file
%*********************************************************
% print current time to screen
disp(['block at t= ', num2str(t)]);
% Save output of fem to file with current time in name
femdata=['Outfem',num2str(t)]; % file name string
fisave(femdata, fem); % save fem file as .mph file
% Output data back to Simulink
block.OutputPort(1).Data = [localaxial(:);unitnormal(:)];
%end Outputs

function Update(block)
% This function increments time (in this case, by 1) after
% each run of the Output function.
block.Dwork(1).Data = block.Dwork(1).Data+1;
%endfunction

A.2.2 M-file Function: creategoem.m

Function creategoem.m is called again during msfun_bone.m; see Section A.1.4 for code and description.

A.2.3 M-file Function: Dinterpolation.m

The function Dinterpolation.m opens and reads the text file, centroid.txt. The text file contains the x-, y-, and z-coordinates of the previous time step’s element centroids. The file also contains the adapted densities at the centroids. Dinterpolation.m uses the centroids’ coordinates to interpolate the adapted density onto the new element mesh.
function newD=Dinterpolation(x,y,z)
% This function opens centroid.txt to retrieve centroid data, calculates the distance from each centroid to the input point, and then finds the smallest distance to locate the centroid closest to the input point. The function outputs the data value associated with the closest centroid.

% This function used in msfun_bone.m.
% Inputs are: x, y, and z - row vectors containing the respective coordinates for the integration points in the finite element problem.
% Output is: newD, a matrix of the interpolated density for the integration points based on a function of the density from the previous step in centroid.txt
% Written by R. Jansen (2008).

fid=fopen('centroid.txt', 'r');
% centroid.txt is centroid data created by createcentfile.m (in prep) or by msfun_bone.m.
% It contains x on a row, y on a row, z on a row, then selected data on the 4th row

[A count]=fscanf(fid, '%f ', inf);
fclose(fid); % close file

% Reshape so that data is arranged as it was in the file
centroid=reshape(A, count/4, 4);

xi=centroid(1,:); % known x points
yi=centroid(2,:); % known y points
zi=centroid(3,:); % known z points
ndata=length(x); % how many points are in row vector input

clear distance newD

for i=1:ndata
% for input point, calculate the distance to the known points (centroids).
distance(1,:)=((xi-x(1,i)).^2+(yi-y(1,i)).^2+...
\[ (z_i - z(1,i))^2 \cdot (1/2); \]

% find the smallest distance
[\text{mindist, index}] = \text{min(distance)};
% set the value of density for that input point to the
% density value of the closest centroid
newD(1,i) = centroid(4, index);
end

A.2.4 M-file Function: intercepts.m

Function intercepts.m finds the unit normal vectors, the unit tangent vectors, and the axial vectors for the control points. These vectors are used to control which direction the control points move during shape adaptation. This function uses subfunctions findcontrolpts.m, findtangents.m, drawclosedspline.m, contoursegintercept.m, and perpdist3d.m.

1 function [localsys, unitnormal] = intercepts(XYZdown, ...  
2 npts, nslice)  
3 %intercepts.m  
4 % This routine demonstrates how to find the normal  
5 % intercept for a point on one bspline to a bspline above  
6 % it in 3D space. For the primary spline, the tangent  
7 % vector at a given point (or set of points) is found.  
8 % Then a search technique is used to find the vector that  
9 % extends from that point (the start of the tangent  
10 % vector) and intersects a different bpline above the  
11 % primary (or below) at a particular segment number and  
12 % local u coordinate such that this vector is  
13 % perpendicular to the tangent. That is, the dot product  
14 % of this new vector with the tangent vector is zero. Next  
15 % we can cross these vectors and calculate a new vector.  
16 % These 3 vectors can then be used to define a local  
17 % coordinate system at the original point (tangential,  
18 % axial, normal)  
19 %  
20 % This function is used within: msfun_bone.m  
21 %  
22 % This function uses user created functions:
% findcontrolpts.m, findtangents.m, drawclosedspline.m, 
% contoursegintercept.m, perpdist3d.m

