An Interactive Design Framework Based on Data-Intensive Simulations: Implementation and Application to Device-Tissue Interaction Design Problems

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Chi-Lun Lin

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Arthur G. Erdman

Feb, 2015

Acknowledgements

First and foremost I would like to thank my advisor Professor Arthur Erdman. I have been amazingly fortunate to have him as my advisor who gave me the freedom to explore on my own, and at the same time guided me with patience when my steps faltered. He always made himself available to advise me in research, academic writing and career development. I am also grateful for the opportunities he offered me in participating various interdisciplinary academic/industrial research projects, from which I grew my professional knowledge and network.

My appreciation also goes to faculty members that have guided me through all these years, including Professor Frank Kelso and my dissertation committee members Professor William Durfee, Professor Daniel Keefe and Professor Rajesh Rajamani. Specifically, Professor Keefe has been a critical advisor throughout my five years here. He has introduced interdisciplinary insights to my research.

I would also like to thank my collaborators and research team members that supported and contributed to this research. This includes Dane Coffey, Daniel Orban, Bethany Tourek, John Huss, Cory Schaffhausen, Nancy Rowe and David Porter. I would like to thank my best friend in the mechanical engineering department Masao Shimada for his thoughtful discussions and all kinds of support through these years.

Finally, I would like to thank three important people in my life. My wife Hui-Chen Hsu, my father Wei-Hson Lin and my mother Hsiu-Chu Chuang. I would not be able to finish this research without their unconditionally support and encouragement.

The funding for my PhD study in these years has come from many sources, including Boston Scientific Corp., Medtronic, Inc. and a main project grant from the National Science Foundation under grant no. IIS-1251069 and the National Institutes of Health under grant no. 1R01EB018205-01.

Dedication

This dissertation is lovingly dedicated to my wife, Hui-Chen Hsu. Her support, encouragement, and constant love have sustained me throughout my life.

Abstract

This dissertation investigates a new medical device design approach based on extensive simulations. A simulation-based design framework is developed to create a design workflow that integrates engineering software tools with an interactive user interface, called Design by Dragging (DBD) [1], to generate a large-scale design space and enable creative design exploration. Several design problems illustrate this design workflow are investigated via featured forward and inverse design manipulation strategies provided by DBD. A device-tissue interaction problem as part of a vacuum-assisted breast biopsy (VAB) cutting process is particularly highlighted. A tissue-cutting model is created for this problem to simulate the device-tissue contact, excessive tissue deformation and progressive tissue damage during the cutting process. This model is then applied to the design framework to generate extensive simulations that samples a large design space for interactive design exploration. This example represents an important milestone toward simulation-based engineering for medical device prototyping.

The simulation-based design framework is implemented to integrate a computer aided design (CAD) software tool, a finite element analysis (FEA) software tool (SolidWorks and Abaqus are selected in this dissertation) and a high performance computing (HPC) cluster into a semi-automatic design workflow via customized communication interfaces. The design framework automates the process from generating and simulating design configurations to outputting the simulation results. The HPC cluster enables multiple simulation job executions and parallel computation to reduce the computation cost. The design framework is first tested using a simple bending needle example, which generates 460 simulations to sample a design space in DBD. The functionality of the creative inverse and forward design manipulation strategies are demonstrated.

A tissue cutting model of a VAB device is developed as an advanced benchmark example for the design framework. The model simulates the breast lesion tissue being positioned in a needle cannula chamber and being cut by a hollow cutting tube with simultaneous rotation and translation. Different cutting conditions including cutting speeds and tissue properties are investigated. This VAB device design problem is then applied to the design framework. Critical design variables and performance attributes

across three main components of the VAB device (the needle system, motor system and device handpiece) are identified. 900 design configurations are generated and simulated to sparsely populate a large-design space of 10^6 possible solutions. The design space is explored via the creative design manipulation strategies and several uses cases are established.

The bending needle example demonstrates the first success of the proposed simulation-based design framework. The 460 simulations are completed with minimal manual interventions. The functionality of DBD is also demonstrated. The inverse and forward design strategies allow interacting with the design space via dragging on a radar chart widget or directly on the visualization of the simulation. Through the interactions the user is guided to the desired solutions.

The VAB tissue-cutting example provides a realistic medical device application of the design framework. The 900 simulations are completed in parallel in the HPC cluster so that the computation time is significantly reduced. The simulation output data is converted to a high-efficiency data format called NetCDF so that the post-compuation for sampling this large design space is made possible. Several use cases are demonstrated. By interacting with the radar chart widget, the user gradually gains the understanding and new insights about the effect of design variable modifications. Next, the direct manipulation strategies via visualization of the simulations are used to solve three issues, including a 'dry tap', moving a leading edge of the tissue sample and narrowing a stress concentration area. These use cases successfully demonstrated the capability and the usability of the design framework.

There are two major contributions of this dissertation. The first is the investigation of the new design approach that enables creative design exploration based on extensive simulation data. This success moves a step toward the simulation-based medical device engineering with big data. The second is the FEA simulation model for the VAB tissue cutting process. This model utilizes realistic breast tissue properties to predict cutting forces during the VAB sampling process, which has not been found in the literature. The studies conducted using this model extend the current understanding of the VAB tissue cutting process under different cutting conditions. All of these achievements illustrate the potential for a future medical device virtual prototyping environment.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Understanding device-tissue interactions is crucial when developing implantable devices and surgical instruments. These interactions interaction are in general too complex to be solved analytically so that they have to be approached numerically or experimentally. In many cases, replicating the scenarios of these kinds of interactions in bench tests is impossible, which results in an early initiation of animal trials with a limited understanding of effects of design parameter selections. As a consequence, more design-test iterations may be required, which is costing and time consuming.

Computer simulation appears to be a solution to overcome this limitation. As the computational techniques mature and high-performance computing (HPC) becomes more powerful, more and more complicated engineering problems (such as complex contact conditions, large nonlinear deformation and fluid-structural interaction) can be simulated efficiently. Many device/tissue interface models can be simulated and evaluated before animal trials begin. Therefore, the engineers gain more useful design insights and produce more confident products before entering animal and human trials.

Evaluating a design usually breaks down to its multiple system components and how these component influence each other. For instance, designing a biopsy tool that samples soft tissue involves evaluation of its needle, cutter and driver. One has to understand how the tissue is being cut under different needle geometry, cutting tip shapes and cutting speeds. One also has to know how the cutter sustains the tissue reaction forces, and thus evaluate the driver selections. Combining all of these system components together, device-level factors (such as the tool ergonomics and manufacturing cost) must

also be considered. As a result, the design space becomes large and design decisions can be very difficult to make.

Current design optimization packages are able to create a process flow that controls engineering software packages (such as computer-aided design (CAD) and finite element analysis (FEA) packages) to simulate design concepts. The users are able to identify optimal design parameters from the resulting design space. The design tasks focus on maximizing/minimizing the parameter selections subject to required constraints. This usually applies to a later design stage with a good understanding of the design problem. In an earlier design stage, it is more important to be able to comprehensively explore the design space and gain knowledge and design insight. An integrated design environment that provides not only the information about the selected design parameters but also all other aspects of design and analysis data is needed. In particular, the 3D visualization of the simulation models should be included to provide information that cannot be easily/possibly presented by numbers. In addition, the traditional design tables, 2D plots and 3D contours are insufficient when the design space becomes large and multi-dimensional. To deal with such big data, we need new visualization methods to quickly understand the effects and trade-offs of design parameters [2]. Finally, exploring design solutions in such a multi-dimensional, large design space requires features beyond what currently available optimization algorithms can offer. New 3D visualization technologies can provide more view angles for the design space, but human interaction with big data becomes essential to discovering new findings and design insights.

In this dissertation, a simulation-based framework for medical device design is proposed. The framework integrates CAD, FEA and HPC system to efficiently generate large amount simulation data. The framework uses an open-source high-efficient data form to communicate with a new human-computer interface named Design by Dragging (DBD) [1], which allows directly interacting and manipulating the data visualization. In order to demonstrate a realistic medical device design process, a sophisticated cutting model of the vacuum-assisted biopsy (VAB) procedure is developed and applied to the framework. To present, the achievements on the simulation-based design framework has been reported [3, 4, 5]. This dissertation provides details of the implementation of the framework, the development of the VAB cutting model, and how the model is introduced to the framework to generate the use cases. These use cases describe how

real simulation-based engineering workflows are improved when new interactive visualizations are utilized to better understand a large design space.

There are two starting points for this research. One is to implement a simulation-based design framework to enable large-scale interactive design exploration, while the other is to develop a realistic VAB cutting model as a benchmark example for the implemented framework. In the following chapter, the background for these two research directions is reviewed. Chapter 3 and 4 describe the implementation of the design framework and the development of a FEA model that simulates the VAB cutting process, respectively. In Chapter 5, the model is introduced to the design framework. A large-scale design exploration using the interactive design strategies is performed and use cases are presented. Finally, the conclusion of this research and avenues for future work are provided in Chapter 6.

Chapter 2

Literature Review

This chapter provides literature review into two research directions. In section 2.1, related works of interactive and integrated engineering design framework are reviewed. The review investigates the existing methods of integrating engineering software tools and performing interactive design tasks. This helps decide a best approach to implement the proposed design framework. Section 2.2 reviews the background of the proposed design problem: the tissue cutting of the VAB. The needle biopsy technology for breast tumor diagnosis and its clinical challenges are briefly reviewed. The science of the VAB cutting method and related studies are investigated. A comprehensive survey of state-of-the-art breast tissue models are also conducted.

2.1 Integrated Engineering Design Environment

2.1.1 VR-CAD Integration

CAD systems have been widely used for design drafting and manufacturing simulation. In a CAD user environment, the designer interacts with a classical WIMP (windows, icons, menus, pointer) interface to complete design tasks. Attempts have been made to integrate existing CAD systems into virtual reality (VR) or other intuitive user interfaces to enhance the user experience and potentially help produce more creative design results. Whyte et al. [6] reviewed three different approaches in construction to the implementation the VR-CAD integration, which are based on library of standard

parts [7], direct model conversion [8] and a central database [9], respectively. This review concluded a few important points for the future implementation of the VR-CAD integration: (1) the direct model conversion is inefficient as a converted model is usually a geometric description and the associate information relating to establishing the model is lost; (2) the library-based approach requires efforts for the development of part libraries at the beginning, which can be useful when complicated user activities or extensive model attributes are involved; and (3) a central database that controls model parameters for both of CAD and VR is particular suitable for rapid prototyping. The implementation would allow changes made in VR to be updated in CAD, but a neutral model format accessible to both VR and CAD has to be established.

Virtual assembling is a main application of the VR-CAD integration. The Virtual Assembly Design Environment (VADE) [10] is a well-known application which allows analyzing and evaluating assembly of mechanical system in a VR environment. Barbieri el al. [11] developed an interface to load constraint data from CAD (Unigraphics NX3) models into a VR environment for virtual assembly analyses. The data decomposition and information translation method (DDITM) [12] is able to export information including geometry, topology, assembly and tessellation from a CAD system (SolidWorks) into a virtual assembly system. Another VR-CAD integration approach for virtual assembly writes necessary information to an exchange file. These works can be categorized into either the direct model conversion or the database-based approach. The former converts model information into a file, while the latter creates a link that connects the CAD and the VR systems. However, making design changes in the VR system and updating the changes to the CAD system is not considered in this type of integration.

Editing CAD objects in the virtual environment is another important research direction. Bourdot et al. [13] developed a system to enable direct 3D editing of CAD (OpenCASCADE) objects within a VR environment. The main module of this system, named Virtual Reality Aided Design, included a parametric model and a mesh management component. The parametric model stored the Feature Dependency Graph (FDG) and Boundary Representation (B-Rep) of CAD objects and a mesh management component was responsible for generating the triangulated surface meshes for 3D rendering. To enable the 3D editing, an inference system called SWI Prolog is used to manage operations that generate topological elements. Ng et al. [14] established another VR-CAD

system. The team integrated the SolidWorks© system into an augmented VR environment. CAD models were visualized in the 3D environment while the user can evaluate designs via gestures. These works further enhanced the database-based approaches implemented in the virtual assembly applications, which directly communicate a CAD kernel to enable model manipulations. These works also emphasized the needs and usefulness of 3D visualization and interactions in product development.

2.1.2 Integrating FEA to a Design Framework

The design environments discussed in the previous section enabled viewing and editing designs with more intuitive user interfaces. However, they are not integrated with simulation tools, such as FEA, to add loading, stresses, etc. for design optimization. There were several research works proposing CAD-FEA integrated methods for product design and analysis [15, 16]. The main challenge was to simultaneously maintain both the CAD model and FEA model while each of them used a different geometry representation and contained different kinds of information. A key component proposed in these research works is to overcome this challenge by using a parametric data model repository working as a middleware to interface between the CAD and FEA models. Thus, the design and analysis workflow is automated and a centralized control over the design parameters is provided.

Integrating FEA to the design process is also emphasized in biomedical applications. The model repository is extended to include more information, e.g. anatomical geometry reconstructed from medical images, tissue database and clinical data. Goh et al. integrated FEA to a a commercial prosthetic CAD workstation. The team developed a technique to convert the CAD data to FEA code including user-specified loading and boundary conditions to predict prosthetic socket interface pressure [17]. A computer-based design framework for designing and testing virtual prostheses [18] is implemented based on a biomechanical model with a set of patient data. The framework integrated Abaqus software for running simulations and adopted the built-in auto-meshing technique with pre-selected mesh elements. Material properties, loading and boundary conditions are also predefined with adjustable types and values.

FEA is also integrated to a virtual platform to create simulations of patient's anatomy for surgical training and planning. The Virtual Physiological Human is a

suite of modeling, simulation and visualization tools, which are currently developed under the European Commission Virtual Physiological Human (VPH) program. In one of the projects, a dysfunctional left ventricle of a pathological heart is reconstructed. A custom FEA code is used to simulate surgical procedures, such as cutting, patching and stitching [19]. This platform has aimed to allow investigating whole human body [20].

2.1.3 Summary

There is a clear need for an integrated virtual design environment. The review in section 2.1.1 and 2.1.2 shows successful examples of integrating CAD and FEA software tools into a design process for modeling, simulation and evaluation. Using a parametric data repository as an interface to communicate between different systems and to provide a centralized control of design parameter has been a preferred approach, which enables efficient data exchange and little information loss. This dissertation describes how to apply this key concept into the implementation of the proposed simulation-based design framework to integrate engineering software tools and provide a new way to populate and represent a massive, multi-dimensional design space.

2.2 Breast Cancer and Needle Biopsy

2.2.1 Breast Cancer Diagnosis

Breast cancer has been one of the major diseases to cause deaths, especially in women. Breast cancer accounts for 25% of all cancers in women and caused more than 500 thousand deaths in 2012 [21]. However, the survival rates of an early diagnosed breast cancer are high and the 5-year survival rate of diagnosed breast cancer in the U.S. is about 90% [22]. Unfortunately, the rate decreases significantly when the cancer is diagnosed at a late stage [23]. Therefore, early and accurate diagnoses are crucial to improve the survival rates.

Ultrasound based percutaneous needle breast biopsy is a procedure to retrieve breast lesion tissue samples for disease assessment. It is a safe and accurate alternative to open surgical biopsy. The procedure removes tissue samples from the target lesion with minimal invasion to microscopically analyze them for breast cancer diagnosis. To

ensure an accurate histological assessment, the biopsy tool must be able to retrieve quality tissue samples. The sample quality largely depends on lesion localization and tissue sample volume [24].

2.2.2 Vacuum-Assisted Biopsy

Vacuum assisted biopsy (VAB) technology was designed to provide large-core breast tissue samples for more accurate diagnosis. Using a more sophisticated cutting method (i.e. rotary or scissor cutting) than traditional tools, VAB technology is capable of removing tougher tissue, such as tissue containing calcifications. Although VAB is reliable and widely used, the technology still has shortcomings and limitations.

Underestimation rates of the breast cancer diagnoses on VAB samples have been reported in previous studies [25, 26, 27, 28]. These studies were conducted retrospectively in medical institutions to investigate the ability of VAB devices to correctly diagnose breast cancer. Several brands of VAB tools with different needle gauges were used to remove tissue samples from different types of target lesions and comparisons were performed. The results suggested that VAB significantly improves the diagnosis accuracy when compared to smaller types of biopsy needle (e.g. fine needle aspiration and core needle biopsy). This is mainly because the VAB tools provide larger needle sizes, use an efficient cutting method and retrieve multiple samples with one insertion. However, in these studies, underestimation rates were reported to be 8.1% for an 8-gauge needle and 13.0% for an 11-gauge needle.

Studies also reported insignificant sample sizes and blank sample collections [29, 30]. In some cases, the biopsy cutting mechanism was not strong enough to traverse through dense tissue. In other instances, tiny tissue samples were obtained when sampling very soft tissue, such as adipose tissue. Occasionally a dry tap was reported, a technical term used to indicate that the tissue sample is not fully separated from the main tissue. As a result, tissue volume was not retrieved in a sampling sequence. Dry taps were also reported during our interviews with interventional radiologists. In the event of any of these situations, the patient must undergo a rebiopsy.

2.2.3 Soft Tissue Cutting in VAB

Although the performance of the VAB tools was studied clinically, the aspects of the device-tissue interface has not been fully investigated. In particular, the medical device designer is interested in understanding how the tissue is cut and its reaction force/torque so that the input power source(s) for a VAB tool can be appropriately selected and the overall tool ergonomics can be improved. The rotating cutting method of VAB is a type of slicing (i.e. cutting with forces in both directions tangential and normal to the cutting surface) as it drives a hollow cylindrical cutter to simultaneously rotate and translate simultaneously to cut the tissue. A few cutting force models of slicing soft materials have been published [31, 32, 33]. The results proved that slicing largely decreases the cutting force in the direction normal to the cutting surface, which reduces the compression experienced by the tissue prior to it being removed. These studies also defined an important cutting parameter, called the slice-push ratio, representing the ratio of the tangential component of the cutting speed to the normal component of the cutting speed. This parameter was identified as a main factor influencing the cutting forces. The slice-push ratio was further investigated analytically and experimentally [34]. The authors suggested that varying the slice-push ratio in a certain range dramatically changes the cutting forces. This finding provides a useful guideline for selecting the cutting speeds for VAB tools, but the results may not represent the realistic cutting conditions as the bench test was conducted on a linear elastic phantom tissue using a low needle insertion speed. In contrast, the VAB tools can be operated with higher cutting speeds and the cutting process can involve more complicated nonlinear tissue responses. These aspects have to be considered when developing a realistic VAB tissue cutting model.

2.2.4 Constitutive Breast Tissue Models

Computer simulation can help enhance and extend our current understanding of the VAB cutting. Constitutive models for three main types of breast tissue, adipose, fibroglandular and tumor, are found in the literature [35, 36, 37, 38, 39, 40]. Hyperelastic models for adipose and fibroglandular tissue are available in Neo Hookean and Polynomial forms, while an approximated Youngs modulus is identified for tumor tissue.

Although these models were originally developed to predict the tumor displacement due to lying prone during surgery or being compressed during mammography, they are currently the best achievable models for use in the simulation. Using these constitutive tissue models, more realistic tissue behavior when undergoing large and rapid deformation can be predicted.

In addition to the tissue deformation, tissue fracture must be described during the cutting process. However, no computational model or test data for breast tissue fracture exists in the literature, only limited information about ultimate tensile stress and strain of soft tissues was presented [41]. Tougher tissue, such as tendon and ligament, can withstand 10-15% strain before fracture, while softer tissue, such as blood vessels, can resist 50-100% strain. Although these statistics are not specifically related to breast tissue, they provided a general idea about the resistance of the soft tissue against the occurrence of the fracture. Specifying the maximum tissue resistance would be a currently available approach to describe tissue removal in the simulation and is used for developing our VAB cutting model.

2.2.5 Summary

In this research, we develop a VAB cutting model as a benchmark example to evaluate the proposed simulation-based design framework. In Chapter 4, we describe how we combine currently available information to develop a tissue-cutter interaction model of VAB cutting to predict tissue reactions. Constitutive models of tissue deformation are integrated with simplified tissue fracture criteria to establish our tissue models. In the simulation, the tissue is positioned in the chamber of a VAB coaxial needle and is cut with the rotary cutting method. Reactions for the three main types of breast tissue are computed under different cutting speeds. Findings, usefulness and future improvements of this model are discussed

Chapter 3

A Simulation-Based Design Framework

3.1 Design by Dragging (DBD)

DBD [1] is an user interface for exploring large design spaces. The design spaces are pre-populated sparsely with solutions of complex FEA simulations and the interface is able to interpolate between these solutions. DBD is featured with intuitive user interaction and data visualization to allow approaching design solutions in both forward (find resulting design outcome from design input parameter selections) and inverse (find potential design configurations for specified design outcome) directions. Dragging is the main designer activity in the DBD environment to explore the design space and manipulate design parameters. The internal algorithms interpret designer's intents from the drag operations corresponding to the parametric models. The interpretation can lead to different kind of responses in the visualization, such as modifying the geometry of the design and manipulating the output fields. Each drag operation returns new information about the design space and the designer is guided to step-by-step navigate to best solutions. This type of design exploration involves extensive user interaction with the data, but requires minimal routine tasks. The human decision is needed in every next exploring step so that expert's knowledge can inform the search process at any stage of the exploration.

3.1.1 Radar Chart Widget

In DBD, a design space is represented by a radar chart widget, which contains a pair of radar charts. One of the charts represents the design input parameter space, while the other represents the design outcome space (see Fig. 3.1). Each spoke of the charts corresponds to one of the parameters/performance attributes. The maximum value of a parameter/attribute is at the location where the spoke intersects the perimeter of the chart, while the minimum is at the center of the chart. Therefore, the length of a spoke represents the range of its corresponding parameter/attribute. A unique set of parameter selections (i.e. a design instance) is denoted by a polyline generated by connecting the selected data point on each spoke. The user's current location in the design space is indicated by a pair of solid polylines in the charts. Figure 3.1 shows the radar chart widget in a simplified form. A design space including 8 design input parameters and 6 output performance attributes is represented. The input radar chart is on the left-hand side, where the design parameters are denoted by P_i. The red polyline indicates the current selection of the input parameters. On the right-hand side is the output radar chart, where the output performance attributes are denoted by A_i. The red polyline shows the values of $A_i's$ corresponding to the selected input parameters.

In a design space, relative distances between each two design instances are computed using the following equation (see [42] for more details):

$$d(A,B) = \sum_{i=1}^{n} \sqrt{w_i * (D(A_i, B_i))^2} \tag{3.1}$$

where $D(A_i, B_i)$ is a function that calculates the normalized distance metric for the i-th input parameter or output attribute and w_i is the weight of the i-th parameter or attribute, which gives the important of the parameter/attribute. A normalized distance metric is also shown as the distance between the two data points on a spoke, if the total length of the spoke is 1. Therefore, the relative distance $d(A_i, B_i)$ calculated by this equation takes all the parameters/attributes and their importance into consideration.

A tab is attached to the remote end of its spoke for quick weight assignments. By left-clicking the tab, a weighting state can be switched between $p$inned, $w$eighted and $f$ree. When a spoke tab is pinned (the weight is infinite), the parameter/attribute is

fixed during any user operation. The weighted state (weight = 1) means a parameter/attribute is attached a highest importance. The free state (weight = 0) indicates that a parameter is not of concern during drag operations. By introducing the relative distance and the weighting into the radar charts, many kinds of design exploration strategies are enabled. For example, the designer can easily look at a particular group of design instances by pinning some design input parameters to certain desired values. As another example, weighted states can be assigned to multiple parameters/attributes to emphasizes the importance of those parameters. This informs the internal algorithm to search the neighborhood of current design (note that relative distances are redefined when each time a new weighting is assigned).

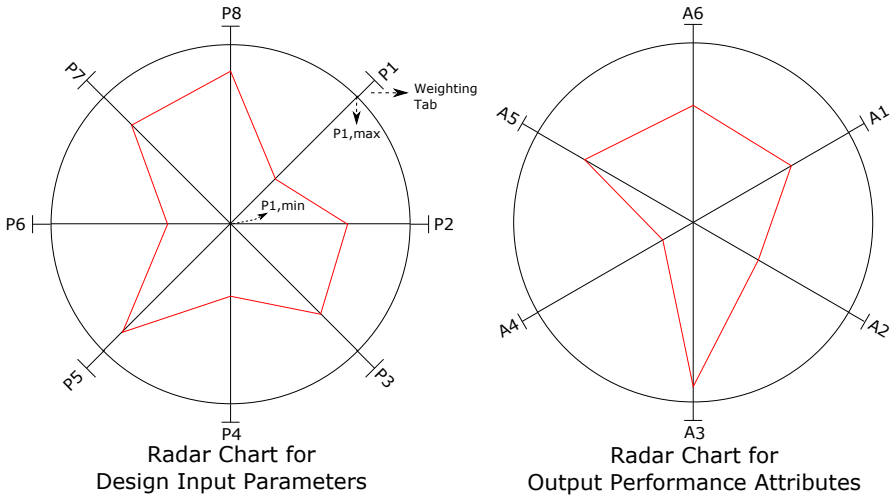


Figure 3.1: The radar chart widget- simplified version

3.1.2 Inverse and Forward Search Using Radar Chart Widget

A simple example of bending a steel cantilever beam is used to explain how to interact with the radar chart widget. As shown in Fig. 3.2, four design input parameters and four output performance attributes are identified and displayed in the radar charts widget. A four dimensional, continuous design space can be generated based on beam equations and user-defined parameter ranges. Next, the normalized distance metric of each spokes is determined and the designer can begin to interact with the widget.