% Inputs are: XYZdown, the 3-by-npts-nslice matrix of all
% the control point coordinates; npts, the number of
% points; and nslice, the number of slices.
%
% Outputs are: localsys, a npts-by-9-by-nslice matrix
% containing a combination of the unit tangent and axial
% vectors and their cross product; and unitnorm, an
% npts-by-3-by-(nslice-1) matrix containing the unit
% normal vectors for the control points.
%
% Adapted by R. Jansen (2008) from intercept3_grow.m
% written by Xia Liu (2005).

colordef none;

localsys=[];

for i=1:nslice-1

    % Read the coordinates of the i-th slice and the slice
    % above it
    [xyzacross1]=XYZdown(:,:,i)';
    [xyzacross2]=XYZdown(:,:,i+1)';

    % Calculate control points defining these splines
    [p1]=findcontrolpts(xyzacross1, npts);
    [p2]=findcontrolpts(xyzacross2, npts);

    % Calculate tangents at local_u=0 for each segment of
    % the first (primary) curve
    ulocals=zeros(npts,1);

    tangents=findtangents(p1, ulocals, npts);

    drawclosedspline(p1, 1, 'g-')
    drawclosedspline(p2, 1, 'r-')

    % First find which segment the normal intercepts in
    % secondary contour
    [segintercept]=contoursegintercept(xyzacross1, ...
                                        xyzacross2, tangents, npts);

    % Then use bisection technique to find local u value
    % of this perp intersector
errorstop=0.00001;

% Want dot product to be at least this close to zero
[perpdist, intersectxyz]=perpdist3d(segintercept, ...
p2, errorstop, xyzacross1, tangents, npts);

% Draw the result
for j=1:npts
    % location on primary contour
    vectorstart=xyzacross1(j,:);
    vectorend=intersectxyz(j,:);
    perpvector=[vectorstart; vectorend];
    hold on;
    plot3(perpvector(:,1), perpvector(:,2), ...
    perpvector(:,3), 'y-');
    hold off;
end

% Calculate axial components
magnitude=sqrt(sum(((intersectxyz-xzacyross1).^2),2));
for j=1:npts
    axial(j,1)=(1.0./magnitude(j))*...
        (intersectxyz(j,1)-xyzacross1(j,1));
    axial(j,2)=(1.0./magnitude(j))*...
        (intersectxyz(j,2)-xyzacross1(j,2));
    axial(j,3)=(1.0./magnitude(j))*...
        (intersectxyz(j,3)-xyzacross1(j,3));
end

% Calculate unit tangents
magnitude=sqrt(sum(((tangents).^2),2));
for j=1:npts
    unittangents(j,1)=(1.0./magnitude(j))*...
        (tangents(j,1));
    unittangents(j,2)=(1.0./magnitude(j))*...
        (tangents(j,2));
    unittangents(j,3)=(1.0./magnitude(j))*...
        (tangents(j,3));
end

% cross(unit_tangents,axial)
for j=1:npts
    unitcross(j,:)=cross(unittangents(j,:), ...
    axial(j,:));
end
% compile unit tangent, axial and unit cross data
localsys(:,:,i)=[unittangents axial unitcross];
%calculate in-plane unit_normal, cross product of $\text{unit}_\text{tangents}$ and z unit vector
for j=1:npts
    unitnormalsingle(j,:)=cross(unittangents(j,:), [0 0 1]);
end
unitnormal(:,:,i)=unitnormalsingle;
end

A.2.4.1 M-file Subfunction: findcontrolpts.m

Function `findcontrolpts.m` is used to find the coordinates of the control points using the splines that are created in `creategeom.m`. Because the geometry used in this project was created using control points, this function is somewhat redundant. However, it is useful in instances where the geometry is originally defined using splines around a sliced image.

function [p] = findcontrolpts(xyzacross, npts)
%findcontrolpts.m
% This function finds the control points using the splines
% that outline the geometry on each slice.
% This function is used within: intercepts.m
% This function is used within: intercepts.m
% Inputs are: xyzacross, an npts-by-3 matrix of coordinate
% points from a single slice with each point on a separate
% row and with x, y, and z coordinates in the first,
% second and third columns respectively; and npts, the
% number of points in the data file
% Output is: p, an npts-by-3 matrix of coordinates points
% of the control points used to scribe a cubic spline
% through the points in xyzacross
%
% Adapted by R. Jansen (2008) from controlpts_via_xyz.m
% written by Xia Liu (2005).