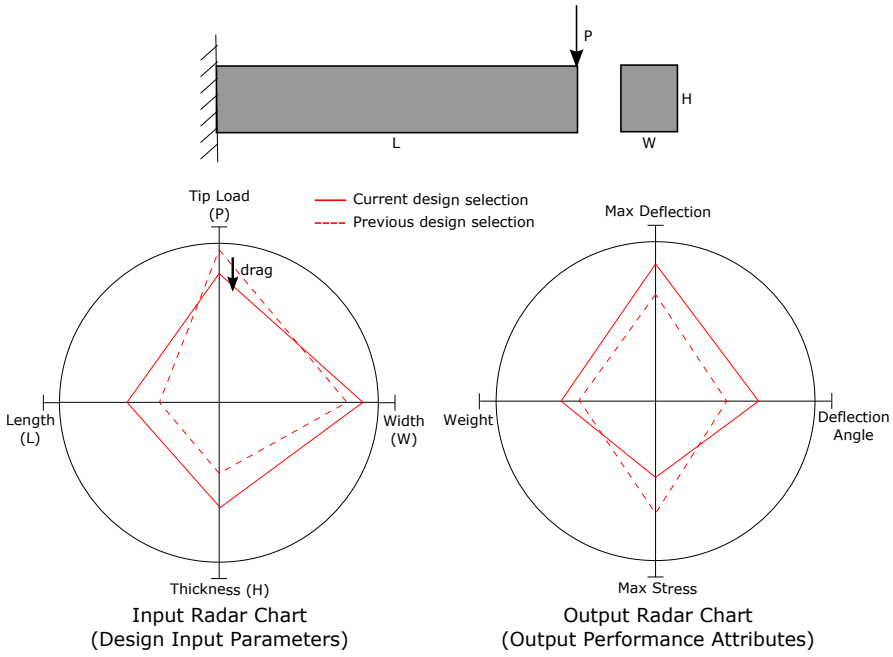


Figure 3.2: The radar chart widget for a bending beam example

The forward search is the most basic design exploration method. In a design space, each set of input values always maps to a unique set of output values. The forward search is based on this one-to-one mapping and the user interaction with the spokes of the input radar chart. In this example, we drag on the spokes of the input radar chart to modify all of the four parameters and switch to the current design instance. The output radar chart also immediately responses to the drag operations and finally reaches to the design instance shown as solid polyline. This type of interaction allows the user to quickly explore through design instances and learn the effect of the design input parameters on the design outcomes. For example, if the effect of the tip load on the output performance attributes is of interest, the designer can drag the current data point on the spoke of the tip load to increase/decrease its value and see how the output radar chart changes correspondingly. The same drag operation can be performed to investigate the effects of other design input parameters.

The inverse search allows searching for optimal input configurations by directly specifying design outcomes. This is made possible by the weighting assignments and the implementation of the relative distances. Here we demonstrate a process of finding

optimal design for a bending beam, as seen in Fig. 3.3. A beam weight m_t and a deflection angle θ_t are targeted as initial design requirements. After setting these two output performance attributes to the target values (step 1) and assign them weighted states (step 2), new relative distances between the design instances are defined. We realize that the maximum stress is too high, so we drag on the corresponding spoke to reduce its value (step 3). During the dragging, the internal algorithm searches the design space and finds the closest design instance that fits the search condition (i.e. the lower maximum stress). In other word, the beam weight and deflection angle of the returned design instance have similar/same values to/as m_t and θ_t. Now the design outcome is satisfied, but the beam is too thick. In that case, the most straightforward way is to directly control the spoke of the thickness in the input radar chart. In order to make sure the desired design outcome is kept, the maximum stress and two previously weighted attributes are all weighted (step 4). This ensures them to keep unchanged or only vary slightly in response to any further drag operations. Finally, we lower the thickness to acceptable values and find design alternatives. These alternatives are the optimal solutions based on the design requirements that have been input. This is a step-by-step design exploration guided by user interaction. In each step, the radar chart widget returns new information and the user is involved to input his/her judgment to move toward the next step.

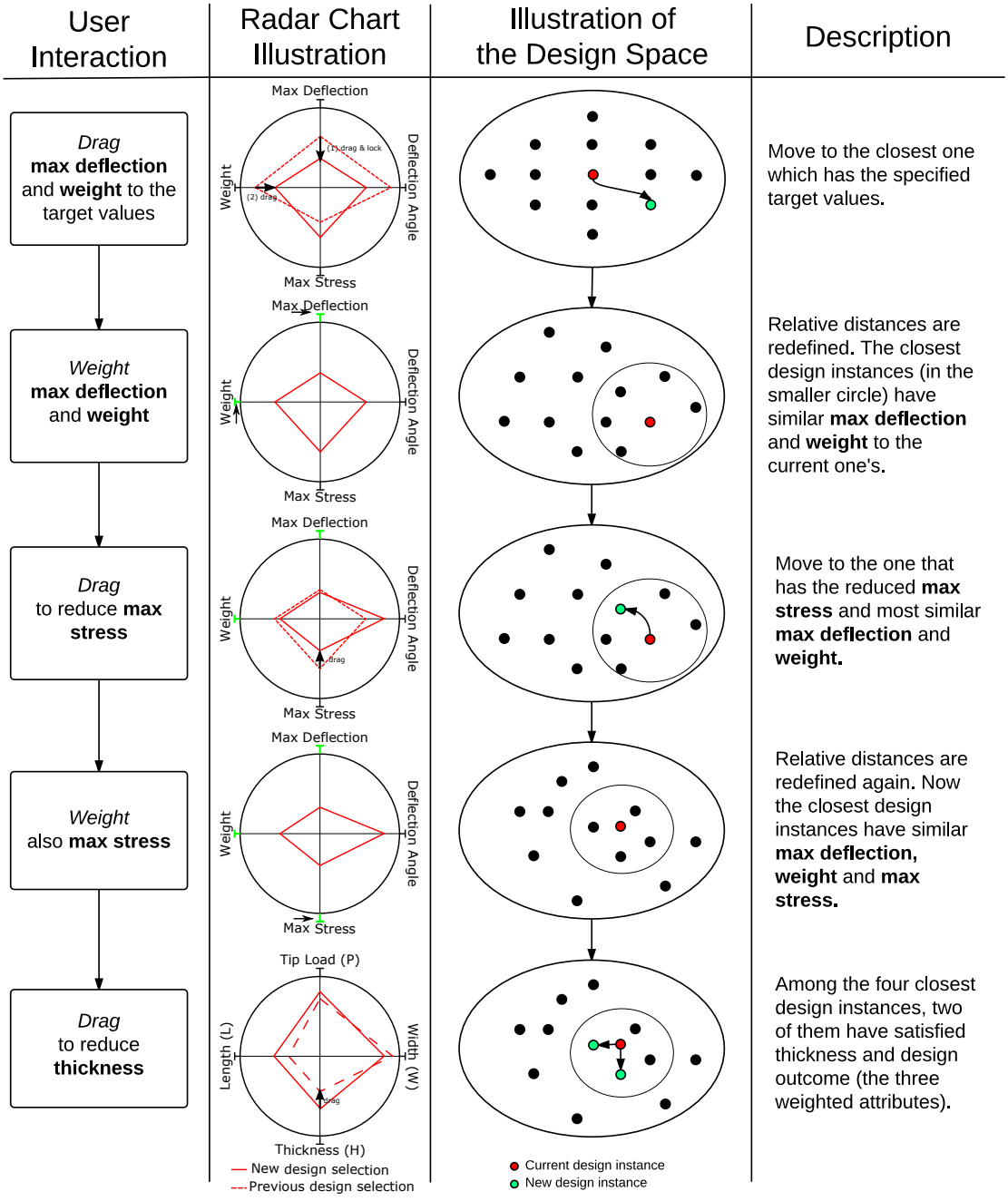


Figure 3.3: A process of finding optimal solutions for the bending beam.

3.1.3 Direct Manipulation via Visualization

In addition to interacting with the radar chart widget, the user can also perform drag operations directly on top of data visualizations. This type of interaction is called direct manipulation, which enables both forward design and inverse design strategies. In forward design, the user can drag on a geometric feature to make design changes. For example, ragging the outer wall of a VAB needle perpendicular its long axis to imply a change in the needle outer radius. The needle outer radius would be decreased when the drag direction is toward the the long axis, while increased when dragging in the opposite direction. The direct manipulation allows modifying many kinds of geometric parameters based on what geometric feature and in what direction the user drags. The internal algorithm interprets the user intent to decide which parameter to increase or decrease. On the other hand, the user can directly manipulate a spatially distributed output field, such as a stress field, to accomplish the inverse design. This inverse manipulation is enabled by the internal algorithm to continuously reshape the output field and display morphs between the pre-computed simulation results.

Figure 3.4 shows an example of manipulating the design space of a breast biopsy needle. In forward design, the user drags on an edge of the opening window on the cannula and move it towards the opposite edge to decrease the window length. In inverse design, the user manipulates the shear stress field resulted from a perpendicular load applied to the needle tip. The goal is to find design instances that have the current high stress region moved away from the corner of the opening window.

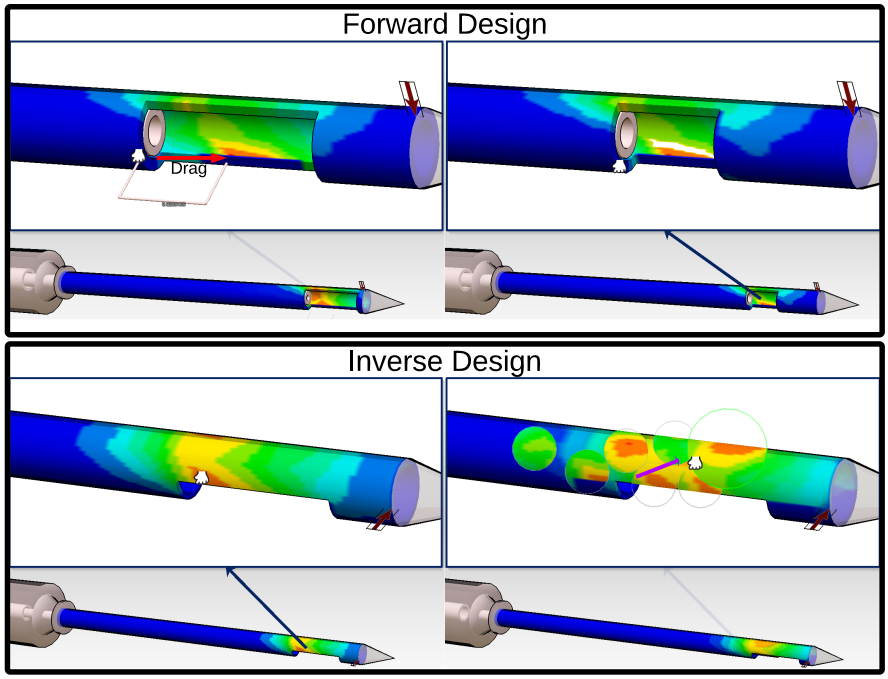


Figure 3.4: Examples of designing a biopsy needle using direct manipulation via data visualizations. In forward design, the user drags an edge of the opening window to the right (left). This operation is interpreted as decreasing the window length (right). In inverse design, the user attempts to move a high stress region (the cursor location) away from the corner of the opening window (left). The user right-clicks on the region, then the system suggests design instances that have the closest distances from the current one, shown as preview bubbles (expanded circle views). Each of the preview bubbles shows a magnified view of local stress distribution, which informs where the high stress region can possibly move to. The user finally moves into the most-right preview bubble to switch to a new design instance (upper-right). The new design instance shows the high stress region has moved away from the window corner (bottom-right).

3.2 A Simulation-Based Design Framework

The goal of the proposed simulation-based design framework is to integrate the design engineering tools with DBD to enable large-scale design exploration. The framework interfaces CAD and FEA systems to provide a semi-automatic workflow for generating

simulation data. The data is loaded into DBD so that it becomes a centralized user environment and data repository in the framework. Taking advantage of the framework, the user can efficiently populate a large design space and interact with it in DBD.

Figure 3.5 defines the scope of the proposed framework, which includes the following key components:

- DBD- Design by dragging, a user interface that enables interactive design strategies.
- CAD system- SolidWorks© 2013 is used in this research.
- FEA system- Abaqus© 6.13. is used in this research.
- High performance computing (HPC) system- its extensive amount of compute nodes allow FEA simulations run with parallel computing and provides a job queue system to accept multiple executions of the simulation jobs.

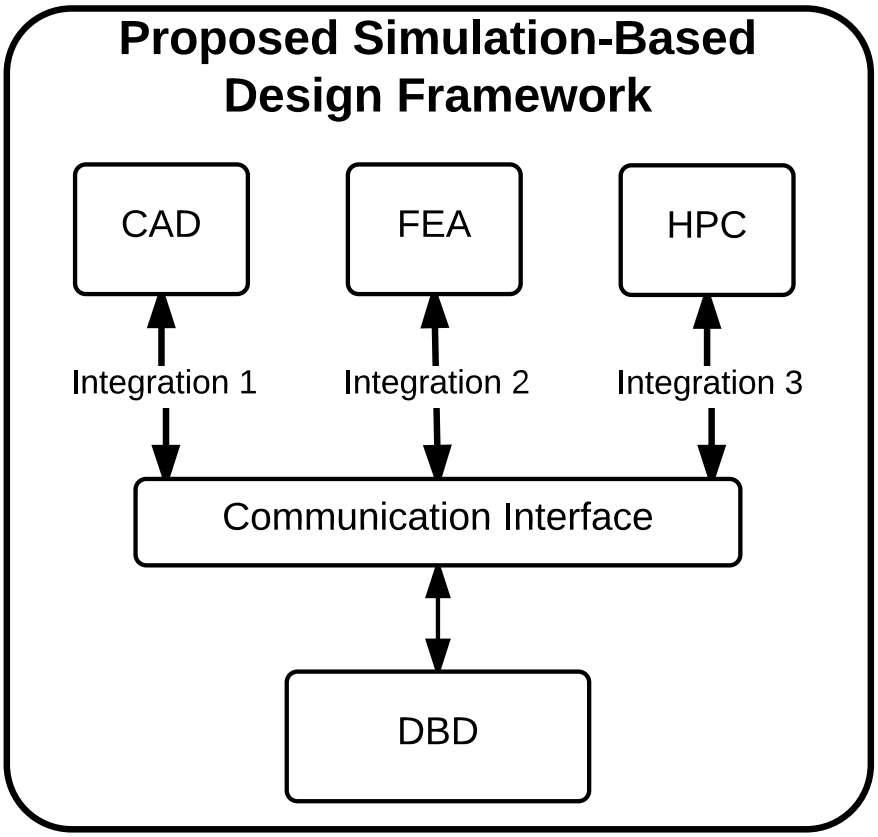


Figure 3.5: The proposed simulation-based design framework.

The implementation of the framework focuses on interfacing between the above-mentioned key components. The communication protocol and data exchange and storage format are developed. The arrows connecting the components indicate the data communication paths. These bi-directional communication paths are implemented by integration 1 to 3, which are listed in Table 3.1.

Table 3.1: Description of the system integrations

Implementation	**Downward Path**	**Upward Path**
Integration 1: CAD-DBD	Viewing CAD models	Making design parameter changes to CAD models
Integration 2: FEA-DBD	Converting FEA data to a readable format	Populating FEA simulation jobs via parameter sets generated in DBD
Integration 3: HPC-DBD	Efficiently loading large-scale FEA data to DBD	Submitting FEA simulation jobs to the HPC system for execution

3.3 Integrating System Components

3.3.1 Integration 1: Viewing Designs and Making Design Changes

This integration enables viewing designs and making design changes in DBD while the changes are simultaneously updated in the CAD system. It should be noted that this integration is not to duplicate CAD functionality or to invent new 3D modeling method. The goal is to interface with the CAD system so that (1) a CAD model can be displayed in DBD and (2) the parameter changes on the DBD model are updated to the CAD model during the user manipulation. A mechanism is developed to allow the bidirectional communication between DBD and the CAD system to read and control the design parameters. In the literature review, we have learned the following facts about the data exchange when integrating CAD into a virtual prototyping environment:

- There is no generic CAD format for a complete model transfer from one CAD system to another. Information loss is inevitable after a model conversion.
- Although every CAD system has fundamental elements in common, such as B-Rep and FDG, these common elements are formulated differently from system to system. Therefore, they are still system-specific. A data translation is still needed when reading geometry into another system.

- Existing VR-CAD integration methods have limitations for making design changes in an existing CAD model. This situation restricts the capability of design exploration.

Previous works have shown that model conversions between different types of system format results in information loss. Modifications made on the converted model are very difficult to translate back to the original system. In the proposed framework, instead of developing a conversion method without information loss, a dynamic link between the CAD and DBD is provided based on the API of the native CAD system. This allows real-time updating geometric parameters to the CAD system. The integration approach is summarized below:

- The data exchange between the CAD system and DBD is based on parametric CAD models. For every design change, only the values of those parameters are modified. Thus, the data loss can be avoided as no model conversion is required.
- The integration is based on the native CAD application programming interface (API). The native API allows the access to read and write pre-defined critical geometric parameters of the CAD model without changing its data structure.

The pre-condition of this integration is that the design concepts are converted into drawings where the important design features and dimensions are identified. Therefore, a parametric CAD model can be introduced as an realization of the design concepts. The parametric CAD model includes a base shape and parameters that define how the shape can vary. For example, a tube can be defined by a base cylinder shape and three parameters: length, outer radius and wall thickness. By setting a combination of these three parameters, a unique design can be generated. Various designs can be obtained by arbitrarily changing these parameters. As shown in Fig. 3.6, from CAD to DBD, the CAD interface reads the parametric model and submits the information to DBD. Based on the information received, a DBD model is established in the DBD environment. As the 3D shape and geometric parameters are identified, the user can start interacting with the model and making design changes in DBD. During the design manipulation, the parameters values of the DBD model keep being modified while the CAD model is being updated through the CAD interface. This made it possible to

populate designs by generating a set of geometric parameters in DBD. In addition, this type of synchronization keeps both models in their native formats. The updated CAD model can be loaded back the original CAD system with all the design features intact.

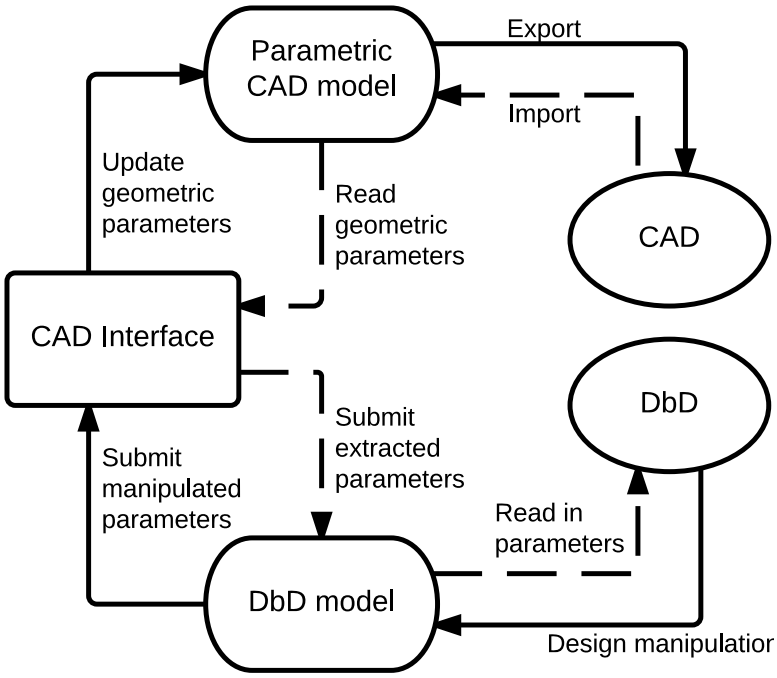


Figure 3.6: The Workflow of making design parameter changes. Dashed arrows denote the data flow from CAD to DBD, while the solid arrows denote the data flow from DBD to CAD.

3.3.2 Integration 2: Running FEA Simulations and Reading Output Data

In integration 1, the CAD interface is developed to allow generating designs with different geometric configurations. Here we discuss how to enable simulating and evaluating the designs with applied load and/or boundary conditions. More specifically, the goal of this integration is to establish a workflow to allow:

- Running FEA simulations from a set of simulation parameters populated in the DBD.

- Reading extensive simulation output data into an efficient data form for further computation in the DBD to enable the direct manipulation.

Figure 3.7 is an expanded view of integration 2 shown in Fig. 3.5. The workflow of running FEA simulations is shown by the dashed arrows. Similar to integration 1, a parametric FEA model is used as an analysis model template. For Abaqus© 6.13, a template can either be an input file or a Python script where critical simulation parameters are identified. The FEA interface is able to read the template and control those parameters. To conduct parametric studies, DBD generates a set of simulation parameters to be submitted to FEA interface. The custom FEA interface uses the parameter values to populate simulation jobs and calls the FEA kernel to execute the jobs. For example, to study the effects of the tip load and the beam thickness on deflection angle, one can specify ranges and numbers of data points for both input parameters in DBD. A set of simulation parameters is generated for the FEA interface to run simulations of all the parameter configurations.

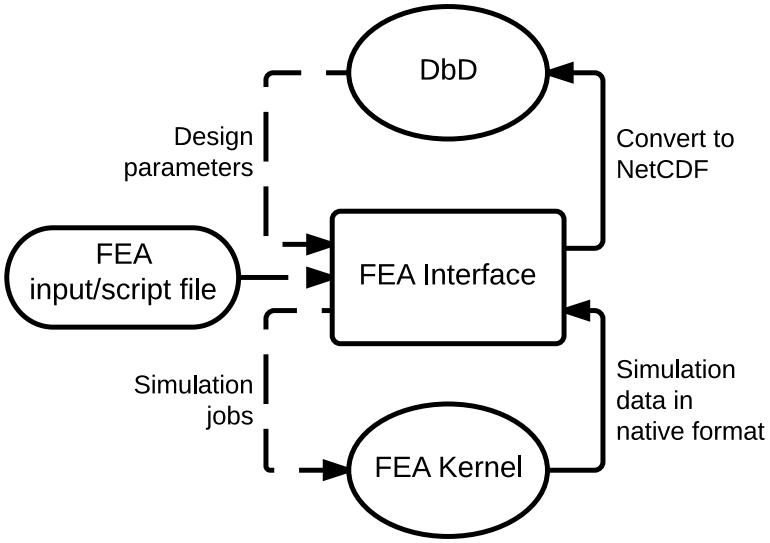


Figure 3.7: The Workflow to run FEA simulations and receive output data. The data flows of running simulations and reading output results are shown by dashed and solid arrows, respectively.

FEA results are typically stored in a custom binary format. The format is not directly readable as its internal data structure is not revealed. Therefore, an interface is needed for extracting desired data fields. Most commercial FEA packages provide a programming or scripting interface to export result data for further post-processing outside the native user environment. However, this type of interface is not designed for efficiently handling large amount of output data sets. Therefore, we proposed a one-time export of the FEA results and conversion of the data into an open-source data format, Network Common Data Form (NetCDF) [43]. The NetCDF is a self-describing data form, which can be used across different computer platforms. A set of NetCDF software libraries provides a high-efficiency data processing for large design spaces that work within our design framework. The exporting and conversion process is shown by the solid arrows in Fig. 3.7. Upon the completion of the simulations, the FEA interface receives all output results from the FEA kernel, reads desired data fields and coverts them into the NetCDF format. The NetCDF data sets are transferred back to DBD for further computation. This one time export and conversion, although requires extra time at the beginning, will later significantly reduces time for every access to the FEA data sets.

In addition to the efficiency, the NetCDF format significantly reduces the required data size for storing FEA results. A 1 GB Abaqus structural analysis output database can be reduced to 200 mb after the conversion. Adding/modifying data groups, variables and their content is also allowed. Thus, the data format can be further adjusted to suit any type of FEA output data sets.

A customized NetCDF hierarchy for structural FEA simulations is shown in Fig. D.1. The data set is represented by groups and variables, which are shown as rectangle and oval respectively. A group can include multiple sub-groups and variables. This allows a data set to contain as many layers as needed. The hierarchy is designed for general structural FEA simulations. However, with the flexibility of adding/modifying groups, variables and their dimensions, the NetCDF format can suit any type of FEA output data sets. The detailed implementation of the NetCDF-Abaqus interface can be found in Appendix D.

3.3.3 Integration 3: Parallel Computation in HPC System

HPC provides two important features to largely reduce time for generating extensive simulation data. One is the batch job system, which allows multiple simulation jobs to be executed simultaneously. The other is the parallel computation, which decomposes a simulation into sub-domains and each of them can be solved across its compute nodes and cores. A HPC system is integrated into the framework so that parallel computation and simultaneous job executions are enabled. The HPC system is provided by Minnesota Supercomputing Institute (MSI) at the University of Minnesota, which is equipped with 1,134 compute nodes and 8,744 cores. Each node has two quad-core 2.8 GHz Intel Xeon X5560 processors and 24 GB main memory. The Abaqus kernel is installed on the HPC system.

We established a protocol to automatically schedule the generated simulation jobs to the HPC system. Every simulation job requests compute nodes and cores from the HPC system to run the simulation. Theoretically, all scheduled simulation jobs can be executed simultaneously. Due to local constraints, limits are set for numbers of simultaneous runs and processors used in parallel computation in this research.

3.4 Results

In this section, we demonstrate the implemented simulation-based design framework. An example of a bending biopsy needle was introduced to the framework. The DBD-CAD integration allowed manipulating the needle geometry in the DBD environment based on a parametric CAD model. Next, a three-dimensional design space including 460 design configurations was sampled. Uses cases of forward and inverse design were provided to show how to use direct manipulation on the visualization to find desired solutions.

3.4.1 An Example of Bending a Biopsy Needle

A VAB procedure begins with inserting needle into the breast to reach the lesion area. During the insertion, the needle is usually exerted a force perpendicular on its outer wall at the region where it contacts the breast skin, tissue or sometimes chest bone. We

simplified the problem by assuming a perpendicular, concentrated force the was exerted to the needle tip. The manitude of the force was set to $2N$. We also assumed that the proximal end of the needle is fully constrained as it usually connects to a handpiece, which is hold by hand or machine.

A parametric CAD model of a VAB needle was created in SolidWorks 2013. As shown in Fig. 3.8, the needle cannula is hollow inside and has an opening window near its distal end. The geometric parameters identified in the model were needle gauge (outer radius of the cannula, OD), inner radius of the cannula (ID) and aperture length. By default, the outer cannula is a 7G needle with regular wall thickness, which results in an OD of $4.57mm$ and an ID of $3.81mm$. The default window length is $20mm$. The length of needle is fixed in this example. The audience is referred to Appendix A for a table of the needle gauge, radii and wall thickness options.

Figure 3.8: Parametric CAD model of the VAB outer cannula

The three geometric parameters were used as input parameters to generate 460 needle configurations. With the prescribed boundary condition (fixed proximal end) and tip load, the bending test was simulated for all of the needle configurations. The simulations output the von Mises stress field over the entire needle region. The simulation output data was integrated into DBD. A default configuration of a 7G needle with regular wall thickness is shown in Fig. 3.9, where the handpiece is manually created by the DBD program for an integrated view of the VAB instrument. The red arrow near the tip indicates the location of the perpendicular force. The predicted von Mises stress field corresponding to the default configuration is also displayed.

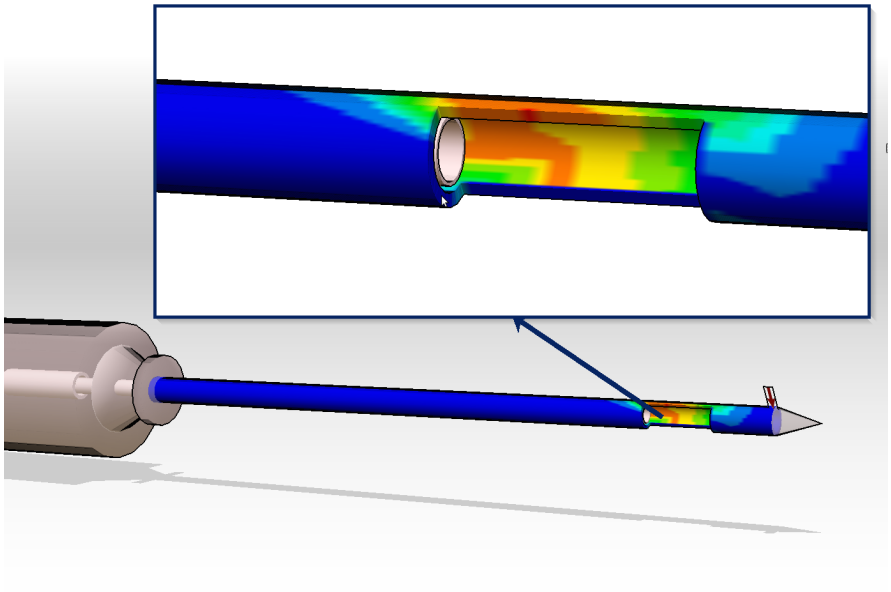


Figure 3.9: The default design configuration and its bending simulations shown in DBD

3.4.2 Manipulating the Needle Design

We were able to control the three design parameters in DBD using drag operations. An example of changing the length of the opening window is shown in Fig. 3.10. In the DBD environment shown on the left, we dragged an edge of the opening window toward the opposite one, as illustrated by the red arrow. This drag operation was interpreted as decreasing the window length. During the drag, the SolidWorks model was being synchronized by the geometric parameters updating through the CAD interface as the solid arrows shown in Fig. 3.6. In this example, the window length was decreased from $30mm$ to $25mm$ and finally to $20mm$. The resulting SolidWorks models are shown on the right of Fig. 3.10. The other two geometric parameters, OD and ID, can also be manipulated by drag operations. To increase or decrease either of them, one can drag on the circle and move away from or toward their shared center.

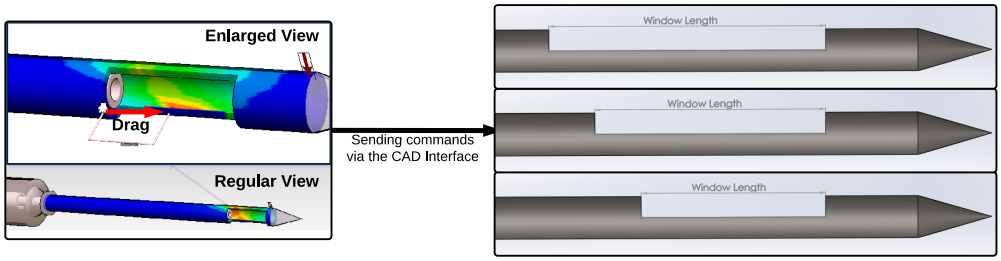


Figure 3.10: Dragging to decrease the window length in DBD with real-time CAD synchronization

3.4.3 Sampling the Design Space

An Abaqus Python script for simulating the bending test was pre-developed and input to the FEA interface (recall the workflow shown in Fig. 3.5). The script was a series of command lines that were used to call an Abaqus kernel to build and run the FEA model for the bending test step-by-step. This process included five main steps as the follows:

1. Importing the needle geometry created by the SolidWorks.
2. Assigning material properties. The properties of stainless steel were used: Young's modulus $= 200 \times 10^9 Pa$, Poisson's ratio $= 0.25$.
3. Generating mesh on the entire geometry.
4. Specifying load and boundary conditions, including the tip load and the fixed condition at the proximal surface of the needle.
5. Running the simulation.

Next, DBD generated a set of design parameters (i.e. all 460 geometric configurations) and submitted it to the FEA interface. The FEA interface parsed the content and generated 460 simulation jobs, each of the jobs was a group of input files containing a command text, a copy of the FEA script and a CAD model of the needle geometry. The command text included lines of Linux commands to initiate the Abaqus module in the

HPC system and executed the Abaqus Python script to run the bending simulations. The CAD models were in STEP format, which was converted from the native SolidWorks format. This conversion was needed for Abaqus to read the geometry. Details about the job content and the complete Python script can be found in the Appendix C.

The 460 simulation jobs were submitted to the Portable Batch System (PBS) queue provided by the HPC system. In this example, the simultaneous job executions were 8. Parallel computation was not enabled as it would not improve much efficiency in this simple bending simulation. The simulation time for each job was less than a minute and all 460 simulation jobs were finished in about a hour.

3.4.4 Visualizing and Interacting with the Simulation Data

Following the workflow shown shown in Fig. 3.7, 460 FEA simulation data sets were converted to NetCDF format and transferred to DBD. Next, DBD computed a sampling of the design space that represented both the input and output parameter spaces (i.e. 460 design instances, each includes its geometric configuration and predicted von Mises stress field). Each pair of the design instances was compared to find their differences and determine the image transition from one to the other (e.g. interpolating between two stress contours). Therefore, the direct manipulation via visualization was enabled.

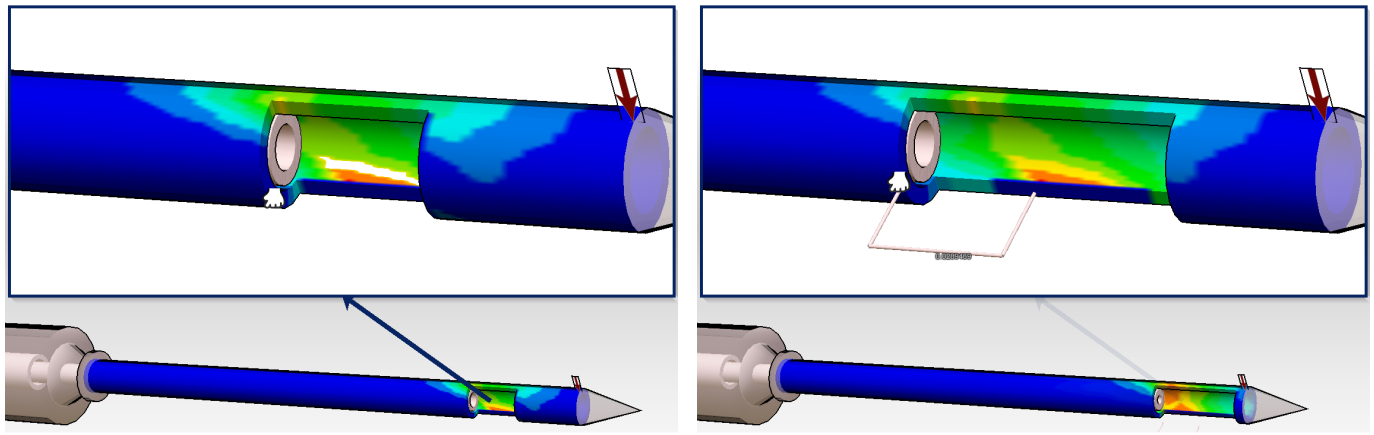
An example of the forward design is shown in Fig. 3.11. Similar to the operation we performed to modify the needle geometry, we dragged an edge of the opening window to increase its length. This operation switched the view from the original design instance (3.11a) to a new design instance (3.11b), which had a new longer window length and the same needle radii. The stress contour also smoothly transitioned from the original one and the new one, thanks to the pre-computation of the design space sampling. Note that the same forward manipulation strategy can be used to change the outer and inner radii and see the resulting stress contour.

In a demonstration of the inverse manipulation (see Fig. 3.12), we set a goal to move a high stress region away from a long edge of the opening window. As shown in Fig. 3.12a, the high stress region occurred near the opening window in the original design instance. To find an alternative design that satisfied our goal, we right-clicked at the high stress region to invoke preview bubbles for design suggestions. In Fig. 3.12b, seven closest design instances were shown by preview bubbles. Recall that the relative

distance between two design instances is determined using the distance metrics and the weighting:

$$d(A,B) = \sum_{i=1}^{n} \sqrt{w_i * (D(A_i, B_i))^2} \qquad \text{(3.1 revisited)}$$

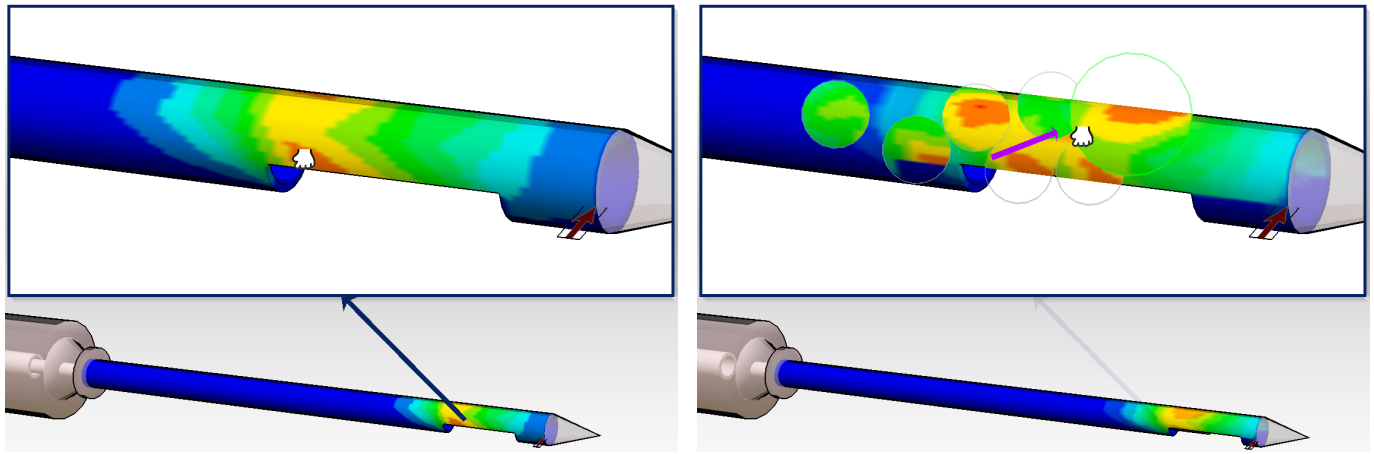
Note that in this case the three design input parameters and the von Mises stress field were equally weighted. We switched into the design instance presented by the preview bubble on the most right. The result showed the high stress region was much farther away from the opening window than that in the original design instance, as seen on the bottom of Fig, 3.12b. This design instance had the same outer radius, but had a thicker wall and a longer window length.



(a) Original design instance with window length = L_{w1}

(b) New design instance with window length = L_{w2}

Figure 3.11: The original design instance had a shorter window length (L_{w1}). By dragging the left edge of the window away from the right edge, the window length was decreased to L_{w2}. Because of the pre-computed sampling of the design space, a smooth transition of the stress contour from Fig. 3.11a to Fig. 3.11b was displayed. Using the forward manipulation strategy, the tendency of how the high or low stress region migrates with responding to the changes of each input parameter can be clearly observed.



(a) Original design instance

(b) New design instance

Figure 3.12: In this example, the goal was to move a high stress region away from the opening window. As shown in Fig. 3.12a, the high stress region originally occurred near the opening window (the cursor's location). To find design alternatives, we right-clicked at the high stress region to invoke the inverse manipulation mode. In Fig. 3.12b, seven preview bubbles were shown and each of them provided an enlarged view of a design instance at the local region. These design instances were computed to have the closest distances from the original one in the design space. Judging from the local stress contours shown in the preview bubbles, the one on the most right showed the high stress region would possibly be moved to the farthest distance from the window. Thus, we moved the cursor to the center of that preview bubble and the display switched to the corresponding design instance. The new design instance was shown on the bottom of Fig. 3.12b, which had a thicker needle wall and a longer window length.

3.5 Conclusion

A simulation-based design framework for large-scale design exploration was implemented. This framework integrated CAD and FEA systems with DBD to enable a Human-in-the-loop design process. An example of a biopsy needle under bending was used to demonstrate the functionality. We identified three design input parameters (the inner radius, outer radius and window length of the needle) and a performance attribute (von Mises stress field) to populate 460 design instances. Differences between the design instances were computed and a three-dimensional design space was sampled. Use cases of forward and inverse design strategies were demonstrated in the DBD environment.

The CAD interface in the framework enabled a two-way communication between SolidWorks and DBD. From SolidWorks to DBD, a parametric CAD model can be loaded and displayed in the DBD environment. In the opposite direction, manipulating the needle geometry is made possible by controlling the three geometric parameters via SolidWorks API. Furthermore, any desired design can be exported as a native or neutral CAD file. For example, 460 STEP files corresponding to the 460 different geometric configurations of the biopsy needle were generated in the process of design space population. The CAD-DBD integration is based on the SolidWorks API to control the native CAD model, the model conversion is not required so that the information loss is avoided. A real-time CAD synchronization is also achieved. Drag operations that manipulate the geometry in DBD are immediately updated to the CAD model. However, there are limitations. First of all, the first time loading the parametric CAD model into the DBD environment involves manual effort. The CAD geometry and its parameters need to be manually translated to defined in the DBD program. It is possible to make the loading process fully automatic if the geometry can be formed by primitive shapes. However, this research was not intended to focus on creating new 3D modeling methods or duplicating existing CAD functionality, but to enable a workflow to import and export CAD models in the framework.

The FEA interface serves as a middleware between DBD and the FEA/HPC system. It automatically parses the input parameter set from DBD, generates Abaqus simulation jobs in the HPC system and returns the simulation results. The approach is based on a FEA template, which can be a script or input file. The FEA interface has the ability to

create as many instances of simulations as specified ranges and numbers of simulation input parameters. This ability is currently limited to FEA models with less complex geometry. More specifically, the geometry of a FEA model has to be prepared to be ready for the automatic mesh generation before it is input to the FEA interface. In some cases, the geometry is too complicated to be automatically set up by a script and manual work is required. For example, automatic mesh generation on irregular geometry often results in poor-quality elements. In such situation, extensive manual mesh refinement at local regions is usually required. This limitation will be further explained in Chapter 5 where a realistic VAB cutting simulation is introduced to the framework.

The 460 simulation jobs were executed in the HPC system. As the low complexity of the bending simulation, parallel computation was not used as it would not improve the efficiency. However, thanks to the batch system, the simulation jobs were separated into 8 groups and the 8 job lines were executed simultaneously. The total time to complete the simulations was reduced from six hours to less than a hour. The improvement would be much more significant when dealing with the realistic tissue-cutter interaction model, which will be discussed in Chapter 5.

In summary, the implementation of the proposed framework was presented in this chapter. We provided a complete process of populating a design space using an example of bending biopsy needle. Functionality of DBD including forward and inverse design strategies were demonstrated . In next chapter, development of the realistic VAB cutting model is presented. In Chapter 5, the model is introduced to the framework and more advanced use cases are provided and discussed.

Chapter 4

Modeling and Simulating Tissue-Cutter Interaction during VAB for Predicting Tissue Reaction Force/Torque

In this Chapter, we present a tissue-cutter interaction finite element model for predicting tissue reaction force/torque during the cutting process of a VAB procedure. This model simulates a common VAB cutting method in which the cutting needle removes breast tissue samples with simultaneous rotation and translation in high speeds. Constitutive breast tissue models were collected from the literature and used to simulate the tissue behavior. We first refine the tissue cutting model to achieve the mesh convergence and the conservation of the system energy. Parametric studies are then conducted to understand the effects of the tissue properties and the cutting speeds on the tissue reaction force/torque and the tissue sample quality. The predictions are compared with a recent experimental study and the results of the comparisons are discussed.

4.1 Problem Description

VAB tools use a coaxial needle technology to retrieve tissue samples. Figure 4.1 shows a photograph of a Hologic ATEC© VAB device and its main components are annotated. The coaxial needle system has an outer cannula and an inner cannula (cutter). The outer cannula is a hollow cylinder with a sharp tip and an opening window (aperture) in the distal wall. To begin a VAB procedure, the needle is inserted into the breast with handheld ultrasound image guidance. The needle position has to be confirmed in the ultrasound screen to make sure the target lesion tissue is correctly drawn into the aperture. Next, the vacuum draws tissue into the aperture and a tissue sample is separated from the main breast tissue by the cutter. Finally, the tissue sample is transported via vacuum to the sample collector. This sampling process is normally repeated five to six times by rotating the needle shaft to complete a standard 360-degree rotation.

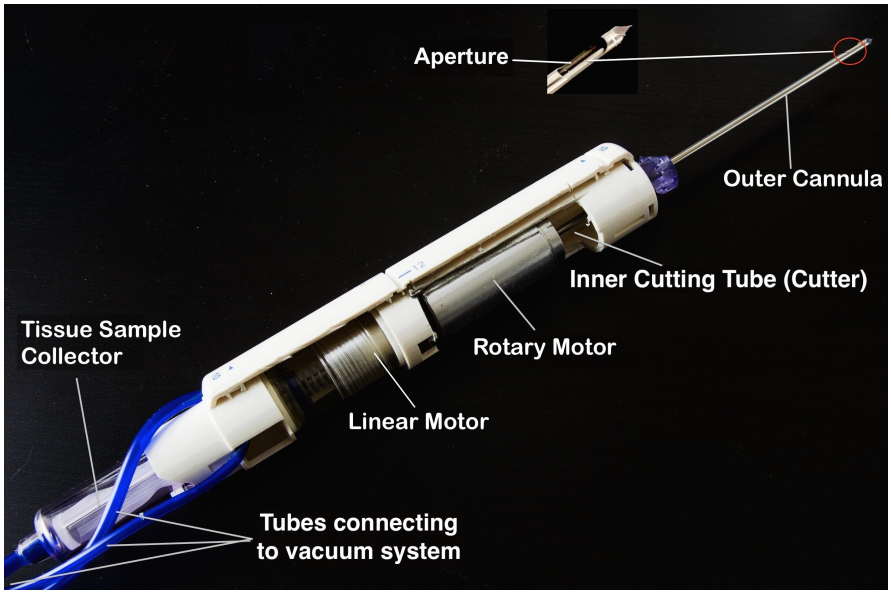


Figure 4.1: The Hologic ATEC VAB device

We focus on the rotary cutting method, which is utilized in a majority of the VAB tools, including Ethicon Mammotome, Bard Vacora, Hologic ATEC and Hologic Eviva

[24]. In this particular method, the inner cutter removes tissue samples with simultaneous rotation and translation in the axial direction. A diagram describing the cutting process is shown in Fig. 4.2. The cutter is at its initial position, which is D mm away from the tissue. The cutter is accelerated to the desired rotary and translational speeds before coming into contact with the tissue. After first contact, the cutter continues to traverse through the tissue a distance L with constant cutting speeds. The stroke S of the cutter is the summation of D and L. The length of the aperture is defined as L_w. In a successful scenario, the tissue will be deformed and progressively ruptured until a tissue sample is fully separated from the main tissue.

The cutter is driven by the combination of a linear motor and a rotary motor (typically DC or pneumatic motors). Enough axial force and torque have to be supplied to overcome the tissue reaction forces. Therefore, predictions of the tissue reaction force and moment are important to the sizing of the motors, which affects the ergonomics of the handpiece design. Larger motors may cut tissue faster, but as the VAB device is handheld, the handpiece then may be too heavy and awkward. Therefore, understanding the tissue reaction force and torque under different cutting speeds would help find the best balance between the cutting ability and handpiece ergonomics.

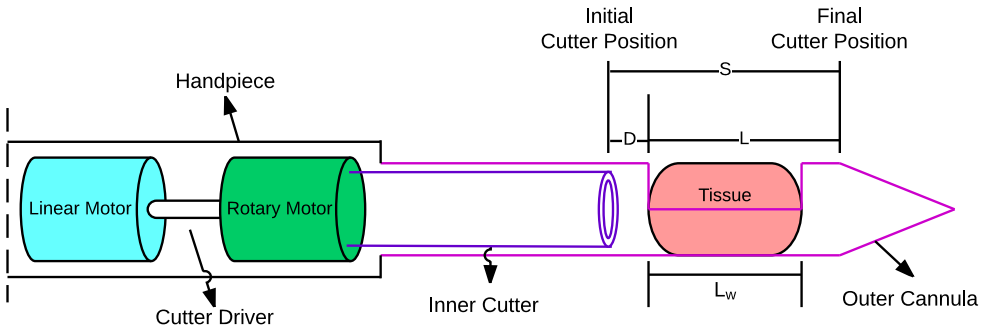


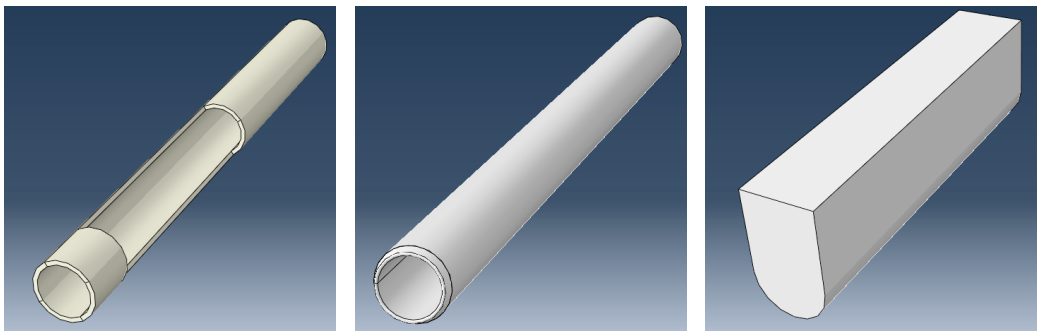
Figure 4.2: The tissue cutting problem in the breast biopsy procedure

4.2 The FEA Model of VAB Cutting

VAB cutting is a transient dynamic process, which involves complicated tissue-cutter contact, nonlinear tissue deformation and fracture. In finite element analysis (FEA), explicit time integration scheme is better suited to this type of problem. Unlike implicit time integration scheme, the explicit integration method does not require iterations and convergence checking in each time increment. This largely increases the robustness and the efficiency of the solution for the dynamic simulation with complex tissue-cutter contact required here. However, it is crucial to pay close attention to the indicators of numerical errors, such as the mesh convergence and conservation of energy for the entire dynamic system. To develop this VAB cutting model, the Abaqus/Explicit analyzer of Abaqus 6.13 is selected to solve the dynamic system including transient cutting impact and nonlinear tissue responses. Numerical errors are evaluated to ensure the accuracy of the solution.

4.2.1 Geometry

The geometry of the coaxial needle system is created in SolidWorks 2013 and imported to Abaqus CAE as two separated parts: the outer cannula and the inner cutter. The outer cannula is a 6G needle with an outer diameter of 5.16 mm and an inner diameter of 4.39 mm, as shown in Fig. 4.3a. The needle tip is eliminated, as it does not interact with any other parts of the simulation. The inner cutter is a hollow cylinder with an outer diameter of 4.19 mm, an inner diameter of 3.43 mm and a sharpened distal surface (see Fig. 4.3b). The tissue geometry is created in the Abaqus CAE (the Abaqus user environment). We assumed that a portion of tissue is drawn via vacuum into the aperture and perfectly conforms to the inner surface of the aperture. As a result, the base of the tissue geometry is a half-circle combined with a rectangle. The radius of the half circle is equal to the inner radius of the outer cannula, while the rectangle has a width equal to inner diameter of the outer cannula and a length defined as 5 mm. The base is extruded by a height equal to the aperture length L_w to form the tissue geometry, as seen in Fig. 4.3c. The assembly of the coaxial needle and the tissue is shown in Fig. 4.4.