% For a cubic spline, there are 4 terms: u^3, u^2, u, and 1. If u=0, then U4=[0 0 0 1]. And since M4=(1/6)*
% -3 1; 3 -6 3 0; -3 0 3 0; 1 4 1 0], the product U4M4 is
% given as below:

U4M4=(1/6)*[1.0 4.0 1.0 0.0];

for i=1:npts
    % Scatter the result into larger matrix first
    % determine subscripts from mod operation
    n=npts-1;
    if (i-1)<0
        sub1=(n+1)+(i-1);  \%(i-1)mod(n+1)
    else
        sub1=(i-1);
    end

    if i==(n+1)
        sub2= 0;  \%(i)mod(n+1)
    else
        sub2=i;
    end

    if (i+1)>=(n+1)
        sub3=(i+1)-(n+1); \%(i+1)mod(n+1)
    else
        sub3=(i+1);
    end

    if (i+2)>=(n+1)
        sub4=(i+2)-(n+1); \%(i+1)mod(n+1)
    else
        sub4=(i+2);
    end

    % Now scatter the results (note that we add one to
    % the subscripts since matlab has no zero subscripts

    UnMn(i,sub1+1)=U4M4(1,1);
    UnMn(i,sub2+1)=U4M4(1,2);
    UnMn(i,sub3+1)=U4M4(1,3);
    UnMn(i,sub4+1)=U4M4(1,4);
end
A.2.4.2 M-file Subfunction: findtangents.m

The function findtangents.m determines the unit tangents of the control points found in findcontrolpts.m.

```matlab
function [tangents] = findtangents(p,ulocals,npts)
%findtangents.m
% This function is used within: intercepts.m
% Inputs are: p, an npts-by-3 matrix of coordinates points
% of the control points used to scribe a cubic spline
% through the points in xyzacross; ulocals, an npts-by-1
% matrix of zeros; and npts, the number of control points
% per slice
% Output is: tangents, an npts-by-3 matrix of the tangent vectors for the control points for a single slice
% This program was adapted from tangents_via_controlpts.m
% written by Xia Liu (2005).

% We do not assume that all the evaluations occur at local
% u=0. Instead we pass down a list of segment numbers and
% the local u value for that segment. Of course, we could
% pass down local u=0 for all segment numbers if we wanted.
% Note that the length of the vectors segnums and ulocals
% should be equal.
% cubic bspline derivative matrix
M4PRIME=(1/6)*[-3 9 -9 3; 6 -12 6 0; -3 0 3 0];

for i=1:npts
    U3=[(ulocals(i))^2 ulocals(i) 1.0 ];
    U3M4PRIME=U3*M4PRIME;
    % get subscripts
```

% Solve linear system
p=UnMn\xyzacross;
\begin{verbatim}
33 n=npts-1;
34 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
35 if (i-1)<0
36    sub1=(n+1)+(i-1);  % (i-1) mod (n+1)
37 else
38    sub1=(i-1);
39 end
40
41 if i==(n+1)
42    sub2= 0;           % (i) mod (n+1)
43 else
44    sub2=i;
45 end
46
47 if (i+1)>=(n+1)
48    sub3=(i+1)-(n+1);  % (i+1) mod (n+1)
49 else
50    sub3=(i+1);
51 end
52
53 if (i+2)>=(n+1)
54    sub4=(i+2)-(n+1);  % (i+1) mod (n+1)
55 else
56    sub4=(i+2);
57 end
58 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
59 % gather appropriate control points for curve segment
60 % need to add 1 to the subscripts since MATLAB doesn't
61 % have zero subscripts
62 p4=[p(sub1+1,:);p(sub2+1,:);p(sub3+1,:);p(sub4+1,:)];
63 tangents(i,:)=(U3M4PRIME)*p4;
64 end
\end{verbatim}