(a) Outer cannula (b) Cutter (c) Tissue

Figure 4.3: Three geometric parts of the FEA model

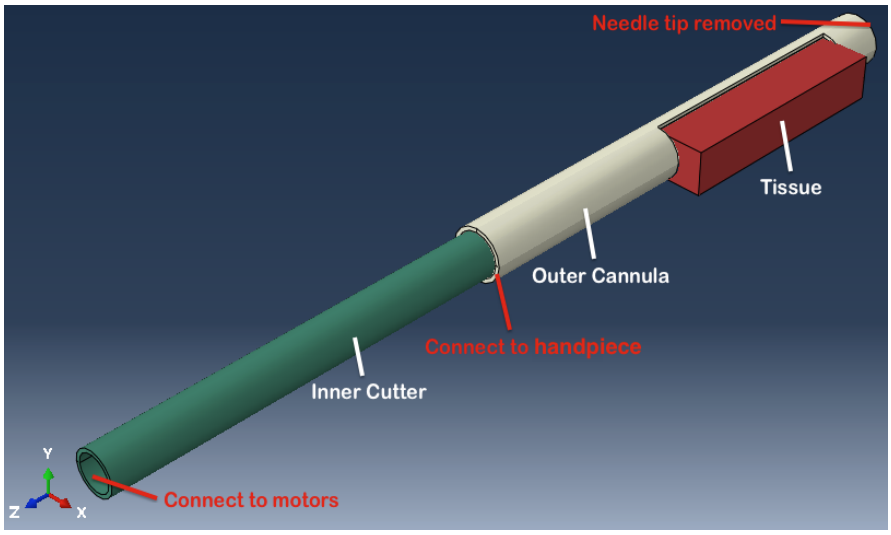


Figure 4.4: The geometric assembly of the FEA model

4.2.2 Material Properties

The outer cannula and the cutter are assumed to be rigid as their deformation is negligible and not of concern. The only information needed for these rigid bodies is the material density for the mass and inertia calculations, for which a value of 8000 kg/m^3 (stainless steel) is assigned to both of them.

The hyperelastic models for adipose and fibroglandular breast tissue are collected from the literature [36, 37, 38, 39, 40]. We found that the variation of the tissue stiffness

among these models is significant. To cover the extreme cases, the softest adipose tissue model and the toughest fibroglandular tissue model are selected. The adipose tissue model is a Neo-Hookean hyperelastic solid with a C1 coefficient of 0.08 kPa, developed by Rajagopal et al. The fibroglandular tissue model is also a Neo-Hookean solid with a C1 coefficient of 105 kPa, developed by del Palomar et al. We selected another Neo-Hookean model that has a medium stiffness of 3 kPa, also developed by del Palomar et al., as the base model for studying the effect of the slice-push ratio and the translational cutting speed. For the tumor tissue, the only available model for ductal carcinoma in situ (DCIS) [35] is used, which is a linear elastic model with a Young's modulus of 2162 kPa.

The fracture behavior of the breast tissue is required for the VAB cutting simulation, but it is not available in the literature. Therefore, the assignment of the fracture behavior is based on assumptions. The maximum tensile and shear strains that the tissue can resist are assigned as the fracture criteria. The element removal technology in Abaqus is enabled. Once either maximum strain is reached for a tissue mesh element, the element is removed and new tissue-cutter contact surfaces are created. In general, soft tissue can withstand more tensile strain than shear strain and softer tissue tends to resist more strain than tougher tissue. Based on this understanding, the maximum tensile strain is set to be double the maximum shear strain for each model and the adipose tissue models have higher maximums than the fibroglandular and tumor tissue models. A summary of the breast tissue models used in our study is provided in Table 4.1.

Table 4.1: Summary of the breast tissue models used in the simulations

Tissue Type	Constitutive Model	Tissue Stiffness (kPa)	Damage Criteria	Author
Adipose (base model)	Neo-Hookean	$C_1 = 3$	$\epsilon_v = 0.6$ $\epsilon_t = 0.3$	Rajagopal et al.
Adipose	Neo-Hookean	$C_1 = 0.08$	$\epsilon_v = 0.6$ $\epsilon_t = 0.3$	del Palomer et al.
Fibroglandular	Neo-Hookean	$C_1 = 105$	$\epsilon_v = 0.3$ $\epsilon_t = 0.15$	del Palomer et al.
Tumor	Linear elastic	$E = 2162$	$\epsilon_v = 0.3$ $\epsilon_t = 0.15$	Wellman et al.

*ϵ_v: maximum tensile strain, ϵ_t: maximum shear strain

4.2.3 Boundary Conditions

The rigid body conditions are set by applying rigid constraints to both the outer cannula and the cutter, limiting their motion to the motion of their reference points. The reference points are located at the center of the distal ring surfaces of the outer cannula and cutter cylinders, respectively. Because of the rigid body constraints, the boundary conditions of the outer cannula and the cutter are instead assigned to their reference points. The outer cannula should be fixed in place during the entire simulation. It is fully constrained at the proximal ring surface that connects to the handpiece and at its outer surface that is surrounded by the breast tissue. Therefore, zero linear and angular displacement is assigned to the reference point of the outer cannula. The reference point of the cutter is assigned constant translational and rotational speeds during its movement from initial to final position. When changing the translational speed (v_a), the simulation step time (t_{sim}) has to be increased or decreased accordingly, so that the cutter travels exactly through stroke S (see Eqn. 4.1).

$$t_{sim} = \frac{S}{v_a} \tag{4.1}$$

As the local tissue portion is initially part of the whole breast, we assume that the connecting interface (i.e. the tissue surface facing in the positive x direction, as shown in Fig. 4.5) is fixed. The load condition caused by the suction is simplified by fixing the tissue surface that contacts the inner wall of the outer cannula.

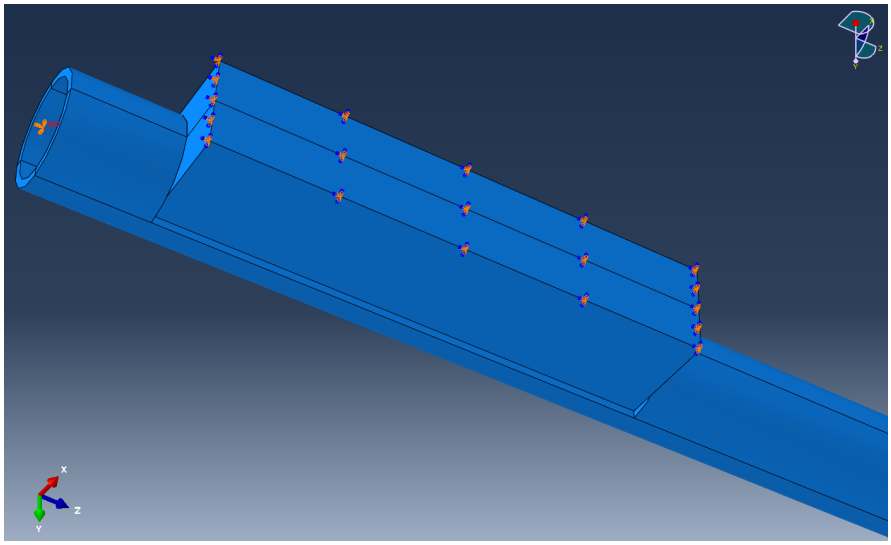


Figure 4.5: The fixed boundary conditions applied to the outer cannula and the tissue

4.2.4 Tissue-Cutter Interaction

Contact conditions are the most critical interaction properties in this VAB cutting model. The cutter is in contact with the exterior surface of the tissue and also the new interior tissue surfaces created by the fracture. A pure master-slave contact pair is assigned, in which the cutter surface is the master and all tissue exterior and interior surfaces are the slave.

A hard contact property is also assigned to the contact pair. This contact property is set to reduce possible numerical errors caused by the surface penetration. In Abaqus/Explicit, the kinematic enforcement for this type of contact condition ensures that the slave nodes (i.e. tissue mesh nodes in our case) do not penetrate the master surface (i.e. cutter surface in our case). Although it is still possible for the master surface to penetrate the slave surface, the penetration can be minimized using a sufficiently refined mesh. Friction is specified for the contact property in the tangential direction.

Considering the high cutting speeds, blood suction and rinsing provided in the VAB operation, a relatively small friction coefficient of 0.05 is specified.

4.2.5 Parametric Studies

We focus on investigating the effects of the translational cutting speed and the slice-push ratio on the tissue reaction force and torque in the axial direction. The relationship between the slice-push ratio and the tissue cutting force/torque has been studied under a low translational cutting speed [34]. This study shows that when the slice-push ratio varies between 0 and 5, its effect to the cutting forces is significant. The experiments were conducted on a phantom tissue with a linear elasticity of 20 kPa, and a translational cutting speed of 2 mm/s was applied, on the basis that the current clinical needle insertion rate is between 0.4 and 10 mm/s. Our model extend the study by including more sophisticated conditions, which are (1) realistic translational cutting speeds ($v_a = [10, 50, 100]\ mm/s$) of VAB that is much faster than the clinical needle insertion rate; (2) the nonlinear tissue models that predict more complicated behavior than the linear artificial tissue (see Table 4.1), and (3) an extended range of slice-push ratio (0-10) that covers higher rotational cutting speeds used by the VAB technology.

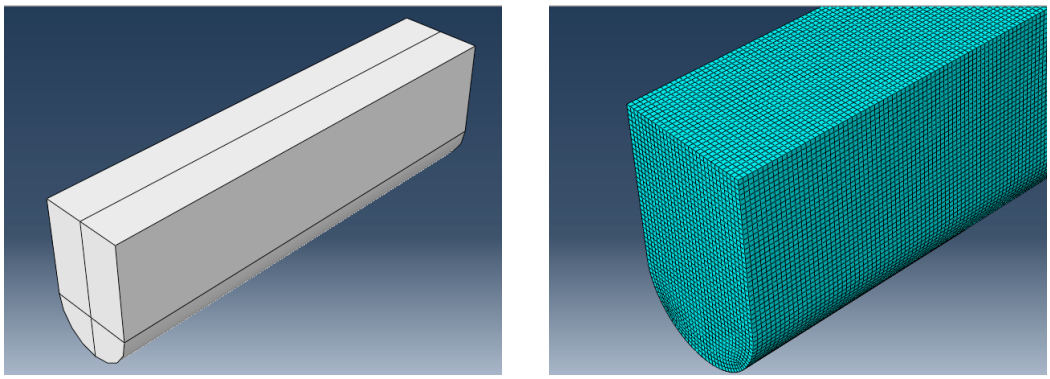
4.3 Results

4.3.1 Mesh Convergence

The mesh convergence was studied using the model with the highest translational cutting speed ($v_a = 100\ mm/s$) and slice-push ratio ($R = 10$) in our defined ranges. This was to make sure that the case of the most excessive and rapid tissue deformation and fracture were investigated. In addition, the computation time of this case was expected to be shorter than others with lower translational speeds as it only requires simulating through 0.25 seconds for the inner cutter to travel through stroke $S = 25\ mm$ (recall equation 4.1). Note that this is a half cycle of the VAB tissue cutting process. A full cycle also includes withdrawing the inner cutter to travel backward by S, which is not simulated here.

A 3D 8-node hexahedral element (C3D8R) was used to mesh the entire model. This

continuum element is recommended for explicit dynamic models to provide equivalent accuracy at less computation cost compared to tetrahedral elements. The hexahedral element is also a better choice for handling large deformation when a proper element size is assigned. For better computing efficiency, the tissue geometry was properly partitioned so that the structured meshing technique can be used (see Fig. 4.6).

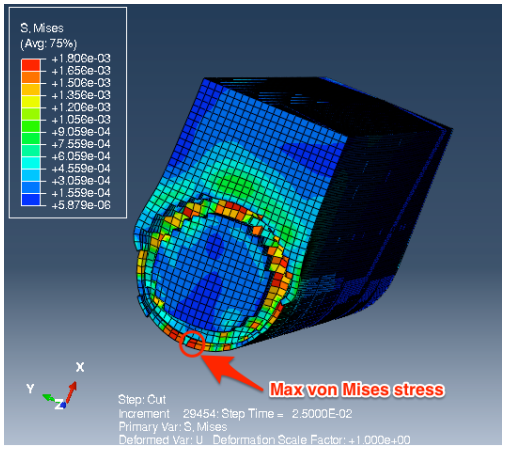


(a) Tissue partitions

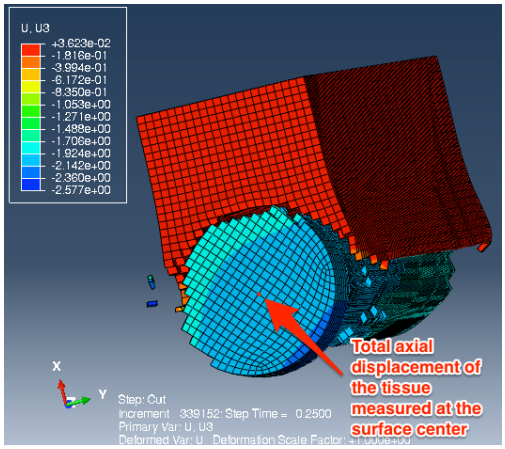
(b) The structure hex mesh generated on the tissue geometry

Figure 4.6: Partitioning the tissue geometry for structured hexahedral meshing

Since the outer cannula and the cutter were assigned as rigid bodies, there were no spatially distributed field variables (e.g. nodal displacements/stresses) computed on them. Therefore, we studied the mesh convergence on the tissue region by observing two field output variables: one was the maximum von Mises stress on the tissue surface that contacted the cutter. We looked at the time step in which the first tissue fracture was observed ($t = 0.025$ *seconds*), as shown in Fig. 4.7a; the other was the total displacement in the axial direction. We measured this at the center of the tissue surface facing the distal end, as shown in Fig. 4.7b.



(a) Maximum von Mises stress on tissue-cutter contact surface at the first time step that tissue fracture occurred



(b) The total axial displacement of the tissue measured at the center of the tissue surface facing the needle distal end

Figure 4.7: Two output field variables of interest observed for mesh convergence

In order to study the mesh convergence, we began with an element size of 0.50 mm and gradually decreased it to 0.12 mm. As shown in Table 4.2, using an element size smaller than or equal to 0.25 mm obtained very similar results for variable 1. The percentages of error were within 2%. However, variable 2 did not converge until the mesh size reached 0.20 mm. As far as both variables were concerned, we concluded that the mesh was converged at element size equal to 0.20 mm.

Table 4.2: Mesh convergence on two field variables

Element size L_e (mm)	Number of elements	Field variable 1: Max. von Mises (kPa) @ first occurrence of tissue removal	Field variable 2: Axial displacement (mm) @ center of distal tissue surface
0.50	5428	1.6702	3.0591
0.40	10904	1.7063	3.2743
0.30	25718	1.7442	2.4708
0.25	47288	1.7641	2.2631
0.20	87630	1.8060	1.7903
0.18	118784	1.8008	1.8040
0.12	404352	1.8288	1.8100

4.3.2 Convergence in Time Integration

It is also important to confirm that the explicit dynamic simulation is converged in the time domain. Insufficiently small time increment size can break the conservation of energy for the system because too much artificial energy is introduced. The total energy (E_{tot}) of the system, which is the sum of the internal energy (E_i), kinetic energy (E_k), frictional energy (E_{fr}) and external energy (E_{ex}), has to remain constant (see Eqn. 4.2).

$$E_{total} = E_i + E_k + E_{fr} + E_{ex} = constant \tag{4.2}$$

The explicit integration scheme does not require iterations to obtain a converged solution at each time increment. Instead, the time increment size must be smaller than a certain value, called the stable time increment or stability limit (Δt_{cr}). Eqn. 4.3 shows how this value is computed, where L_e is the characteristic length of the mesh element and p is the wave speed of the element material.

$$\Delta t_{cr} = \sqrt{\frac{L_e}{p}} \tag{4.3}$$

The wave speed p can be calculated using Eqn. 4.4, where E is the Young's modulus of the material and ρ is the density of the material.

$$p = \frac{E^2}{\rho} \tag{4.4}$$

The stability limit for the entire system is computed using the shortest element length $L_{e,min}$. In contact problems, the requirement of a highly refined mesh usually results in a very small $L_{e,min}$. With a constant wave speed p fixed to the material properties, the stability limit computed by Eqn. 4.3 can be extremely small. Based on the element size (0.20 mm) that we observed the mesh convergence, the estimated stability limit is 3.64^{-7} seconds. If running a simulation with a time increment not exceeding this limit, the entire solving process will take a state-of-the-art computing unit more than 300 hours of CPU time to complete. Even though we were able to use parallel computation technology to divide the simulation into 8 compute nodes,[1] the total computation time was still estimated to be more than 24 hours. Thus, the computing efficiency has to be improved further to make our parametric studies affordable.

Mass scaling is a technology that scales up the mass of the small elements to increase the stability limit. A common approach used in Abaqus is to set a target time increment (Δt_d) for a simulation. The stability limit of the entire system ($\Delta t_{cr,sys}$) will be monitored in each time increment. The masses of those elements that have Δt_{cr} less than Δt_d will be scaled to force $\Delta t_{cr,sys}$ greater than or equal to Δt_d. Therefore, the solving efficiency can be significantly improved. In our mesh convergence studies, Δt_d was set to 1×10^{-6} seconds so that the simulation runs were completed in reasonable time frames. For example, the running time of the simulation with an element size of 0.20 mm was reduced significantly from 24 hours to 7 hours.

The mass scaling technology is needed here for obtaining simulation results within reasonable time. However, it must be used carefully as it introduces artificial energy to the dynamic system through the mass increase. Too much artificial energy will cause the simulation to produce inaccurate results. Therefore, the conservation of energy of the system has to be monitored during the entire simulation. We investigated the quality of the solutions obtained using three different Δt_d. In addition to the one we used in the

[1] Using more than 8 compute nodes has provided no further decrease of computation time.

mesh convergence study (1×10^{-6}), the other two were 5×10^{-5} and 5×10^{-6}. These two larger Δt_d were used to test if the simulation time could be further reduced without breaking the conservation of energy constraint.

The total energy of the system E_{total} was computed over time to monitor the conservation of energy. The largest Δt_d (case 1: 5×10^{-5}) resulted in a significant increase of E_{total} with time. In the cases of the two smaller Δt_d (case 2: 5×10^{-6}, case 3: 1×10^{-6}), E_{total} remained constant. The three cases are shown in Fig. 4.8. This result suggests that the errors caused by artificial energy is negligible when a target time increment smaller than 5×10^{-6} is used.

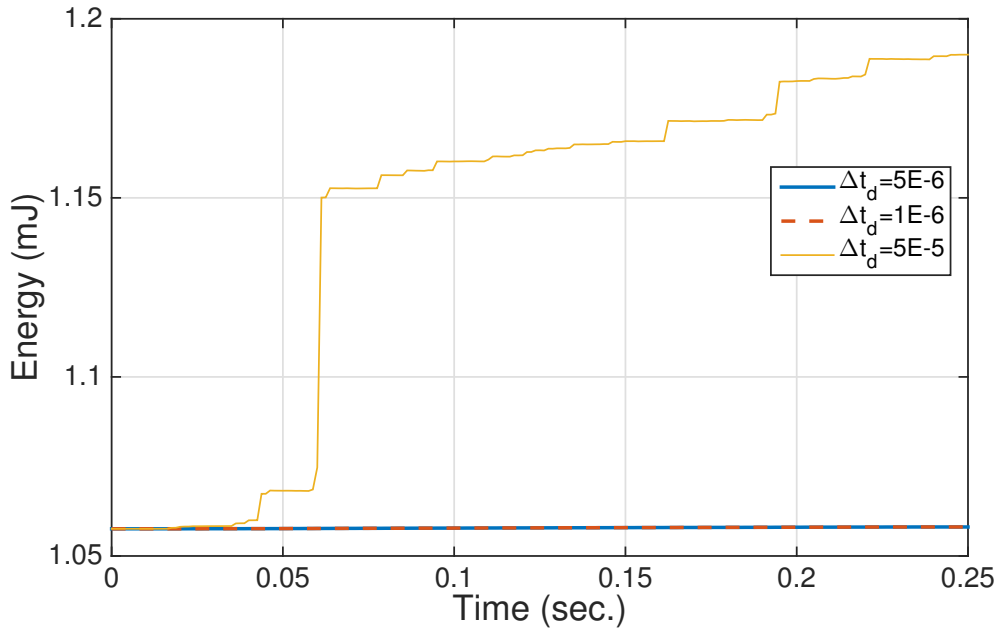


Figure 4.8: The fixed boundary conditions applied to the outer cannula and the tissue

We checked the two variables that were previously observed in the mesh convergence studies (see Table 4.2). The converged solution (i.e. element size = 0.20 mm) was used as the basis of the comparison (see case 3 in Table 4.3). The result shows that increasing Δt_d lowers the accuracy of the solution, especially for the variable 2. The accuracy of case 1 for variable 2 was clearly worse, while cases 2 and 3 provided similar results for

both variables. This finding agrees with the plot of the total energy shown in Fig. 4.8.

Table 4.3: The effect of mass scaling on the mesh convergence

Target time increment Δt_d	Field Variable 1: Max. von Mises (kPa) @ first occurrence of tissue removal	Field variable 2: Axial displacement (mm) @ center of distal tissue surface
Case 1: 5×10^{-5}	1.7703	1.2684
Case 2: 5×10^{-6}	1.7981	1.7624
Case 3: 1×10^{-6}	1.8060	1.7903

Next, we looked at the tissue reactions in the same three cases. The tissue reactions, including the tissue reaction force and torque, are time-varying variables computed at a total of 201 time steps over the entire simulation time. These two variables are good candidates for the sensitivity analysis of the time increment size. In the plots of the tissue reaction force and torque (Fig. 4.9 and 4.10), the fluctuation diminishes as Δt_d becomes smaller. Comparing with case 3, the noise is significant in case 1, but is much improved in case 2. Δt_d used in case 1 was clearly too large and resulted in instability. This also confirms the result illustrated in Fig. 4.8.

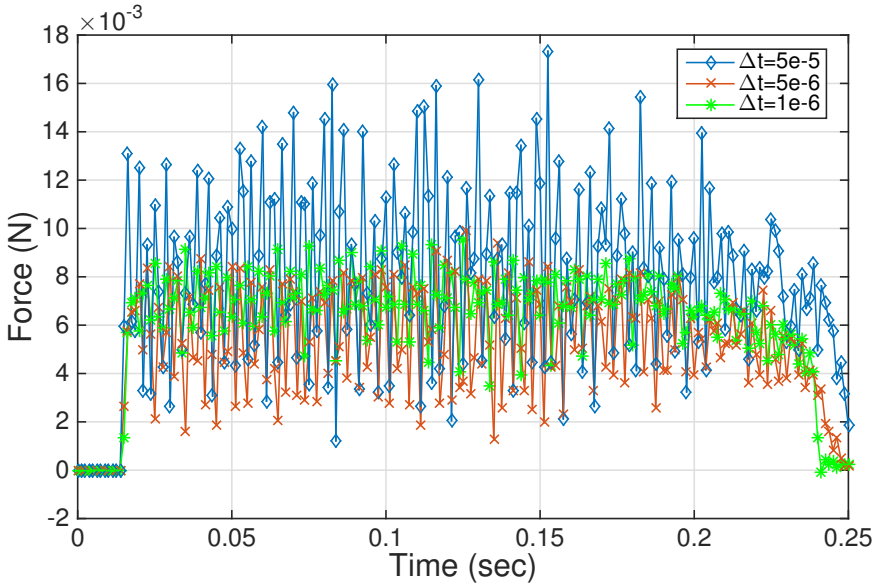


Figure 4.9: Axial reaction force on the cutter in different target time increment sizes

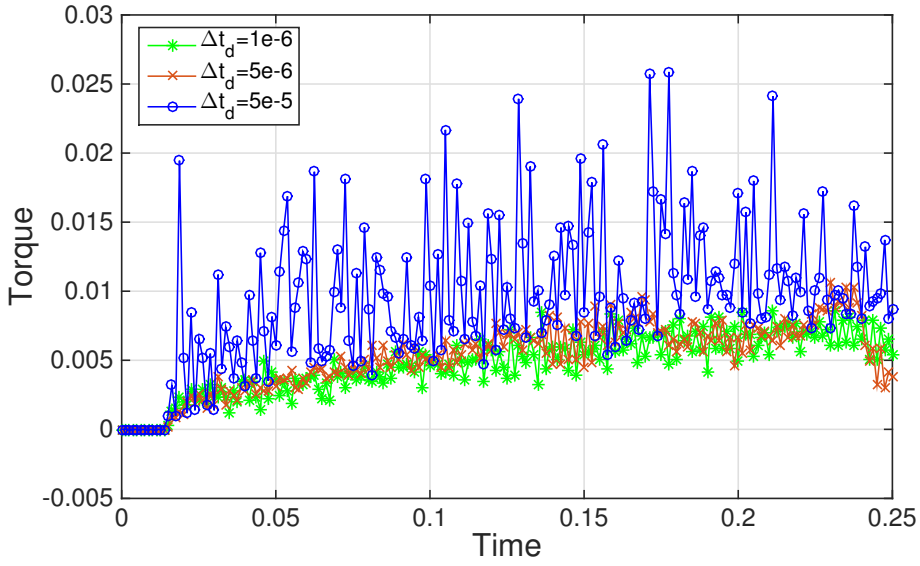


Figure 4.10: Axial reaction torque on the cutter in different target time increment sizes

Finally, a comparison of the computing efficiency in the same three cases is shown in Table 4.4. Each of these simulations was divided into 8 sub-domains and solved by 8 cores of 2.8 GHz Intel Xeon X5560 in parallel. In case 1 that had the largest Δt_d the simulation time was reduced to only 15 minutes, but the solution was found to be inaccurate due to the excess of artificial energy involved. In addition, significant noise appeared in the predicted tissue reactions. Δt_d used in case 2 was 10 times smaller than that in case 1. The total simulation time increased to 2.43 hours, but the computing efficiency was still largely improved compared to case 3.

Table 4.4: The comparison of computation efficiency using different stability limits

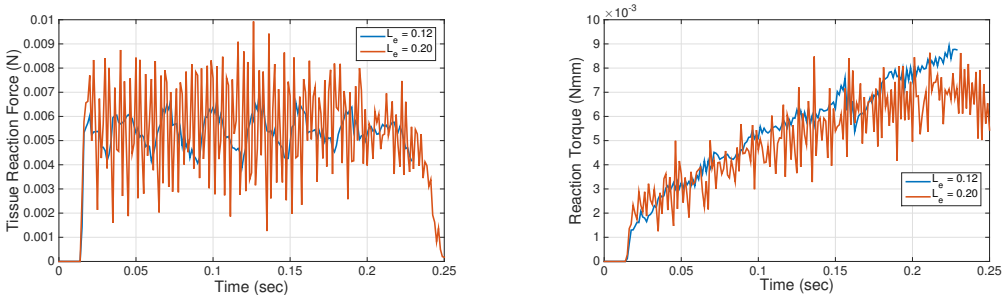
Threshold of stability limit	CPU time (hours)	Simulation time on 8 HPC cores (hours)
Case 1: 5×10^{-5}	1.80	0.25
Case 2: 5×10^{-6}	19.33	2.43
Case 3: 1×10^{-6}	111.67	14.00

In conclusion, setting Δt_d to 5×10^{-6} seconds (i.e. case 2 in this study) resulted in a

converged solution for the variables under consideration here. The solution reached the best balance between the noise level of tissue reactions and simulation time among the three cases being compared. Using a larger Δt_d (case 1) induced too much numerical error (including noise presented in the tissue reaction force/torque) in the simulation results, while a smaller Δt_d (case 3) significantly increased the simulation time.

4.3.3 Base Model for the Parametric Studies

As discussed previously, fluctuations were evident in the predicted tissue reaction force/-torque. The periodic release of the tissue resistance at each newly occurred fracture was believed to be partially responsible for these fluctuations. Another cause was the noise introduced by the numerical errors, which should be reduced. By decreasing the element size from 0.20 to 0.12 mm, a significant amount of the fluctuation was removed, as seen in Fig. 4.11.[2] Decreasing the target time increment has also been found to slightly lowered the fluctuation (see the green and red curves in both Fig. 4.9 and 4.10. However, either of these strategies would be about six times more expensive than the current simulation configuration (i.e. $L_e = 0.20$ and $\Delta t_d = 5 \times 10^{-6}$) in terms of simulation time.



(a) Noise reduction in the axial tissue reaction force

(b) Noise reduction in the axial tissue reaction torque

Figure 4.11: Noise reduction by further refining the mesh.

The efficiency provided by the current simulation configuration is favored, but the presence of the fluctuations in the tissue reaction force/torque needs to be reduced to

[2] The simulation with this very fine mesh was not fully completed (stopped at $t = 0.225sec$ of the full simulation time $t = 0.25sec$) due to its extremely long computing time.

ease the difficulty of comparing between different data sets. Therefore, we applied the Savitzky-Golay filter[3] [44] to smooth the output data. A sample result is shown in Fig. 4.12, where the original and smoothed tissue reaction force data is plotted in circle and cross markers, respectively. Although the smoothed data does not capture the maximum and minimum values of the tissue reaction force, it approximates a trend line of how the magnitude of the data changes over time by passing through the mean value of each fluctuation.

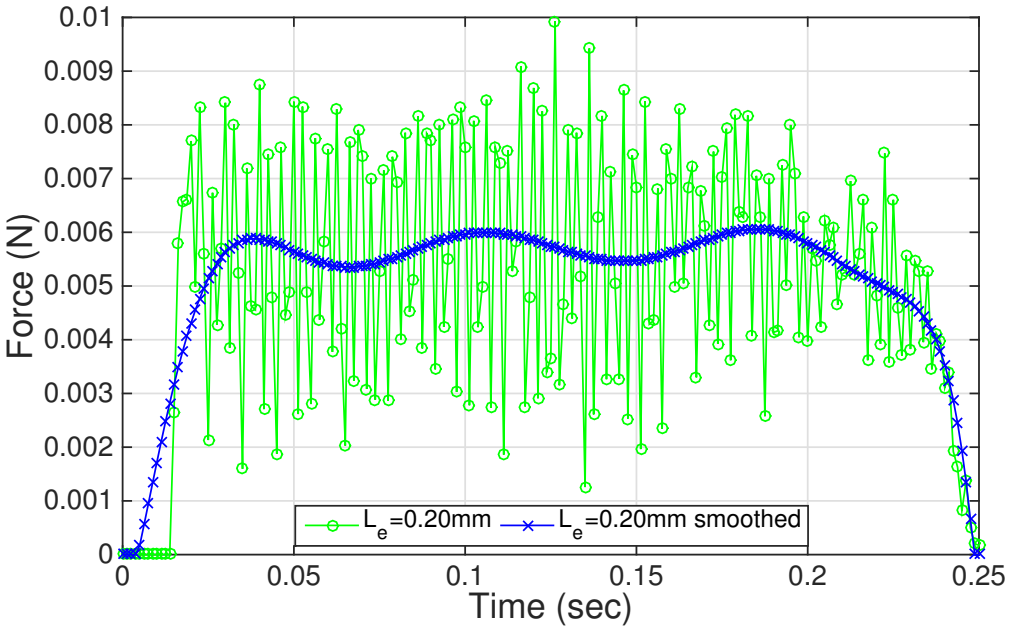


Figure 4.12: Comparing the original data curve of the axial tissue reaction force with the approximated trend line.

The trend line obtained from the current mesh ($L_e = 0.20\ mm$) was found similar to that of the very fine one ($L_e = 0.12\ mm$), as seen in Fig. 4.13. Percentage differences between the data points in the two trend lines were computed into a vector $\bar{e}$. As defined in Eqn. 4.5, a percentage difference between i-th data points in both data sets is the

[3] Savitzky-Golay filtering is a widely used, least-squares polynomial smoothing method to remove noise from data points and is available as a function in Matlab.

difference divided by the average. The mean and maximum percentage differences found in $\bar{e}$ were 2.85% and 6.80%, respectively. This indicates that increasing the mesh size from 0.12 mm to 0.20 mm would only slightly sacrifice the accuracy of the solution, but improve the solving efficiency by more than 10 times. Taking this trade-off into consideration, the current simulation configuration ($L_e = 0.20, \Delta t = 5 \times 10^{-6}$) was used to generate tissue reaction data for the parameter studies demonstrated in the following sections and the comparisons between the data sets were based on their trend lines.

$$e_i = \frac{abs(data1_i - data2_i)}{avg(data1_i + data2_i)} \tag{4.5}$$

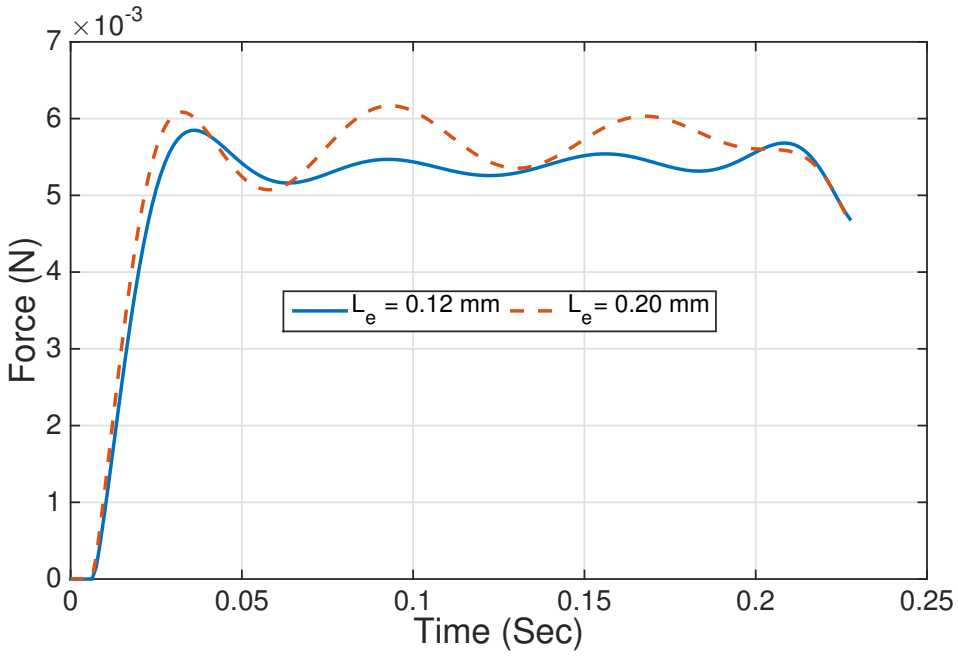
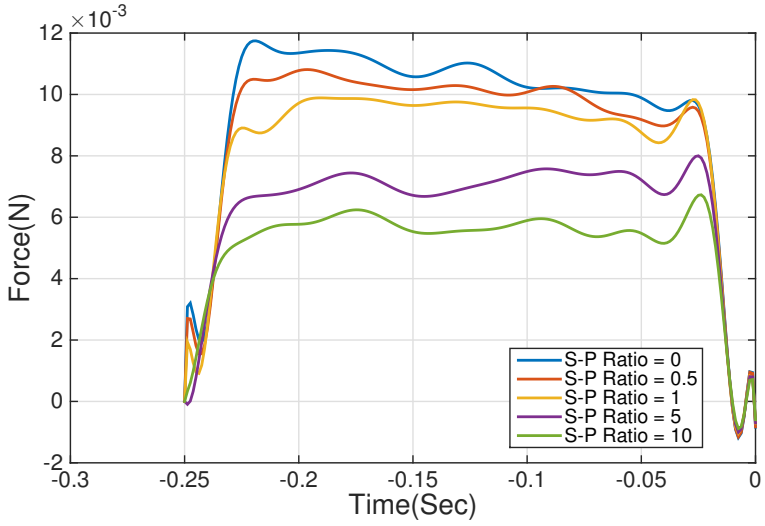


Figure 4.13: Comparing the trend lines of two data sets of axial tissue reaction force: current mesh ($L_e = 0.20mm$) and a very fine mesh ($L_e = 0.12mm$).

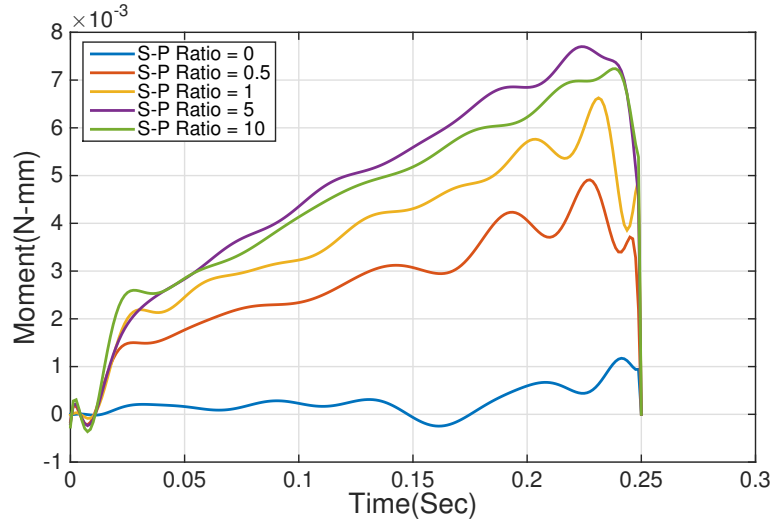
4.3.4 Effects of Slice-Push Ratio under Different Translational Cutting Speed

Tissue reactions under different combinations of rotational and translational cutting speeds were investigated. Five slice-push ratios: $R = [0, 0.5, 1.0, 5.0, 10]$ and three translational cutting speeds: $v_a = [100(high), 50(medium), 10(low)]\ mm/s$ were defined to generate 15 cutting speed configurations. The base tissue model (see Table 4.1) was used to simulate the cutting process under these 15 configurations. The tissue reaction force and torque on the cutter were computed so that 15 temporal data sets were obtained.

We first analyzed the effect of the slice-push ratio on the tissue reaction force and torque under the high translational cutting speed ($v_a = 100\ mm/s$). The predicted trend lines of tissue reaction force and torque under different slice-push ratios were plotted in Fig. 4.14. From Fig. 4.14a we found that increasing the slice-push ratio reduces the tissue reaction force and the reduction becomes significant when the slice-push ratio is larger than 1. In Fig 4.14b, the tissue reaction torque grows as the slice-push ratio becomes higher, but the growth is more obvious when the slice-push ratio is between 0 and 0.5. These tendencies of how the tissue reaction force and torque change along with the slice-push ratio generally agreed with the previous findings [34], but the pattern of change was different. The previous study concluded that both the tissue reaction force and torque change rapidly when the slice-push ratio is less than 1 and begin to stay constant when the slice-push ratio is larger than 5. In our study, the tissue reaction force dropped by an average of only 5.38% when the slice-push ratio increased from 0 to 1, while the average reductions when the slice-push ratio increased from 1 to 5 and from 5 to 10 were 24.10% and 19.05%. The slice-push effect on the tissue reaction torque that we found is more similar to the previous study. The increase of the tissue reaction torque in the slice-push ratio from 0 to 1 was 2.7 times the increase in the slice-push ratio from 1 to 5. However, it should also be noted that there was a slight decrease of the tissue reaction torque when the slice-push ratio increased from 5 to 10.



(a) Tissue reaction force

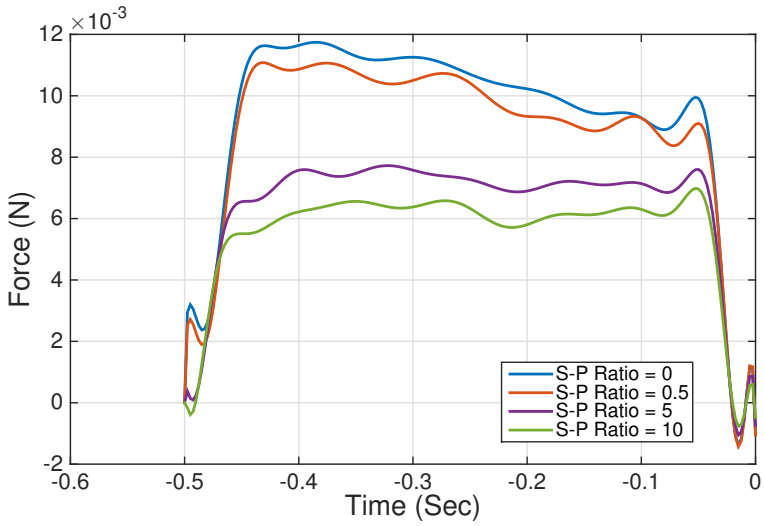


(b) Tissue reaction torque

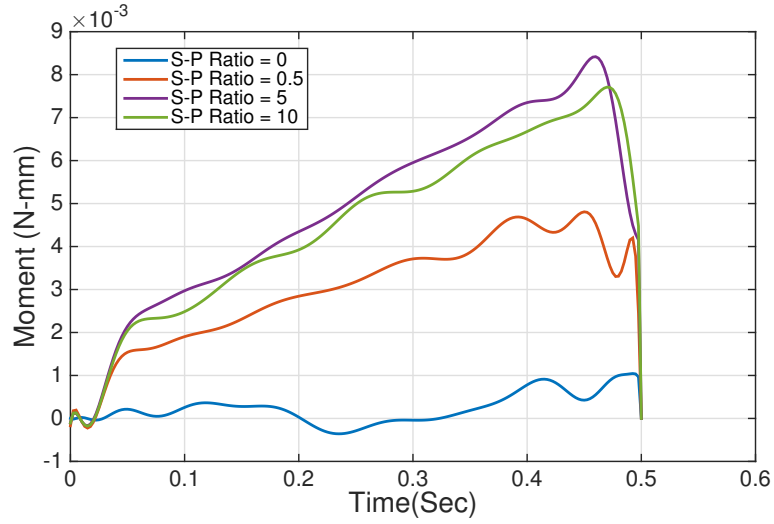
Figure 4.14: Comparing tissue reactions under different slice-push ratios (high translational speed)

Next the slice-push ratio cases with the medium translational cutting speed ($v_a = 50$ mm/s) were observed. Note that when the translational cutting speed is reduced from high (100 mm/s) to medium (50 mm/s), the travel time of the cutter is doubled, which

approximately doubles the simulation time. Therefore, we skipped the case with a slice-push ratio = 1 to save computing time and resources. The results of these cases were very similar to what we found in the cases with the high translational cutting speed, as seen in Fig. 4.15.



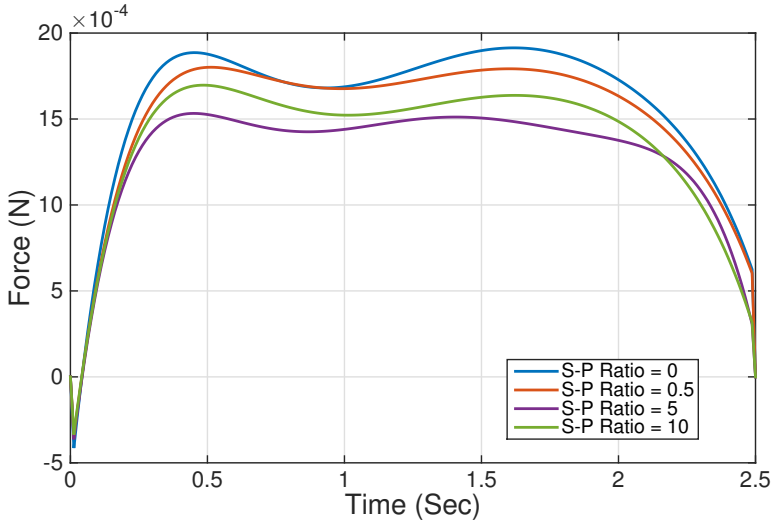
(a) Tissue reaction force



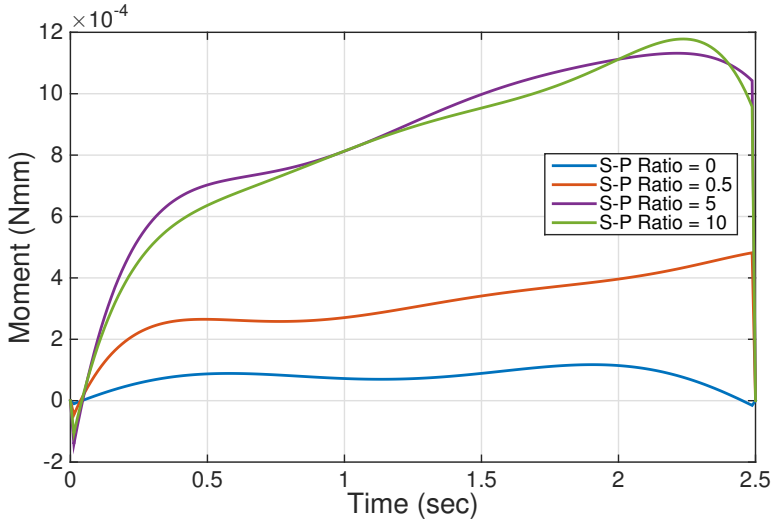
(b) Tissue reaction torque

Figure 4.15: Comparing tissue reactions under different slice-push ratios (medium translational speed)

The same four slice-push ratios with a low translational cutting speed ($v_a = 10$ mm) were simulated. The results are shown in Fig. 4.16. The trend lines and the slice-push ratio effects were different from the case studies of high and medium translational cutting speeds. The reaction force was less sensitive to the changing slice-push ratio. Moreover, the tissue reaction force rose when the slice-push ratio increased from 5 to 10. The response of the tissue reaction torque was similar, but its increase became larger when the slice-push ratio was between 0.5 and 5.



(a) Tissue reaction force



(b) Tissue reaction torque

Figure 4.16: Comparing tissue reactions under different slice-push ratios (low translational speed)

Increasing the slice-push ratio also improves the quality of the retrieved tissue samples. This was observed in the visualization of the deformed tissue shape at the final step of the simulation. We examined how the tissue was deformed and fractured at a cut plane that separates the entire assembly into two symmetrical parts (see Fig. 4.17).

A comparison of different slice-ratios is shown in Fig. 4.18. On the top is a case of the highest slice-push ratio = 10, while on the bottom is a case of a very low slice-push ratio = 0.1. Both cases have the same translational cutting speed ($v_a = 100\ mm/s$), so the rotation speeds were 477.33 rad/s (top) and 4.77 rad/s (bottom). The differences between these two cases are clearly evident. First, the high rotational cutting speed generates more shear force on the tissue surface so that the tissue is more impacted by the shear damage instead of the tensile damage. This can be seen at the proximal end (left-hand side of the figure) of the tissue sample, where with the higher slice-push ratio the tissue is less pushed in the axial direction. Secondly, the high slice-push ratio also tends to provide a cleaner cut at the distal end (right-hand side of the figure) of the tissue sample. In the case of high rotational cutting speed (top), the fracture occurs at the surrounding area of the cutter tip. However, when cutting with the low rotational speed (bottom), the tissue fracture propagates away from the cutter tip. This is because there is no sufficient shear strain produced at the tissue surface that directly contacts the cutter tip. Instead, the tissue is torn by the push in the axial direction. Enlarged views of the distal region are shown in Fig. 4.19, where a case with a medium slice-push ratio = 1 is added to the comparison. The result evidently shows how the cutting process can be gradually improved by increasing the slice-push ratio.