A.2.4.3 M-file Subfuncion: drawclosedspline.m

The function drawclosedspline.m draws and plots a closed spline through the
control points in \( p \). It is called twice during \textit{intercepts.m} - first to create a spline for the
current slice and then again for the slice above it.
function drawclosedspline(p, nsamplespersegment, colorarg)
%drawclosedspline.m
%
% This function is used within: intercepts.m
%
% Inputs are: p, an npts-by-3 matrix of coordinates points
% of the control points used to scribe a cubic spline
% through the points in xyzacross; nsamplespersegment,
% number of splines to use to connect control points; and
% colorarg, the color with which to plot the splines
%
% This program adapted from drawclosedspline3.m written by
% Xia Liu (2005).

bsplinedecords=[];
M4=(1/6)*[-1 3 -3 1; 3 -6 3 0; -3 0 3 0; 1 4 1 0];

npts=length(p);
oorder=4; % cubic spline (3+1=4)

for i = 1:npts
    for j = 0:nsamplespersegment-1
        localu = j/nsamplespersegment;

        % evaluate u^3, u^2, u, 1 and put in U
        for k=1:oorder;
            U4(1,k)=localu^(oorder-k);
        end

        n=npts-1;
        if (i-1)<0
            sub1=(n+1)+(i-1); % (i-1)mod(n+1)
        else
            sub1=(i-1);
        end

        if i==(n+1)
            sub2= 0; %(i)mod(n+1)
        else
            sub2=i;
        end

        if (i+1)>(n+1)
A.2.4.4 M-file Subfunction: contoursegintercept.m

The function contoursegintercept.m uses the tangents found in findtangents.m and the closed splines created by drawclosedspline.m to find the closest point on the spline
above for each point in the current spline. The function’s output is `segintercept`, a matrix containing the vectors to connect the two points.

```matlab
function [segintercept]=contoursegintercept(xyzacross1,...
    xyzacross2, tangents, npts)
%contoursegintercept.m
% This function finds the segment in the secondary spline
% defined from p2 (xyz coords, u_local=0) where the normal
% from primary points on p1 will cross. This info can
% then be used to conduct root search in perpdist3d.m.
% This function is used within: intercepts.m.
% Inputs are: xyzacross1, the coordinates of the current
% slice; xyzacross2, the coordinates of the next slice;
% tangents, a matrix contains the tangent vectors for the
% control points of the current slice; and npts, the
% number of control points in a slice.
% Written by Xia Liu (2005).
% how many intercepts we're looking for, keeps
% segintercept as npts-by-1 matrix
segintercept=zeros(npts,1);
for i=1:npts;
    for j=1:npts;
        vectors(j,:)=xyzacross2(j,:)-xyzacross1(i,:);
        magnitude(j) = sqrt(vectors(j,1)^2 + . . .
            vectors(j,2)^2 + vectors(j,3)^2);
        dotproducts(j)=dot(tangents(i,:), vectors(j,:));
    end
    % look for the index where the sign of the dot product
    % changes (should be 2 places)
```
```matlab
nextdotproducts=[dotproducts(2:npts) dotproducts(1)];
test=dotproducts.*nextdotproducts;
k=find(test<=0);

% of these segments, select the one with the smaller
% vector length
kk=find( magnitude(k) == (min(magnitude(k))) );
segintercept(i)=k(kk);
end
```