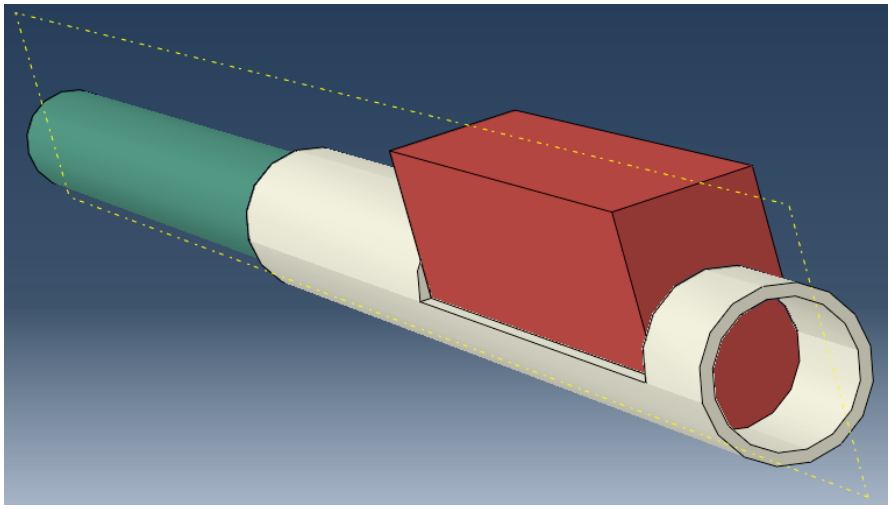


Figure 4.17: The symmetrical cut plane

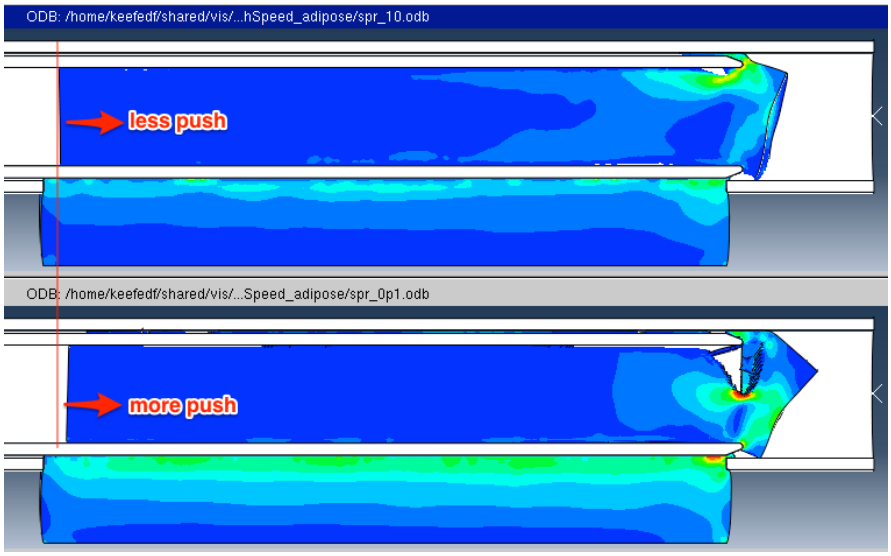
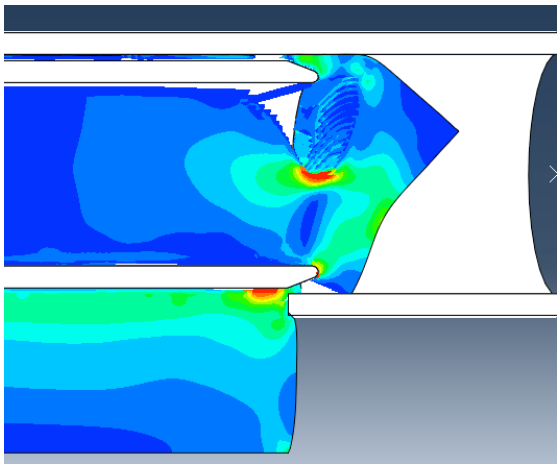
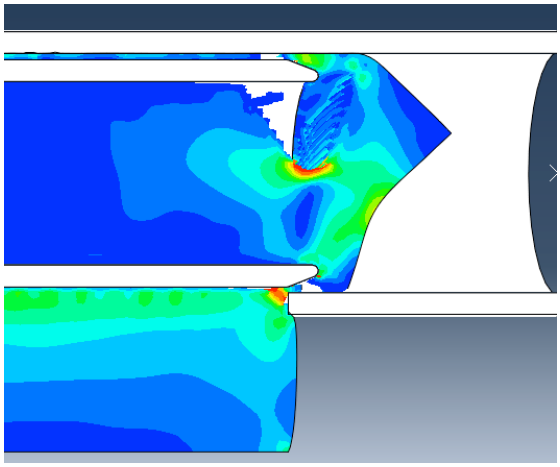


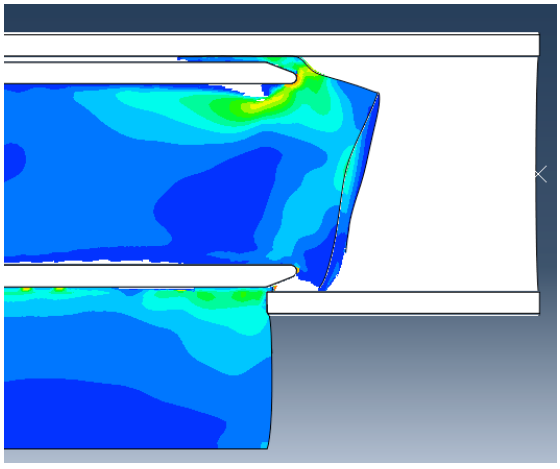
Figure 4.18: Comparing cutting with low and high slice-push ratios



(a) Slice-push ratio = 0.1



(b) Slice-push ratio = 1



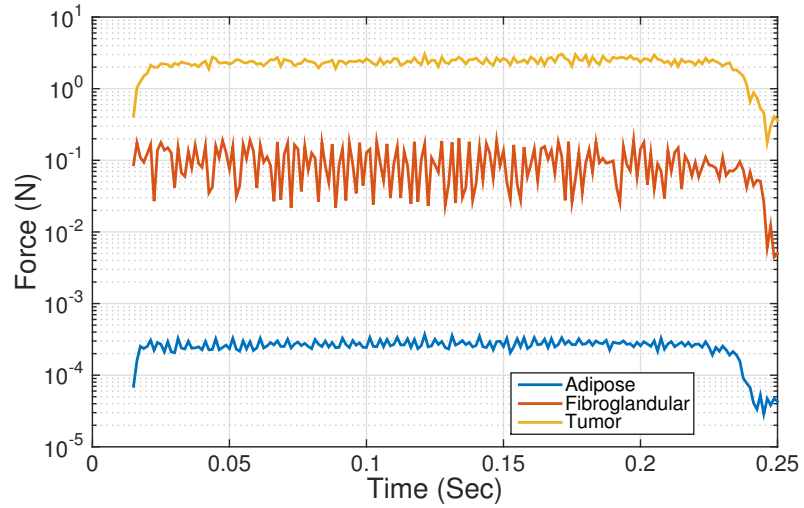
(c) Slice-push ratio = 10

Figure 4.19: Cutting in different rotation speeds (high translational speed)

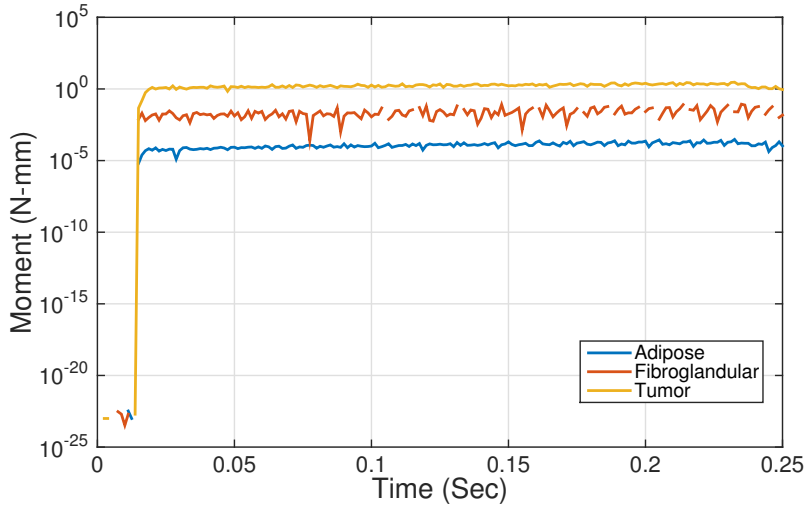
4.3.5 Sampling Different Types of Breast Tissue

This section investigates the cutting of the adipose, fibroglandular and tumor tissue (see Table 4.1). The high translational cutting speed ($v_a = 100mm/s$) is chosen because it has the shortest computation time. This only requires simulating a total time of 0.25 seconds for the cutter to travel through the stroke S (25 mm). If the medium translational speed ($v_a = 50\ mm/s$) or the low translational speed ($v_a = 10\ mm/s$), this time would increase to 0.5 or 2.5 seconds for the cutter to complete the movement and simulation time would be multiplied. A medium slice-push ratio = 1 is used so that the cutter rotates with a medium speed (47.73 rad/s). The goal is to investigate cutting tissue with different levels of stiffness.

The mean reaction force of the tumor tissue is 23.62 times the mean reaction force of the fibroglandular tissue. This number is close to the ratio of their stiffness (2162/105 = 20.59). This result indicates that the nonlinear behavior does not have much influence here (recall that a linear elastic model is used for the tumor tissue and a Neo Hookean model is used for the fibroglandular tissue). The mean reaction force of the tumor tissue is 10639 times the mean reaction force of the adipose tissue (recall that a Neo Hookean model is also used for the adipose tissue). The number obtained in this comparison is much less than the ratio of their stiffness (2162/0.08 = 27025). The result of this comparison (tumor vs. adipose) is very different from the previous one (tumor vs. fibroglandular). In the former case, the stiffness is the main variable that controls the magnitude of the tissue reaction force. The stiffness appears not to be the only factor that affects the tissue reaction force in the latter case due to the fact that the adipose tissue is assigned higher maximum strain resistance. The relationships are found different in the comparisons of the tissue reaction torque. The reaction torque of the tumor tissue is 72.68 times that of the fibroglandular tissue and 16208 times that of the adipose tissue. It is unclear which of the stiffness, nonlinear behavior and maximum strain resistance has more impact to these differences. The semi-logarithmic plots of the tissue reaction force and torque are shown in Fig. 4.20.



(a) Tissue reaction force



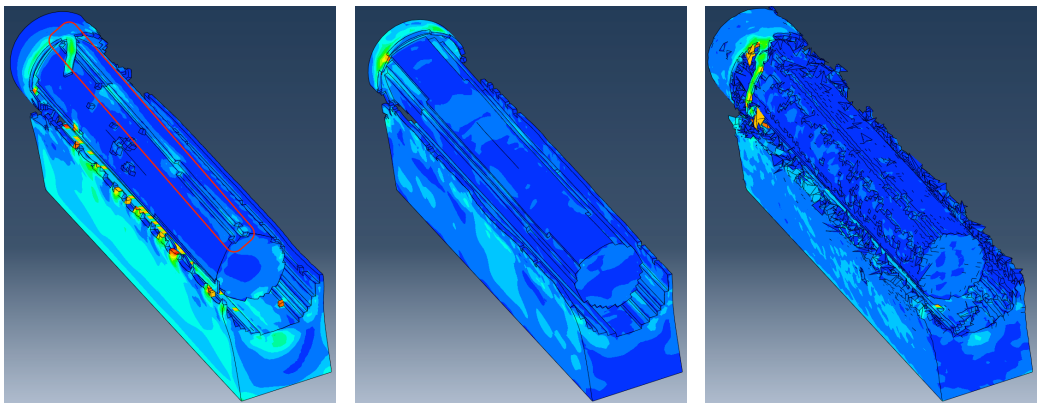
(b) Tissue reaction torque

Figure 4.20: Comparing tissue reaction force and torque when cutting different types of tissue

Figure 4.21 shows the shapes of different types of tissue samples being removed at the end of the cutting process. The von Mises stress field is visualized on the tissue surfaces, where a standard rainbow spectrum is used to represent from high to low stress values with colors from red, green to dark blue. Judging from smoothness of these tissue

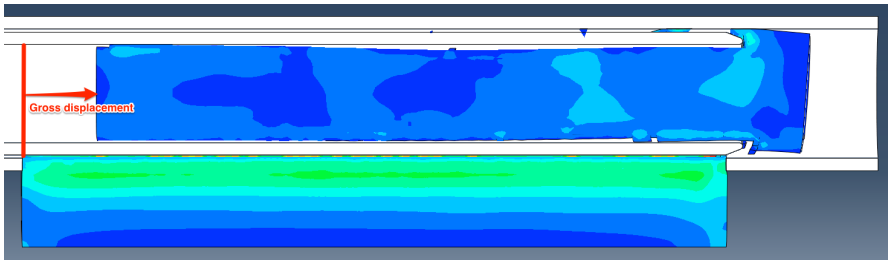
surfaces, the fibroglandular tissue sample as shown in Fig. 4.21b has the best quality. First of all, the outer surface of the fibroglandular tissue sample is smoother than the other two tissue samples. The adipose tissue sample (Fig. 4.21a) is also clean, but not in the region of the red rectangle, which is on the opposite side of the aperture. We can clearly see that the outer surface of the tumor tissue sample (Fig. 4.21c) is very rough, which has much tissue damage caused by the drag and push of the cutter.

Observing from the cut-plane view (see Fig. 4.17 for the definition), it is clear that the fibroglandular tissue sample is the least pushed of all three cases (see Fig. 4.22). This can be confirmed by looking at the displacement of both the distal and proximal surface of the tissue sample. Based on that the original tissue position is defined the fixed edges of the aperture, Fig. 4.22b clearly shows the smallest gross displacement, while the tumor tissue sample is largely pushed towards the needle tip. From both the 3D view and the cut-plane view, it is still difficult to clarify whether the tumor tissue sample is fully removed or not.



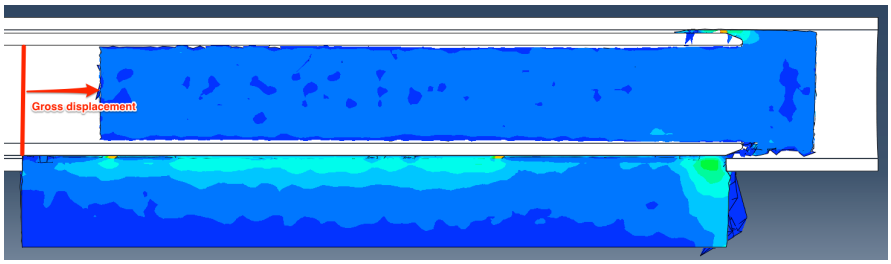
(a) Adipose tissue sample (b) Fibroglandular tissue sample (c) Tumor tissue sample

Figure 4.21: Visualization of the tissue samples, where the cutter and the outer cannula are suppressed



(a) Adipose tissue sample

(b) Fibroglandular tissue sample



(c) Tumor tissue sample

Figure 4.22: Visualization of the tissue samples at the cut plane

4.4 Conclusion

A tissue-cutter interaction model for predicting tissue reaction force and torque was developed. We confirmed that the solution converged when the element size is smaller than or equal to 0.20 mm. The total energy of the system conserved when the time increment was smaller than or equal to 5×10^{-6} seconds. This simulation configuration provided a good balance between the computing efficiency and the solution quality for conducting the parametric studies. If the goal were to obtain optimal quality of the solution, the mass-scaling threshold should be set smaller or removed and the mesh

should be further refined to have an element size of 0.12 mm or smaller. We found that these two strategies could reduce the noise appearing in the tissue reaction force and torque, but the downside is a significant increase in the simulation time.

The effects of the slice-push ratio under different translational cutting speeds were investigated by comparing the trend lines and mean values of the tissue reaction force and torque. The results showed that increasing the slice-push ratio reduced the tissue reaction force in the axial direction of the cutter. This tendency generally agreed with a previous study [34], but how the tissue reaction force reduces with the increasing slice-push ratio was different. In our study, the reduction of the tissue reaction force was much more significant when the slice-push ratio increased from 1 to 10, but was less when the slice-push ratio increased from 0 to 1. In the previous study, however, the reduction of the tissue reaction force was significant when the slice-push ratio was between 0 and 1, but became trivial when the slice-push ratio was larger than 2. This conflict may be caused by two factors: (1) In the previous study the tissue was simply cut by a hollow cylinder, while we used a realistic VAB needle structure, which included an aperture mechanism for capturing the tissue and constraining its movement; (2) the translational cutting speed for VAB used here was much higher ($[100, 50, 10]$ mm/s vs. 2 mm/s) than what was used in the previous study. In addition to the tissue reaction force and torque, we evaluated the tissue sample quality by looking at the visualization of the deformed tissue shape. The results showed that the tissue sample quality gradually improved with the increase of the slice-push ratio.

The same strategies of investigation were used to compare cutting different types of breast tissue. The toughest tissue (i.e. the tumor tissue) resulted in the most extreme tissue reaction force and torque, while the softest tissue (i.e. the adipose tissue) had the smallest tissue reaction force and torque. Looking further into each of the tissue reaction force and torque, there are a few factors with different levels of influence. The tissue reaction force was affected the most by stiffness, while the maximum strain resistance was also influential. However, in the comparisons of the tissue reaction torque, all of the three factors, tissue stiffness, nonlinear behavior and the maximum strain resistance, can be influential and it is unclear which one plays a more important role. Based on the conditions used in the simulations, the fibroglandular tissue sample had the most clean-cut profile. The adipose tissue sample also had a smooth outer surface, but was

significantly displaced at the end of cutting process. Comparing to the adipose tissue sample, the tumor tissue sample had a very rough outer surface and was displaced about equally. The larger displacement means more likely the tissue was torn apart and the fracture would eventually propagate away from the main cutting path. This type of cutting tends to create irregular sample shape and result in a smaller volume of tissue sample. Furthermore, when the tissue displacement becomes too large, the tissue sample can eventually touch the distal end of the needle and cause unexpected results, such as a dry tap.

The greatest uncertainty in this VAB tissue cutting model relates to the tissue properties. The breast tissue deformation models used here are mainly designed for predicting the overall breast deformation under gravitational load or pre-compression, not for rapid deformation in a cutting process. In addition, assumptions were made for the behavior of the tissue fracture, as it is currently unavailable. Maximum tensile and shear strain resistances were specified and two general characteristics were given: (1) soft tissue tends to fracture more easily by shear than tension and (2) softer tissue can resist higher strains before fracture occurs and vice versa. The numbers of these maximum strains were set based on information collected from general soft/tough tissue types, such as tendon, ligament and blood vessels. The assumed tissue fracture behavior would not represent realistic cutting conditions, but was a currently available approach to conservatively predict tissue reaction force and torque for the engineering design of VAB devices. To develop more realistic tissue fracture properties, more tissue characteristics (e.g. viscoelasticity, anisotropy and strain rate) should be considered.

The boundary conditions of the tissue can also be improved. The fixed condition that simplified the presence of suction can be replaced by an accurate pressure field applied by the vacuum. The pressure data can be obtained from the VAB device specifications or direct measurements. Using a local breast tissue portion and fully constraining its surface connecting to the global breast is a necessary simplification. If the condition at the local-global breast interface was of concern, the model would include the entire breast and simulate a needle insertion process and then the cutting process. However, this would require a much more modeling efforts and longer simulation time.

To our knowledge, the proposed model is the first to realistically simulate the VAB cutting. We integrated state-of-the-art breast tissue models and a range different of

cutting speeds into this model. The results have the potential to provide important insight for designing and improving VAB devices.

Chapter 5

Design and Analysis of a VAB Device Using the Simulation-based Framework

5.1 Overview

In this chapter, the VAB cutting model developed in Chapter 4 is introduced to the implemented simulation-based framework, which is discussed in Chapter 3. Comparing to the previous example of the bending biopsy needle, this VAB tissue cutting example includes more realistic design problems and the simulation of the tissue cutting process is with higher complexity and requires much more computation time. In addition, the design parameters involved in this problem are across multiple components of the VAB device system. For example, the predicted tissue reaction force/torque affects the selections of the power source(s), as well as the handpiece design ergonomics. Design trade-offs between the parameters of multiple system components have to be considered when making design decisions. The simulation-based framework enables an integrated, human-in-the-loop design approach in the DBD environment to explore the design space. This new design approach based on extensive simulations is different from what current design optimization packages are offering. The use cases demonstrated in this chapter show the potential of this approach to provide insights/solutions for problems that are

difficult to identify/solve using the traditional ways.

5.2 The Tissue Cutting Problem

Here we briefly review the tissue cutting problem. The tissue cutting process, as illustrated in Fig. 5.1, starts after the suspicious lesion tissue is drawn into the chamber of the inserted biopsy needle. The inner cutter removes a tissue sample with translation and rotation motion driven by a linear and a rotary motors. During this process, the tissue is deformed and progressively damaged by the cutter motion, while the cutter is loaded by the tissue reaction force and toque. In a successful tissue sampling sequence, a tissue sample should be retrieved when the cutter reaches its final position.

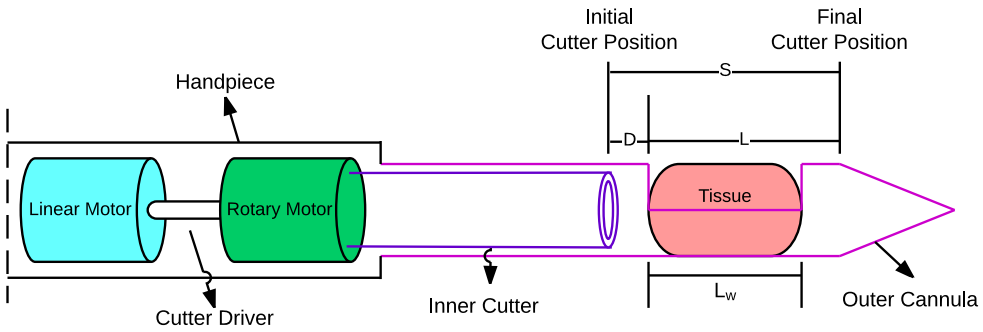


Figure 5.1: The tissue cutting problem in the breast biopsy procedure (repeated from Fig. 4.2).

To define a reasonable scale for our design optimization study, a few parameters are pre-selected. First, the needle geometry is not a variable, which means the effects of different needle sizes and cutter designs are not of interest in this study. Secondly, the performance of the linear motor is not evaluated. We assume that the linear motor is capable of providing required translational cutting speed and force to remove the tissue. In addition, the vacuum is also excluded. The load condition caused by the vacuum suction is simplified by a fixed boundary condition, as discussed in Section 4.2.3. After

this predefined condition, we are specifically interested in the following questions for the VAB device design:

1. How well is the tissue is being cut under different cutting speed configurations, i.e. the translational and rotational cutting speeds?
2. How do different types of breast tissue behave and react to the cutter during the cutting process?
3. How does tissue reaction force and torque affect the performance of the selected motors?
4. Is the VAB handpiece ergonomic, i.e. having a proper weight for single-handed operation?
5. Does the handpiece have a reasonable component cost?
6. What are the tissue sampling rate[1] (tissue volume retrieved per second) and the estimated procedure time?

These questions can be separated into three subsystems: the tissue-cutter system (Q1 and Q2), the motor system (Q3), and the overall device system (Q4-Q6).

5.3 Design Objectives and Critical Parameters

When designing a VAB device, one has to find balance between the device performance, tissue sample quality, device ergonomics and cost. For example, increasing the cutting speeds will improve the biopsy efficiency to remove tissue sample faster, however, this would likely require heavier and/or more expensive motors leading to poorer ergonomics. On the contrary, if we sacrifice the biopsy efficiency so that a lighter and/or cheaper motor can be used, the reduced cutting speeds can result in a longer procedure that the patient has to tolerate or a poorer capability of removing tissue samples. As a first step in the VAB device design, we establish the following design objectives:

- To increase tissue sampling rate.

[1] Tissue sampling rate is a function of the motor speed, the diameter of the inner cutter, the opening window length and width and the tissue properties.

- To be able to retrieve sufficient total tissue volume (the sum of 5 samples).
- To reduce operation time.
- To reduce the weight of the handpiece.
- To reduce the cost of the system.

Four design variables are considered for the design optimization, as seen in Table 5.1. The rotational cutting speed is not directly controlled, instead it is determined by the combination of the slice-push ratio R and the translational cutting speed v_a. This is because the slice-push ratio has been found to be one of the key parameters that affects the variation of the tissue reaction force and torque, as previously reviewed [31, 32, 34]. The tissue type B specifies which type of breast tissue to be used in the cutting simulation. The available tissue types are adipose, fibroglandular and tumor tissue. The rotary motor choice M_r specifies one out of five pre-selected motor models, while the linear motor is assumed to always provide desired translational cutting speed and overcome the tissue reaction force in the axial direction. The geometric parameters of the outer cannula and the inner cutter are not design variables here. The same geometric configuration of the VAB needle shown in Chapter 4 is used here, which is a 8G-RW inner cutter contained by a 6G-RW outer cannula.

Table 5.1: Design variables for the VAB design optimization

Design Variables	Description
v_a	Translational cutting speed (along the long axis of the needle)
R	Slice-push ratio, which is the ratio of translational cutting to the rotary cutting speed
B	Which type of the breast tissue
M_r	Rotary motor choice, which specifies a selected motor from a pre-approved motor list.

The established design objectives identified five performance metrics to be improved/optimized, which are listed in Table 5.2. A few points should be noted in

this table. First, the total procedure time t_p is defined as the total time of five cycles of the cutter to move between the initial position and the final position, plus four rest intervals between each trip. Secondly, the tissue sampling rate V_s defines how much tissue volume can be retrieved per second for an entire procedure. Thirdly, the motor overload factor K varies along with the changing tissue reaction force/torque during a cutting process. A set of K values of the rotary motor selection is computed using the predicted tissue reaction torque at each simulation time step and K_{max} is the maximum value in the set. Again, the performance of the linear motor is not considered here. Finally, in order to evaluate the handpiece ergonomics and cost, the mechanical system including the motors and the needle is considered.

Table 5.2: Performance metrics for the VAB design optimization

Performance Metrics	Description
t_p	Total procedure time required for retrieving 5 samples
V_s	Tissue sampling rate (tissue volume retrieved per second)
K_{max}	Maximum motor overload factor for the rotary motor
w_m	The mechanical system weight including the motors and the coaxial needle system
c_m	Price required to build the mechanical system

5.4 The Solving Process for the Performance Metrics

To obtain the performance metrics listed in Table 5.2, we need to solve for three systems: tissue-cutter system, motor system, and device system. An overview of the solving process is shown in Figure 5.2. With prescribed cutting parameters as the initial input, the three systems are solved sequentially. The tissue-cutter system is first solved to predict tissue reaction force and torque, which are input to the motor system. Next, the rotary motor is evaluated to compute motor's performance attributes . Finally, an overall device evaluation is performed. All the information from the first two systems are used to calculate device performance indexes, such as device weight and required total procedure time. The solutions of each system also answer the questions asked in

Section 5.2.

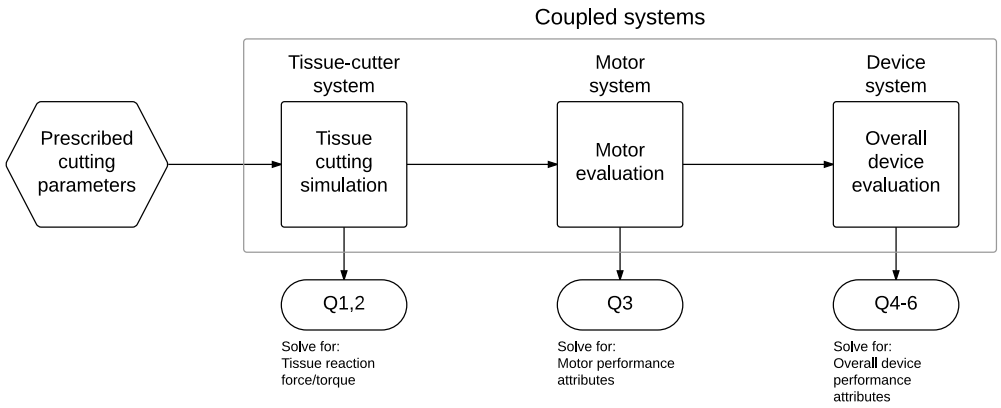


Figure 5.2: Process of solving the coupled systems for the VAB design

Each of the three systems are further explained in the following sub-sections.

5.4.1 The Tissue Cutting Simulation

The tissue cutting process is simulated using the VAB cutting model discussed in Chapter 4. The simulation requires three input variables: v_a, R, and B, which are directly from Table 5.1. The process of the cutter traversing through the tissue is simulated and three types of intermediate result data are output: tissue reaction torque on the cutter $T(t)$ and the displacement field $\underline{U}(t)$ and the von Mises stress distribution $\underline{\sigma}(t)$ over the entire tissue region. $T(t)$ is one of the input parameters to the motor system and the other two are spatially distributed fields which are used to provide the 3D visualization of the deformed tissue body and stress contour displayed on it. These three types of output data are temporal data sets computed at each of the simulation time steps. The input and output of the tissue-cutter simulation are summarized in Fig. 5.3.

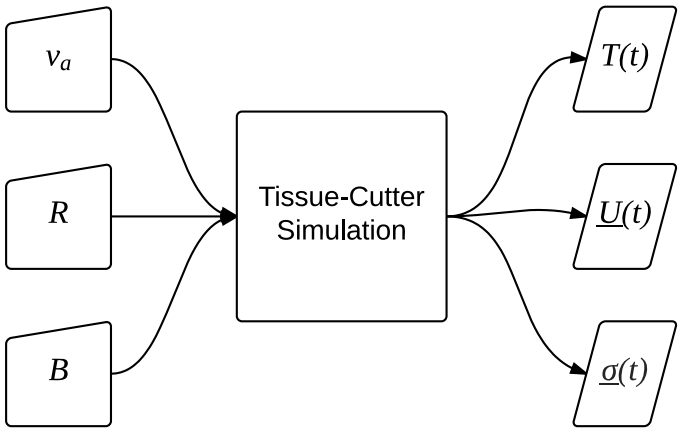


Figure 5.3: Input and output of the tissue-cutter simulation

5.4.2 The Motor Evaluation

The motor evaluation module computes the rotary motor performance attributes when its driven component, the inner cutter, is under a varying load caused by the tissue reaction torque. The cutting process is a short-term operation so it is allowed to briefly overload the motor. Using predicted tissue reaction torque from the tissue cutting simulation, the motor's overload factor is computed. Then, the maximum motor on-time and minimum motor off-time are suggested, which ensure over-heating will not occur.

A series of motor equations are used to evaluate the performance of the selected motor. These equations are based on the motor selection guides [45] of Maxon Motor®. Two input variables are required for this evaluation, one is the $T(t)$ predicted by the tissue simulation and the other is M_r from Table 5.1. M_r is an index number that specifies a particular rotary motor model with its own characteristics, weight and cost. A list of the motor characteristics used in the equations is provided in Table 5.3.

Table 5.3: Motor characteristics used in the evaluation of overheating.

Motor Characteristics	
m	Torque constant
I_N	Rated Current
R_{th1}	Thermal resistance housing-ambient
R_{th2}	Thermal resistance winding-housing
τ_ω	Thermal time constant

Given a required translational cutting speed (v_a), we first calculate the time for a single sampling sequence t_s, where the cutter travels from the initial position to the final positions and back to the initial position. Referring back to Fig. 5.1, D, L and S are the distance between the initial cutter position and the contact position (i.e. the position that the cutter begins to contact the tissue), the distance between the contact position and the final cutter position and the distance between the initial cutter position and final cutter position, respectively. The cutter has to travel through these distances with acceleration, constant speed and deceleration. Figure 5.4 illustrates the profile of the translational cutting speed. From the initial position, the cutter is accelerated to v_a within D and creates a first contact with the tissue. Then, the cutter traverses through L to the final position while remaining the translational cutting speed at v_a. Finally, the cutter moves backward by S to return to the initial position with the translational cutting speed being decelerated to zero. This completes a single sampling sequence and the travel time t_s can be calculated by the equation below:

$$t_s = \frac{2D}{v_a} + \frac{L}{v_a} + \frac{2S}{v_a} \tag{5.1}$$

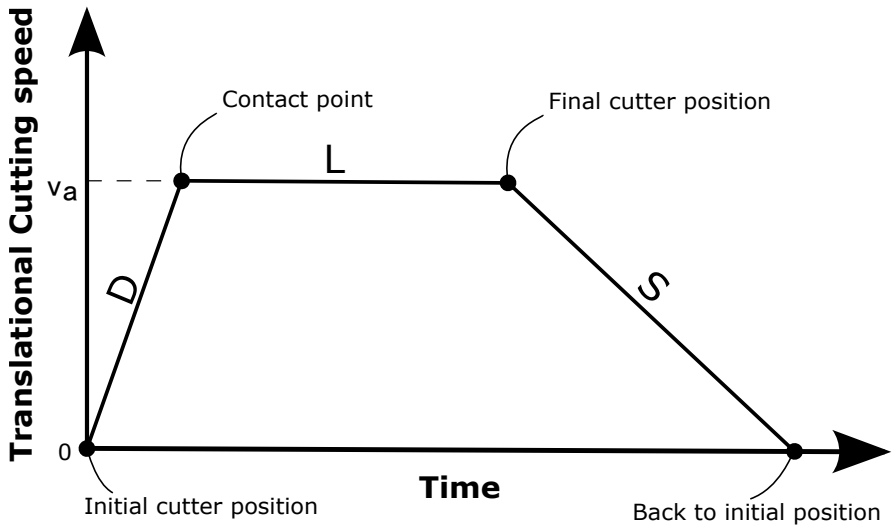


Figure 5.4: Translational cutting speed profile

The rotational cutting speed has a similar profile as the one shown in Fig. 5.4. Here we only consider the constant speed period during which the tissue reaction torque ($T(t)$) is applied to the cutter and results in a motor load. To evaluate for over-heating of DC motors, the motor load is converted to the electrical load $I_{mot}(t)$ by a torque constant m:

$$I_{mot}(t) = \frac{T(t)}{m} \tag{5.2}$$

Using I_{mot}, the rated current I_N, and thermal resistances R_{th1} and R_{th2}, the motor overload factor K can be calculated;

$$K(t) = \frac{I_{mot(t)}}{I_N} \cdot \sqrt{\frac{R_{th1}}{R_{th1} + R_{th2}}} \tag{5.3}$$

Then, the maximum motor on-time $t_{on,max}$ and the minimum motor off-time $t_{off,min}$ are calculated using K_{max} and $I_{mot,max}$, which occur when the T_{max} is applied to the cutter:

$$t_{on,max} = \tau_\omega(\frac{K_{max}^2}{K_{max}^2 - 1}) \tag{5.4}$$

where τ_ω is the time constant of the motor, as listed in Table 5.3.

$$t_{off,min} = (\frac{I_{mot,max}^2}{I_N^2} - 1) \cdot t_s \tag{5.5}$$

The input and output of the motor evaluation is summarized in Fig. 5.5.

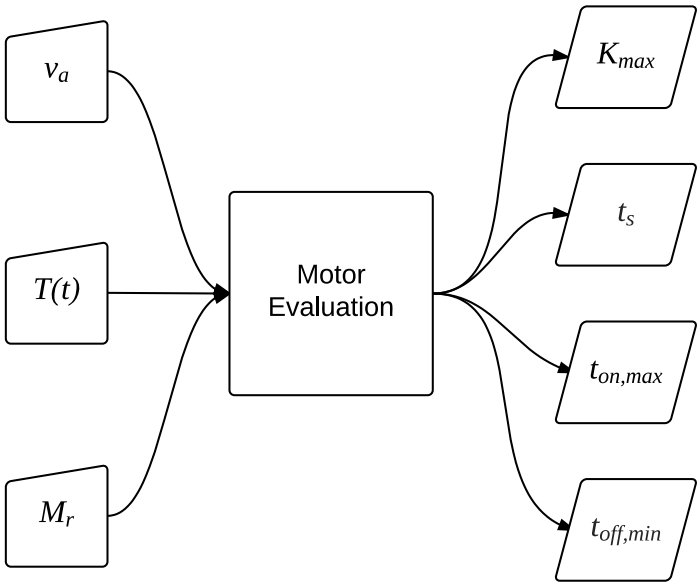


Figure 5.5: Input and output of the motor evaluation.

5.4.3 Overall Device Evaluation

In Section 5.4.2, one of the performance metrics K_{max} is obtained from the motor evaluation using predicted tissue reaction torque. This section computes the other four performance metrics in Table 5.2.

Total procedure time (t_p). t_p is defined as time required for retrieving five tissue samples. The operation time for each sampling sequence (t_s) is the time for the inner cutter to complete a back-and-forth movement between its initial and final positions, which can be computed from Eqn. 5.1. In addition, a rest time between each sampling sequence is recommended to avoid overheating the rotary motor. The minimum rest time can be calculated from Eqn. 5.5. For five sampling sequences with four rest

intervals, t_p is calculated below:

$$t_p = 5t_s + 4t_{off} \tag{5.6}$$

Tissue sampling rate (S_t). In the VAB cutting simulations, we are able to compute the tissue sample volume V_t by summing the volume of the mesh elements that are captured in the chamber of the inner cutter. The tissue sampling rate is defined as tissue volume retrieved per second:

$$S_t = \frac{5V_t}{t_p} \tag{5.7}$$

Mechanical system weight (w_m). w_m is the total weight of the motor system and the coaxial needle system. The motor system is a combination of a linear motor and a rotary motor, while the coaxial needle system includes an outer cannula and an inner cutter. In this VAB design problem, three of the four components are fixed for all design configurations, only the rotary motor is a variable. Therefore, the total weight varies along with only the weight of the rotary motor w_r:

$$w_m = w_r + w_s \tag{5.8}$$

where the constant w_s is the sum of the weights of the linear motor and the coaxial needle system.

Component cost (c_m). Similar to w_m, c_m is the total cost of the four components included in the mechanical system, where only the price of the rotary motor varies. The fixed cost for the other three is defined as c_s.

$$c_m = c_r + c_s \tag{5.9}$$

The input and output of the overall device evaluation is summarized in Fig. 5.6.

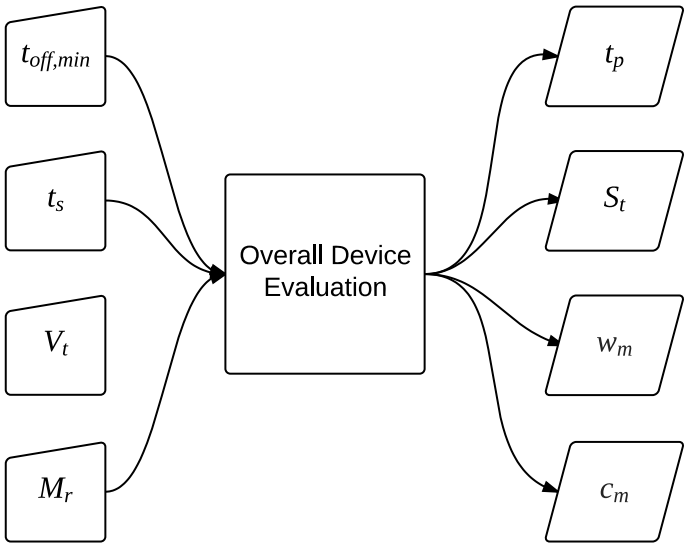


Figure 5.6: Input and output of the overall device evaluation.

5.4.4 Summary of the Solving Process

The complete process of solving for the three coupled system is illustrated in Fig. 5.7, where the evaluation modules, system input and system output variables are marked in different shapes and colors. First, the four design variables (green blocks) shown on the top are specified/selected and input to each of the evaluation modules (blue blocks). The tissue-cutter simulation output four intermediate parameters/data fields (white blocks), where $T(t)$ and V_t are computed for the motor evaluation module and the overall device evaluation module respectively, and where $\underline{U}(t)$ and $\underline{\sigma}(t)$ are used to visualize the cutting simulation in DBD. The motor evaluation module outputs one of the performance metrics (purple blocks), K_{max} and a suggested maximum motor on-time, which has to be confirmed larger than the required time for a single sampling sequence (t_s). Finally, the remaining four performance metrics are calculated in the overall device evaluation module.

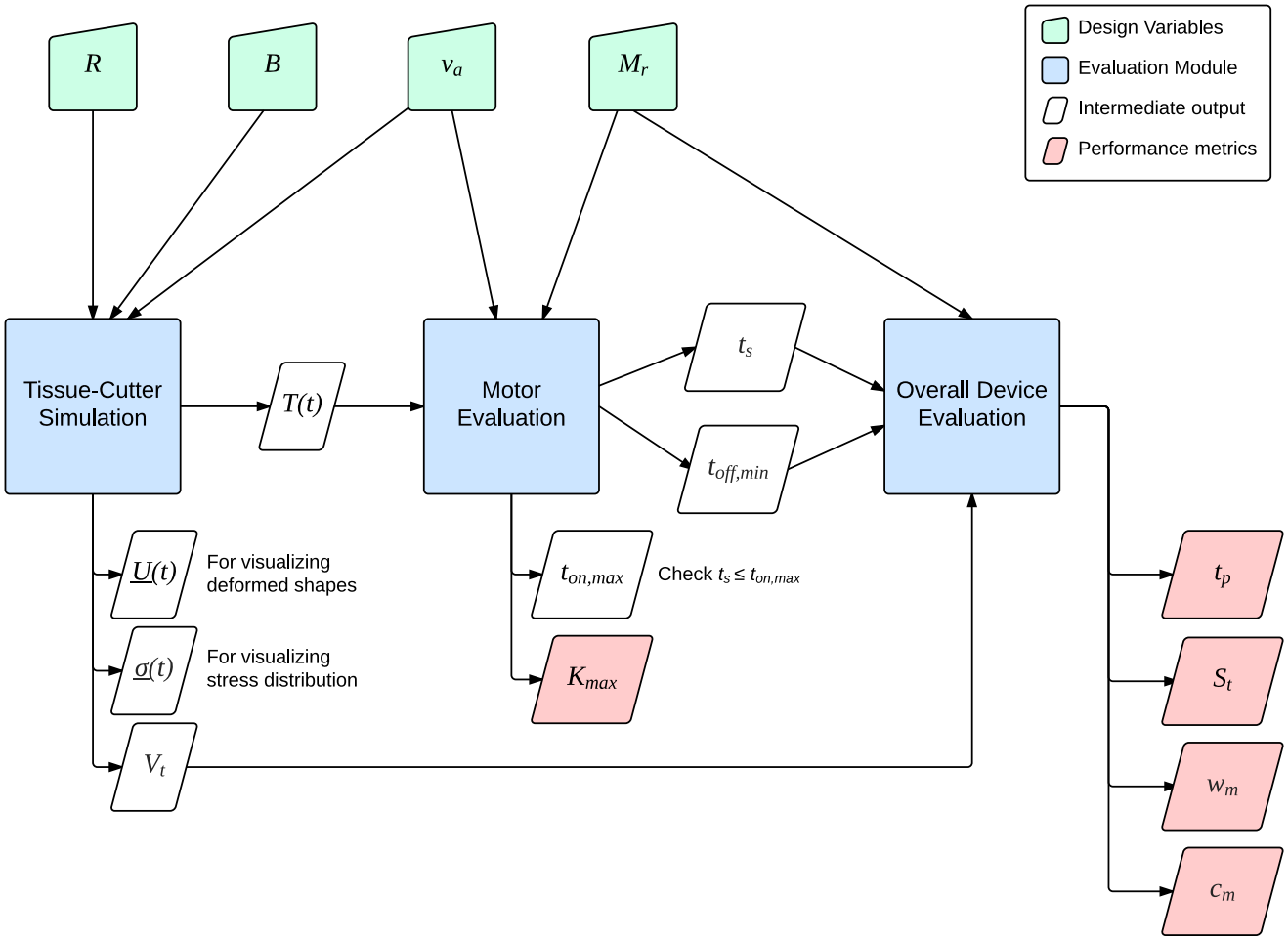


Figure 5.7: Solving three coupled systems from input design variables to output performance metrics.

5.5 The Design Space

5.5.1 Ranges of Design Variable Values

In order to define a design space, maximum and minimum values of the design variables must be specified. Here we discuss how these values are decided.

Translational cutting speed (v_a) and **slice-push ratio (R)**. Slice-push ratio has been recognized as a critical factor that affects the cutting forces [31, 32, 33]. Another experimental study [34] suggested that this effect is significant when $0 \leq R \leq 2$ and becomes minimal when $R > 5$. The specimen of that study was a phantom tissue with a Young's modulus of 20 kPa and the translational cutting speed used was 2 mm/s based on the fact that the common clinical insertion rate is between 0.4 and 10 mm/s [46]. Here we first extend the range of R from $[0 \leq R \leq 5]$ to $[0 \leq R \leq 10]$ as our studies involve stiffer and softer tissue types with nonlinear properties. The translational cutting should be much faster. The rotational speeds found in patents of rotational cutting biopsy needle range from 100 to 5000 RPM [47, 48, 49, 50]. Using this range with the maximum and minimum R, the range of the translational cutting speed is estimated to be 10 to 100 mm/s, which results in a maximum rotational cutting speed of 4558 RPM.

Tissue type (B). Tissue type is a discrete variable, which indicates one of the three available breast tissue types: adipose, fibroglandular and tumor. The tissue properties of each tissue type are described by a constitutive model. Specifically, the constitutive breast tissue models used here are items # 1, 3 and 4 listed in Table 4.1.

Rotary motor choice (M_r). Rotary motor choice is also a discrete variable, which is an index number pointing to one of available of rotary DC motors. These motors are assumed pre-approved models that are ready for being evaluated in the simulations.

5.5.2 Populating the Design Space

The levels of each design variables also need to specified for populating the design space. Table 5.4 describes the levels and number of samples for each design variable. The translational cutting speed v_a is a discrete design variable ranging from 10 to 100 mm/s. An intermediate value is inserted to obtain high (100 mm/s), medium (50 mm/s) and low speed (10 mm/s) settings. The levels of R are non-uniformly distributed

in its defined range. When $0 \leq R \leq 2$, a smaller increment of 0.1 is used as this range has been previously shown to be a rapid change zone for the cutting forces. When R is large than 2, i.e. $2 < R \leq 10$, the increment is increased to 1. With this non-uniform distribution, 20 sample points of R are obtained. The levels of B is decided by the types of those constitutive breast tissue models available in the literature, which are adipose, fibroglandular and tumor tissue. Five rotary motor models are selected from Maxon DC motor products, for which the characteristics of a motor distinct to another so that trade-offs between these motors can be clearly seen. The motor data is provided in Table 5.5.

Table 5.4: Design table for the VAB design problem.

Design Variables	Levels	Number of Samples
v_a	[10 , 50, 100] (mm/s)	$n = 3$
R	[0, 0.1,...1, 2,..., 10]	$n = 20$
B	[adipose, fibroglandular, tumor]	$n = 3$
M_r	[1, 2,..., 5]	$n = 5$
Total samples		900

Table 5.5: Characteristics of the five motor choices.

Motor Index	1	2	3	4	5
Part No.	118536	315174	352923	226748	110066
m (Nmm/A)	0.782	1.15	9.17	10.9	10.7
I_N (A)	0.72	1.66	0.403	0.84	0.253
R_{th1} (K/W)	46	39.8	21.3	15.8	29.8
R_{th2} (K/W)	14	5.1	10.5	4.0	5.5
τ_ω $(Sec.)$	58	1.51	11	15.4	3.53
w_r (g)	17	13	33	159	21
c_r (\$)	102.63	288.88	72.25	181.88	54.38

900 design configurations are generated to sparely populate this four dimensional design space with a total of 10^6 possible solutions, if an increment of 0.1 is used for v_a and R. These design configurations are simulated in a HPC cluster provided by Minnesota Supercomputing Institute (MSI). Each of the simulation jobs is run under 8-core Sandy bridge E5-2670 2.6 GHz processor and 3 simulation jobs are executed simultaneously. The simulation output data is converted into NetCDF format for DBD to compute a design space sampling from the 900 simulation output data sets. Relative distances and smooth transitions (i.e. image warps) between all pairs of design instances are calculated to provide interpolation between the configurations and to enable forward and inverse strategies.

5.6 Summary

We presented a complete workflow from an initial VAB design problem description to a sampling of the design space represented in DBD. A solving process including the tissue cutting simulation, the motor evaluation and the overall device evaluation solves for defined performance metrics from prescribed design variables. Each of the simulation/evaluation modules is a user-developed parametric model. The tissue cutting simulation module is a FEA model, which we discussed in Chapter 4. The motor and overall device evaluation modules are a series of equations which we programmed in Matlab. The design framework integrates all these modules together, provides efficient solutions using HPC resources and outputs the simulation data to DBD for sampling the design space. The entire workflow is illustrated in Fig. 5.8.

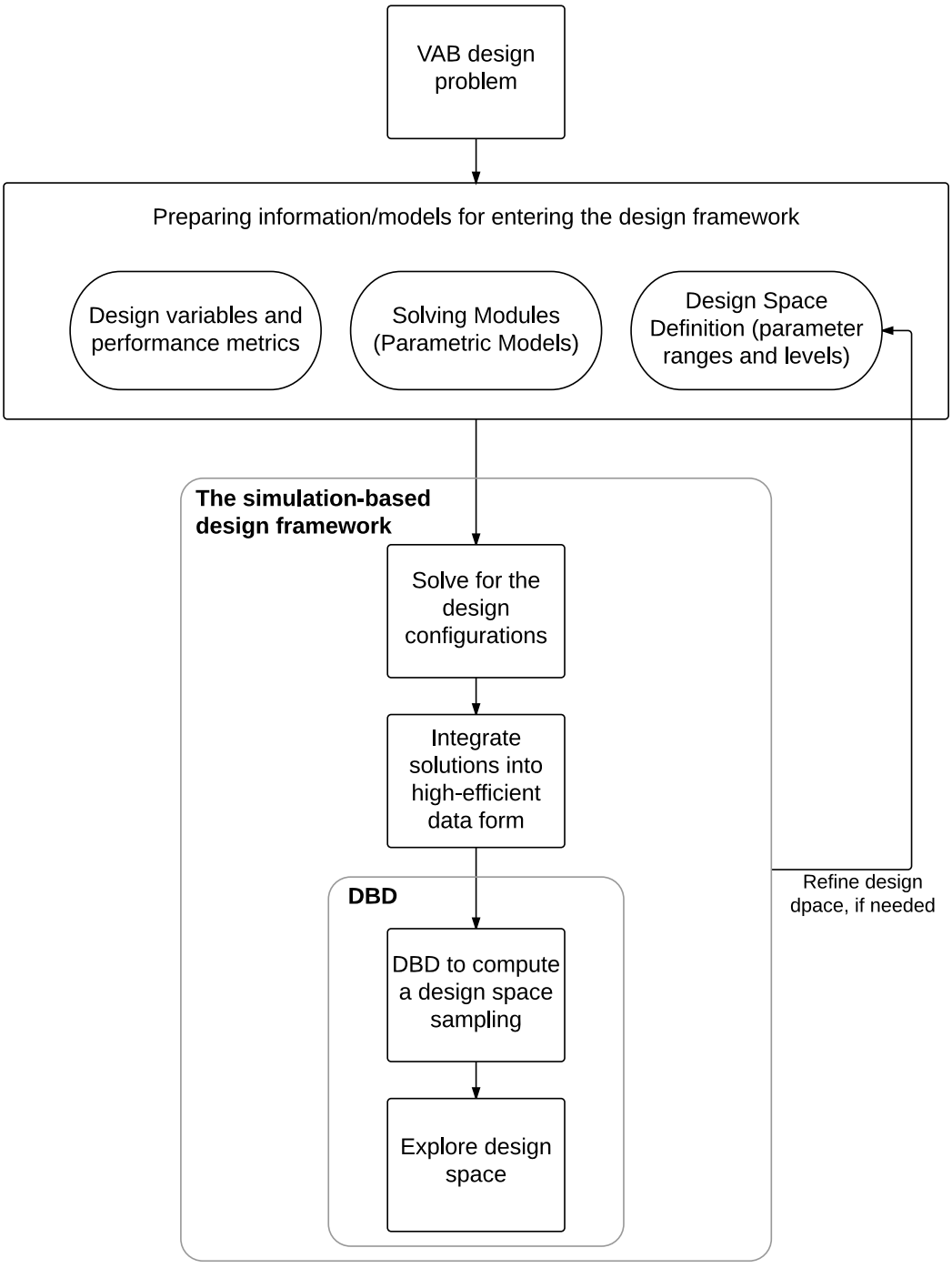


Figure 5.8: The workflow of populating and exploring the VAB design space

5.7 Results

5.7.1 Solution Efficiency

900 simulation jobs were created and executed in the HPC cluster. This process went through two simulation programs in sequence: (1) the tissue-cutter simulation (first module in Fig. 5.7) in Abaqus and (2) the motor and overall device evaluation (second and third modules in Fig. 5.7 were combined) in a custom Matlab program. In step 1, 180 Abaqus simulation jobs were generated using all combinations of the first four design variables in Table 5.4. Next, the result of each of the Abaqus simulations was used in the Matlab program for evaluating the device performance with the five motor choices. The design framework was able to automated the process in most of these steps, including creating and submitting the 180 simulation jobs, converting simulation results into the high-efficiency NetCDF format and running the Matlab program to evaluate the device performance to generate the final 900 datasets.