### A.2.4.5 M-file Subfunction: `perpdist3d.m`

The function `perpdist3d.m` uses `segintercept` from `contoursegintercept.m` and the tangents from `findtangents.m` to find the vector that is perpendicular to the tangent of the point on the current slice and intersects the closest point of the slice above as found in `contoursegintercept.m`. This vector is the axial vector, offset by a distance of `intersectxyz`. This function uses subfunction `samplespline.m`.

```matlab
function [perpdist, intersectxyz] = perpdist3d...
(segintercept,p2,errorstop,xyzacross1,tangents,npts)
%perpdist3d.m
% This routine uses a bisection root finding technique.
% The function for which a root is being sought is the dot
% product between the tangent on the primary contour
% (at a particular xyz location) and the vector extending
% from that tangent location to a position on the
% secondary contour. When the dot product==0, the two
% vectors are perpendicular (hence, we locate the
% perpindicular intersection on the second contour).
% This function is used within: intercepts.m.
% This function uses user created function:
% samplespline.m.
% Inputs are: segintercept, matrix of intercepts from
% contoursegintercept.m; p2, the control points for the
% slice after xyzacross1; errorstop, maximum error
```
for i=1:npts;
    ulower=0.0;
    uupper=1.0;
    err=100.0;

    % location on primary contour
    vectorstart=xyzacross1(i,:);
    tangentstart=tangents(i,:);
    j=0;

    while err>errorstop;
        xyzlower = samplebspline(p2, ulower, ...
            segintercept(i), npts);
            %vector for 'lower' root search
            vect_lower = xyzlower - vectorstart;

        xyzupper = samplebspline(p2, uupper, ...
            segintercept(i), npts);
            %vector for 'upper' root search
            vect_upper = xyzupper - vectorstart;

        uroot=(uupper+ulower)/2.0; %bisection method
        xyzroot = samplebspline(p2, uroot, ...
            segintercept(i), npts);
            %vector for 'bisector' root search
            vect_root = xyzroot - vectorstart;

        % calculate dot products search vectors and
        % tangents. (The factor of 100 spreads out the
        % result.)
        dot_lower = dot(vect_lower,tangentstart)*100;
        dot_upper = dot(vect_upper,tangentstart)*100;
        dot_root = dot(vect_root, tangentstart)*100;
err = abs(dot_root);

test = dot_lower * dot_root;

if abs(dot_lower) < 1e-8
    uroot = 0;
    xyzroot = samplebspline(p2, uroot, ...
                             segintercept(i));
    vect_root = xyzroot - vectorstart;
    err = 0.0;
elseif test == 0
    err = 0.0;
elseif test < 0
    uupper = uroot;
else
    ulower = uroot;
end
j = j + 1;
if (j > 100)
    [err j]
    break
end
perpdist(i) = norm(vect_root, 2); % vector magnitude
intersectxyz(i,:) = xyzroot;

A.2.4.6 M-file Subroutine: samplebspline.m

The subroutine samplebspline.m find the next set of x- and y-coordinates along
the slice’s spline to be passed to perpdist3d.m.

function [splinecoords] = samplebspline(p, u, i, npts)
%samplebspline.m
% This subroutine finds the next set of x-y coordinates
% along the slice’s spine for use in perpdist3d.m.
% Inputs are: p, pre-determined control points defining a
% CLOSED bspline; u, order number; i, segment number; and
% npts, the number of control points per slice.
%
% Output is: splinecoor, a single pair of coordinates
% along with the tangent (derivative) at this point.
% This function is used within: perpdist3d.m
% Written by Xia Liu (2005).

n=length(p)-1;
order=4; % cubic bspline
M4=(1/6)*[-1 3 -3 1; 3 -6 3 0; -3 0 3 0; 1 4 1 0];

%evaluate u^3, u^2, u, 1 and put in U
for j=1:order;
    U4(1,j)=u^(order-j);
end

% Now pick out the required 4 control points
if (i-1)<0
    sub1=(n+1)+(i-1);  % (i-1)mod(n+1)
else
    sub1=(i-1);
end

if i==(n+1)
    sub2= 0;  % (i)mod(n+1)
else
    sub2=i;
end

if (i+1)>(n+1)
    sub3=(i+1)-(n+1);  % (i+1)mod(n+1)
else
    sub3=(i+1);
end

if (i+2)>(n+1)
    sub4=(i+2)-(n+1);  % (i+2)mod(n+1)
else
    sub4=(i+2);
end

%need to add 1 to the subscripts since MATLAB doesn't have
%zero subscripts
\begin{verbatim}
53  \textbf{p4}=[\textbf{p}(\text{sub1}+1,:) ; \textbf{p}(\text{sub2}+1,:) ; \textbf{p}(\text{sub3}+1,:) ; \textbf{p}(\text{sub4}+1,:)];
54  \textbf{splinecoords}=(\textbf{U4} \cdot \textbf{M4}) \cdot \textbf{p4};
\end{verbatim}

A.2.5 **Simulink Blocks: get localaxial & get unitnormal**

The output of the s-function `msfun_bone.m` is column vector \textit{u}, which contains both the mechanical signal data for shape adaptation (in this case, local axial strains) and the unit normal vectors for the control points. MATLAB function blocks \textit{get localaxial} and \textit{get unitnormals} are used to separate \textit{u} accordingly as shown in Table A.2.

<table>
<thead>
<tr>
<th>Block</th>
<th>MATLAB function</th>
</tr>
</thead>
<tbody>
<tr>
<td>get localaxial</td>
<td>\textit{u}(1:\text{npts}(\text{nslice}-1))</td>
</tr>
<tr>
<td>get unitnormals</td>
<td>\textit{u}(\text{npts}(\text{nslice}-1)+1:\text{npts}\times4(\text{nslice}-1))</td>
</tr>
</tbody>
</table>

Table A.2: Definition of MATLAB functions \textit{get localaxial} and \textit{get unitnormals}.

A.2.6 **Simulink Subroutine: Shape Adapt**

The Simulink subroutine \textit{Shape Adapt} (see Figure 2.15) has inputs of both the local axial strain and the unit normals found during `msfun_bone.m`. The local axial strain signal is put through a series of blocks to convert the strain magnitudes into a bone formation rate using a mechanostat function. The formation rate and the unit normals are recombined into a single column vector using a matrix concatenation block and are passed to MATLAB m-file function \textit{calcspeed.m}.

145
A.2.6.1 Simulink Blocks for Mechanostat Function

Five Simulink blocks comprise the mechanostat function (see Figure 2.15 and Figure 2.17). The first block is an absolute value block that takes the absolute value of the strain signal. The next block is a dead zone function that sets the specified range of strain inputs to be equal to zero. The dead zone function also offsets the block’s input by the start or end value in order to eliminate discontinuities. In this implementation, the range is set from .0001 to .0002 to create a dead zone of 100 µε to 200 µε surrounding the remodeling equilibrium strain of 150 µε.

The signal then passes through MATLAB function block, slopes.m (see Section A.2.6.2). The function scales the filtered strain input according to the slope of the mechanostat function.

The following matrix gain block turns the mechanically driven shape adaptation on and off. The adaptation is carried out when the gain is set to 1; when the gain is set to 0, the adaptation is not included in the simulation.

The output of the MATLAB function block is passed to a saturation block that limits the signal to a specified range. For this application, the range was set with an upper limit of 0.00012 and a lower limit of -0.00012 to establish the apposition or resorption rate to a maximum of 0.00012 m/wk.

A.2.6.2 M-file Function: slopes.m

The m-file function slopes.m allows the slopes of the mechanostat function to be different by scaling inputs greater than zero at one rate and inputs less than zero at
another. In this implementation, inputs less than zero are multiplied by 1.2, and inputs greater than zero are multiplied by 0.045.

```matlab
function rate = slopes(u)
    %slopes.m
    % This function creates the sloping portion of the
    % mechanostat function used during shape adaptation in
    % BoneFeedback.mdl
    %
    % Input is u, the column vector containing the axial
    % strain signal after it has passed through the dead zone.
    %
    % Output is rate, the column vector containing the rate of
    % bone formation.
    %
    % Determine the length of the input
    le=length(u);
    % Setup the output variable
    out=zeros(le,1);
    for i=1:le % for each element in u
        % if the input is less than zero, apply a gain of 12
        if u(i)<0
            out(i,1)=u(i)*1.2;
        % if the input is greater than zero, use a gain of .42
        else
            out(i,1)=u(i)*0.45;
        end
    end
    rate=out;
```

**A.2.6.3 M-file Function: `calcspeed.m`**

The function `calcspeed.m` multiplies the mechanical signal for shape adaptation and the unit normals at each control point to determine how the control points should be along the unit normals for the shape adaptation. The function outputs `speedvector1col`, a
column vector containing the distance each coordinate should be adjusted to adapt the control points accordingly.

```matlab
function [speedvector1col] = calcspeed(u,npts,nslice)
% calcspeed.m
% This function calculates the speed vector of how fast
% the geometry is changing using the mechanical signal and
% the unit normal.
% This function is used within BoneFeedback.mdl.
% Input is: u, a (3*npts*(nslice-1))-by-1 matrix where
% entries 1:npts*(nslice-1) are mechanical signal data and
% entries npts*(nslice-1)+1:npts*4*(nslice-1) are unit
% normal data; npts, the number of points; and nslice, the
% number of slices.
% Output is: speedvector1col, a (npts*(nslice-1)*3)-by-1
% matrix.
% Adapted by R. Jansen (2008) from Cal_speed.m written by
% Xia Liu (2007).

% separate u into the mech signal and unit normal data

% size npts*(nslice-1)-by-1
mechsignal=u(1:npts*(nslice-1));

% size 3*npts*(nslice-1)-by-1
unitnormal=u(npts*(nslice-1)+1:npts*4*(nslice-1));

% reshape the unit normal data matrix so that it is size
% npts-by-3-(nslice-1)
unitnormal=reshape(unitnormal, npts, 3, (nslice-1));

% reshape unit normal data matrix into 2D matrix with
% slices in order. tempunitmornal will be size
% npts*(nslice-1)-by-3, where rows correspond to points
% and columns are the x, y, and z coordinates of the unit
% normal for that point.
for i=1:nslice-1
    tempunitnormal(npts*(i-1)+1:npts*i,:)=unitnormal(:,i,:);
end
unitnormal=tempunitnormal;
```
% calculate speed vector. size of speedvector should be 
% npts*(nslice-1)-by-3 
for i=1:npts*(nslice-1) 
    speedvector(i,:)=mechsignal(i)*unitnormal(i,:); 
end 

% reshape speedvector into a (npts*(nslice-1)*3)-by-1 
% matrix 
speedvector1col=speedvector(:); 

A.2.7 Simulink Subroutine: Grow

The Simulink subroutine Grow (see Figure 2.18) inputs the unit normals and 
scales them using a matrix gain block. The matrix gain K corresponds to the distance 
along the normal that the control point will be moved; thus a positive K produces a 
growing geometry and a negative K produces a shrinking geometry.

A.2.8 M-file Function: deltaXYZ.m

The function deltaXYZ.m outputs the matrix containing the amount the x-, y- and 
z-coordinates in XYZ1col should be adjusted to control the shape adaptation. The 
function reshapes the input matrix so that the x-, y-, and z-coordinates are arranged in the 
same fashion as XYZ1col.

function [dXYZ]=deltaXYZ(u,npts,nslice) 
%deltaXYZ.m 
% This function reorganizes the input to line up with the 
% coordinate matrix of XYZ1col. The output dXYZ is added 
% to XYZ1col to determine the geometry of the next time 
% step. 
% 
% This function is used in msfun_bone.m.
Input is: \( u \), the matrix output from calcspeed.m; \( npts \),
the number of control points per slice; and \( nslice \), the
number of slices.

Output is: \( dXYZ \), the matrix of how the original
coordinates should change for the next iteration.

Adapted by R. Jansen (2008) from deltaXYZ.m written by
Xia Liu (2007).

\[
\text{deltaacross}=\text{reshape}(u, npts, 3, nslice-1); \\
\text{deltadown(:,:,j)}=\text{deltaacross(:,:,j)');}
\]

\[
deltadown(:,:,nslice)=0; \text{ last slice does not remodel} \\
dXYZ=deltadown(:,); \text{ output into 1 column} \\
dXYZ=[dXYZ;\text{zeros(npts*(nslice-4),1)}]; \text{ pad with zeros to match input size}
\]