Due to local constraints (particularly the system memory) on the computer where the DBD software is running on, a coarser mesh ($L_e = 0.3$) was used for the tissue cutting simulations compared to the mesh used in Chapter 4. This larger mesh size resulted in a much shorter simulation time and a processable total data size. Although a higher numerical error was presented in the simulation results, it was acceptable for the purposes of demonstrating the interactive design strategies in DBD. With this mesh size, a single tissue cutting simulation was completed in 2 to 113 minutes. The large differences between the individual cases were mainly due to different translational cutting speeds (v_a). In a simulation, the inner cutter had to travel through stroke (S) with v_a. Therefore, the travel time (t_c) being simulated was:

$$t_c = \frac{v_a}{S} \tag{5.10}$$

With the high translational cutting speed (100 mm/s), it took the inner cutter 0.25 seconds to travel through $S = 25mm$. For the medium and low translational cutting speeds (50 mm/s and 10 mm/s), the travel time increased to double and 10 times longer, respectively. This increase in the travel time would directly extend the simulation time when a same size of the stable time increment is used. The ability of the tissue models to handle large deformation also influenced the simulation time. The Neo-Hookean

hyperelastic models of the adipose and gland tissue provided much better computing efficiency than that of the linear elastic tumor tissue model. The simulations time for cutting tumor tissue was about 4 to 40 times longer than simulating with the other two types of tissue. A summary of the average simulation time in different conditions is shown in Table 5.6.

Table 5.6: Average simulation time (minutes) in cases of different tissue types and translational cutting speeds.

	Adipose	Fibroglandular	Tumor
$v_a = 100\ mm/s$	1.53	1.50	6.3
$v_a = 50\ mm/s$	1.95	1.90	16.85
$v_a = 10\ mm/s$	2.97	4.45	113.00

All 180 tissue cutting simulations were completed in 16.72 hours on the HPC cluster. We were allowed to run each of the simulation jobs under 8 cores of Sandy bridge E5-2670 2.67 GHz processors with 3 simultaneous job executions. The simulation results were stored in Abaqus native file format (ODB files), which were then converted to NetCDF for post-processing in other programs (i.e. the Matlab program and DBD). The conversion time for each ODB file was about 30 minutes. The conversion of the result data does not support parallel computation, but 8 simultaneous job executions were allowed. The total conversion time was about 12 hours. Finally, all of the 180 NetCDF data sets were completed with a total time of 29 hours (simulation time plus conversion time).

In order to evaluate the efficiency of this semi-automatic simulation-conversion process, we conducted some test runs with different cutting speeds and tissue types using only 4 cores of the same Intel Xeon 2.67 GHz processors, which is similar to the performance of a high-end desktop workstation. The average simulation time for a single simulation was about 33 minutes. Estimating from this result, completing all 180 simulations and converting the results one-by-one would require more than 180 hours. A comparison of the efficiency between the semi-automatic process in the design framework and a manual process in a 4-core workstation was shown in Fig. 5.9.

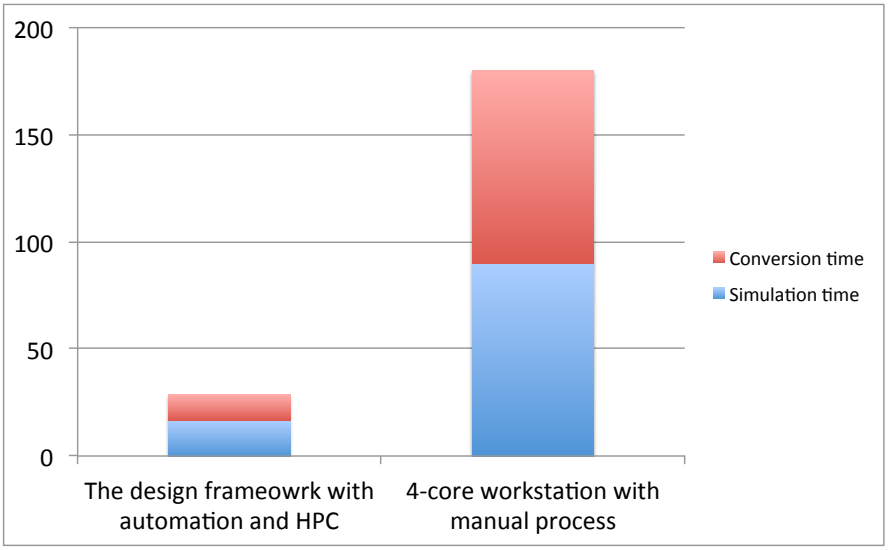


Figure 5.9: Comparison of total time to generate 180 simulation data sets

5.7.2 Representation of the Design Space in DBD

The 900 simulation result data sets were loaded into DBD, each of which was defined as a design instance. To sample a design space from these 900 design instances, the DBD computed the warps, which described smooth transitions between the design instances. Figure 5.10 shows the graphical user interface of DBD with the design space sampling loaded in it. The DBD user interface includes two main rectangle panels where the user interacts with the design space. The smaller panel on the right contains the input radar chart at the top and output radar chart at the bottom. The red polyline in the input radar chart signifies the current design configuration, while the corresponding design outcome is shown by another red polyine in the output radar chart. A tab located at one end of each spoke indicating the weighting status of that design variable/performance attribute. Clicking on the tab will quickly switch the weighting in a sequence of weighted (green, $weight = 1$), pinned (red, $weight = \text{inf}$) and free (grey, $weight = 0$). It is also possible to assign a weighting between 0 and 1 by user input. The bigger panel on the left displays the visualization of the von Mises stress field computed for the current design configuration. Inside the panel, there are a regular view at the bottom and enlarged local view at the top. This particular screenshot is showing the stress contour

in 3D (regular view) and at a 2D cut-plane (enlarged view).

The screenshot shown in Fig. 5.10 is one particular design instance out of 900. To quickly switch between the 900 design instances, one can drag the spokes in the input radar chart and see how the output radar chart responses. Inversely, one can also directly specify design outcome by dragging the spokes in the output radar chart and the internal algorithm will search for a design configuration that best fits what the user implies based on the weighting and where the user drags from and to. The visualization panel is designed not only for viewing but also for direct manipulation strategies. The drag operations can be directly performed on the visualization to manipulate either the geometry features or the stress contour. The system will search through the design space and return the design instance that best matches the manipulation. Recall that the geometry is fixed in this design problem so the focus is placed on the directly manipulation on the von Mises stress field of the tissue.

It should be noted that the number of spokes in radar chart widget is more than the number of design variables/performance metrics which we previously specified. Some of the extra spokes can be used for triggering different display modes, such as switching between the mesh view and von Mises stress contour view and showing/hiding the outer cannula. For example, in Fig. 5.10, we control the outer cannula to be hidden, the mesh of the inner cutter to be displayed and the tissue to be shown as the von Mises stress contour view. The other extra spokes are mainly for providing additional information, such as the motor characteristics.

Also notice that one of the spokes (the only one that has a green tab in Fig. 5.10) in the input radar chart corresponds to the simulation time step. This allows scrolling through time steps of the tissue cutting simulation. If the spoke is pinned at a certain time step, this particular time step will always be shown when switched between design instances. Another way to scroll through time steps and view the animation of a simulation is using the grey bar visible on the bottom of the screen to control the timeline. Figure 5.11 shows 3 time steps out (from left to right) of 21 of an entire tissue cutting simulation, where the mesh of the inner cutter is first shown (top) and is then hidden (bottom) in each time step.

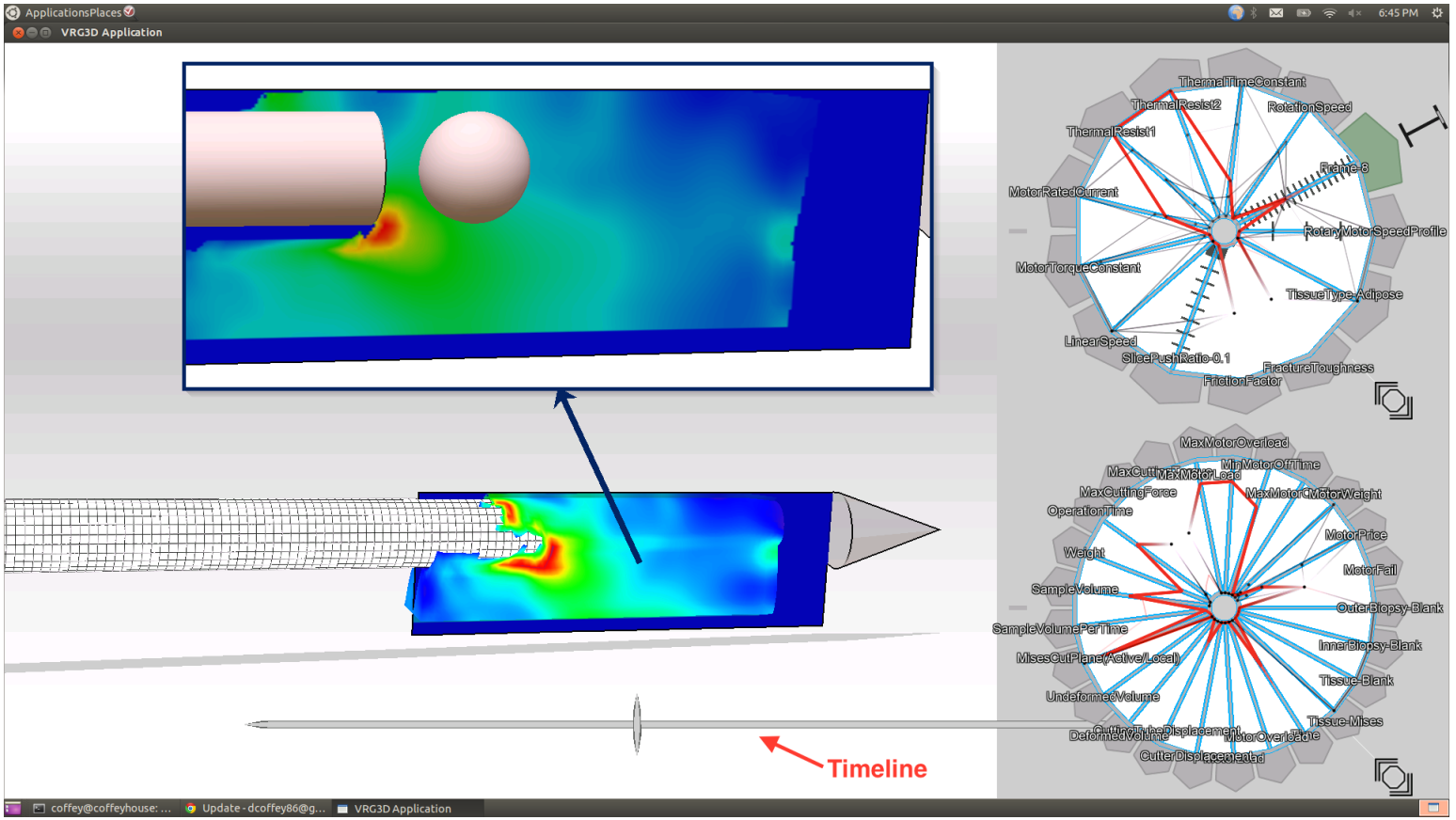


Figure 5.10: The DBD graphical user interface with the design space sampling loaded in it. On the bottom of the visualization panel displays the tissue cutting simulation at time frame 8 out of 21, in which the cutter has penetrated into the tissue by a certain depth. The outer cannula of the biopsy needle is hidden for better viewing the stress field of the tissue, the mesh view is turned on for the inner cutter and the stress contour view is turned on for the tissue. On the top of the visualization panel is an enlarged view of a selected local tissue region displayed at a cut plane. Note that the grey sphere in this view is created for test purposes, which should be ignored.

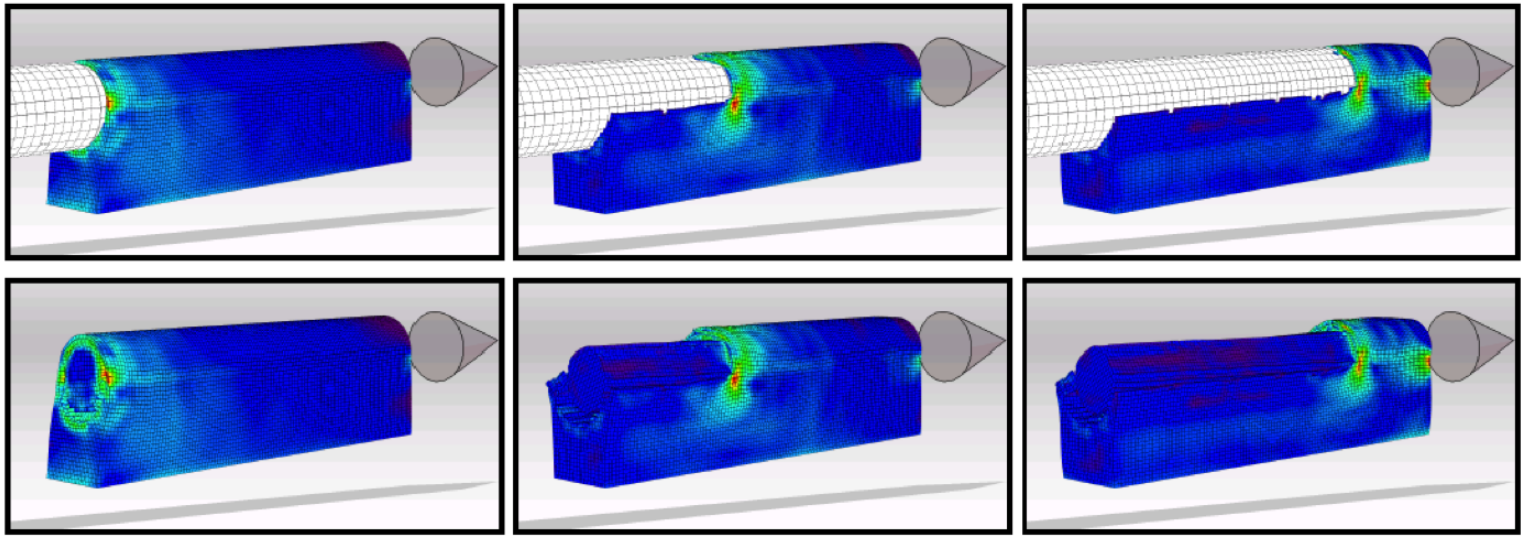


Figure 5.11: Three time steps at early, middle and late stages of an entire tissue cutting simulation. The top row shows the three time steps with the outer cannula hidden and the bottom row shows the same three time steps with the both outer cannula and inner cutter removed.

5.7.3 Forward Design via Radar Chart Widget: Understanding the Effects of Design Input Parameters

The forward design strategy was performed, by which we dragged on the spokes of design input variables to understand their effects on the design outcomes. Here we report the scenario that we interacted with the radar chart widget and the insights gained through the interactions.

Figure 5.12 shows the scenario performed to study the effects of the slice-push ratio on the tissue sample volume and the maximum overload factor of the selected motor. We first set the predefined condition by interacting with the input radar chart. The slice-push ratio was weighted by clicking to turn its tab green as this was the design variable being studied here. The time step is locked to its maximum and the rotary motor choice is locked to # 3. Locking these two variables was done by grabbing their spokes to the specified values and then assign them red tabs. With these settings, the system would return only design instances with the motor # 3 and only display the simulation at the final time step, i.e. the end of the tissue cutting. Note that the motor # 3 was arbitrarily selected for demonstrating the scenario. We will study the performance of different motors in a later forward design task. All parameters in the output radar chart were set to free status because from input radar chart to output radar chart every design configuration maps to a unique set of results. If there is any weighted or locked parameters in the output radar chart, some of the design configurations may be skipped when dragging the input spokes.

Next, we began the main design task. We dragged the slice-push ratio spoke while locking the translational cutting speed and tissue type in place. This was performed for cutting different types of tissue (adipose and fibroglandular) with different translational cutting speeds (high and low). The insights gained are reported below:

1. The relationship between the slice-push ratio and the tissue sample volume was unclear, but lower slice-push ratios tended to contribute to a relatively larger volume.

2. In general, a higher translational cutting speed resulted in a higher tissue sample volume.

3. The motor overload factor increased when the slice-push ratio increased. This

is intuitive as a higher rotational cutting speed would result in a higher tissue reaction torque.

4. The motor was less overloaded when cutting fibroglandular tissue than compared to cutting adipose tissue.

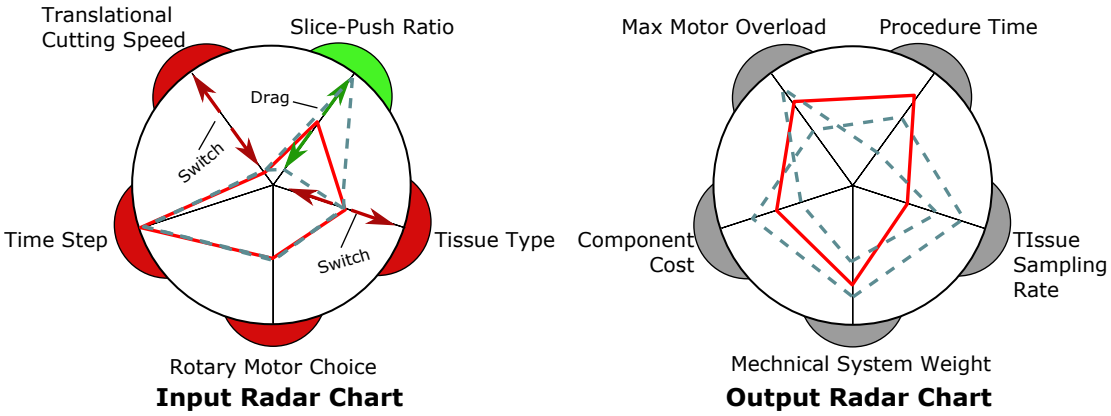


Figure 5.12: A use case of the forward design via radar chart widget. The red tabs lock the design variables so that they do not response to any drag operations. The green tab assigned to the slice-push ratio means it is weighted. This informs the system that the current value of the slice-push ratio is desired. The internal algorithm will suggest design instances with a similar value of the slice-push ratio. As a result, every time one tries to increase/decrease the slice-push ratio, the value will only change by 1 unit instead of varying dramatically. The solid polyline on the input radar chart denotes the current design configuration, while dashed polylines denote previously searched design instances. When dragging on the spoke of the slice-push ratio, the output radar chart responses to reflect the new slice-push ratio. The designer can quickly understand the effects of the slice-push ratio on the performance metrics by looking at how the red polyline of the output radar chart varies.

In the second use case, we focused on understanding the performance of each rotary motor choice with respect to different slice-push ratios when the low translational cutting speed is prescribed. This scenario is shown in Figure 5.13, where the translational cutting speed was first locked to the low value. We switched between different

combinations of the rotary motor choice and tissue type, then we dragged through the range of the slice-push ratio to observe how the performance metrics were affected. The following insights were observed:

1. Using motor 1 or 2 would result in a light-weight, but relatively high-cost handpiece design. This was directly observed from the changes of the component cost and the mechanical system weight when switching between different motor choices.

2. When the slice-push ratio was greater than 4, both motor 1 and 2 were significantly overloaded which could result in overheating problems or or a long rest time after each tissue sampling sequence.

3. The motor overload factor was significantly affected by the slice-push ratio.

4. In general, slice-push ratio has a smaller effect on sampling rate, but this effect is more significant on motor 2.

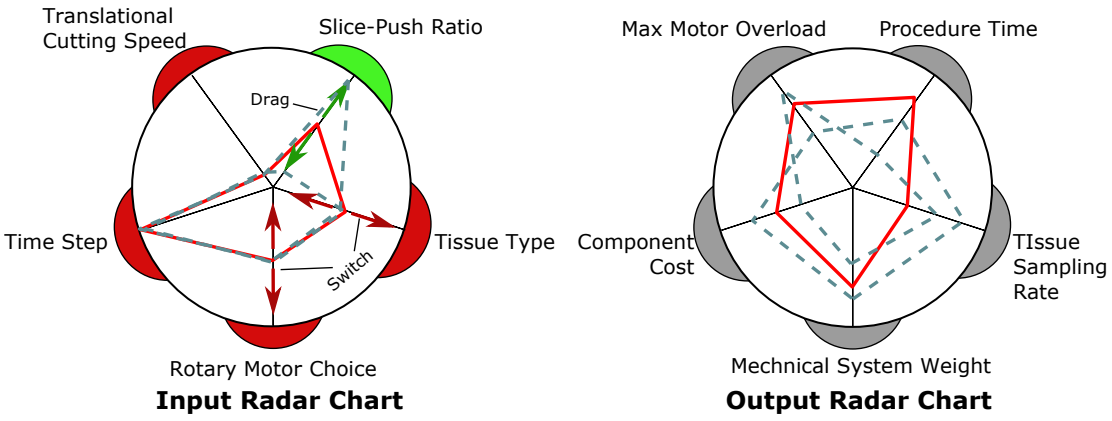


Figure 5.13: Use case 2 of the forward design via radar chart widget.

We repeated the same interaction performed in use case 2, but with the translational cutting speed set to high, as seen in Fig. 5.14. The following insights were gained:

1. When the slice-push ratio was greater than 4, both motor 1 and 2 were significantly overloaded, which could result in overheating problems or a long rest time after

each tissue sampling sequence (the same finding as insight 2 gained in the use case 2).

2. The motor overload factor was significantly affected by the slice-push ratio (the same finding as insight 3 gained in the use case 2).

3. The high translational cutting speed resulted in much higher tissue sample volumes than that in the low translational cutting speed cases.

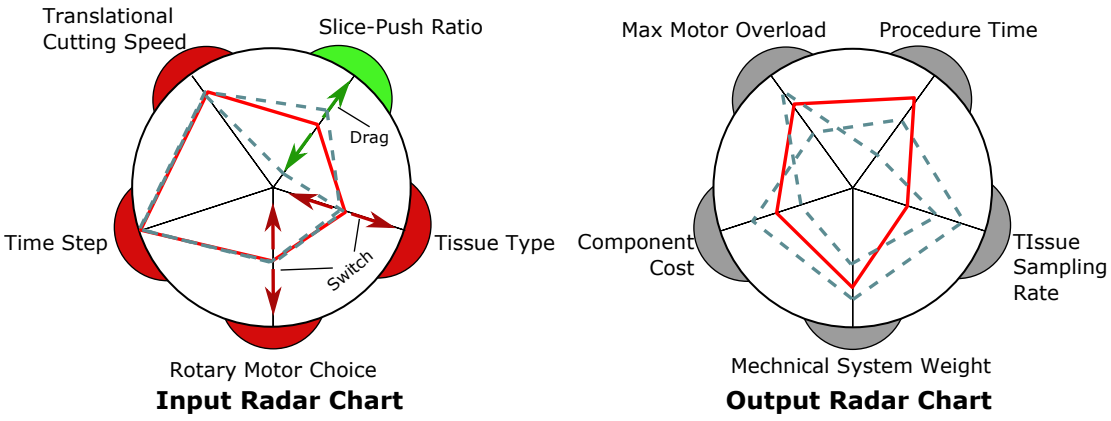


Figure 5.14: Use case 3 of the forward design via radar chart widget.

5.7.4 Inverse Design via Radar Chart Widget: Finding Solutions by Specifying Design Outcome

Based on the insights gained from the forward design interactions, we performed inverse design to find optimal solutions by directly specifying a desired design outcome. In this type of design, the user mainly interacts with the output radar chart. Figure 5.15 shows a use case to balance the trade-offs between component cost, mechanical system weight and tissue sampling rate. The design goals were to maximize the sampling rate for adipose tissue, maintain a proper system weight and minimize the component cost. As a first step, the weighting was assigned to both input and output radar charts. In the input radar chart, the tissue type was locked to adipose and the time step was locked to final (red tabs). This allowed us to look at the end results of those design instances of

sampling the adipose tissue. In the output radar chart, the three performance attributes of interests were weighted (green tabs), while the others were free (grey tabs). Next, a start point for design exploration was located by setting the tissue sampling rate to its maximum, which was the first design objective stated here. From this starting point, we dragged on the mechanical system weight to its mid point. In this operation, the system searched only in the neighborhood of maximum tissue sampling rate in the design space because of the assigned weighting states. Therefore, when we reached the mid value of the mechanical system weight, the design instance we found still had a relatively high tissue sampling rate. Then we began to drag on the spoke of the component cost to reduce its values. While reducing the value of the component cost, we kept observing the input radar chart to see what design configurations were returned. We finally stopped at a design solution located at [linear speed, slice-push ratio, rotary motor choice] = [100, 0.1, 3], which the three performance attributes were balanced based on our judgment. An experienced user of DBD can finish these steps in just a few minutes and obtain several interesting design solutions for further evaluation.

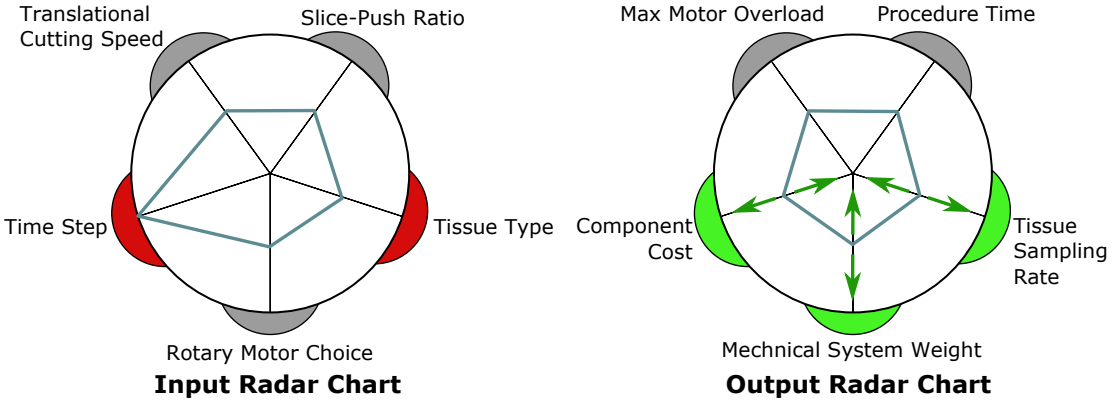


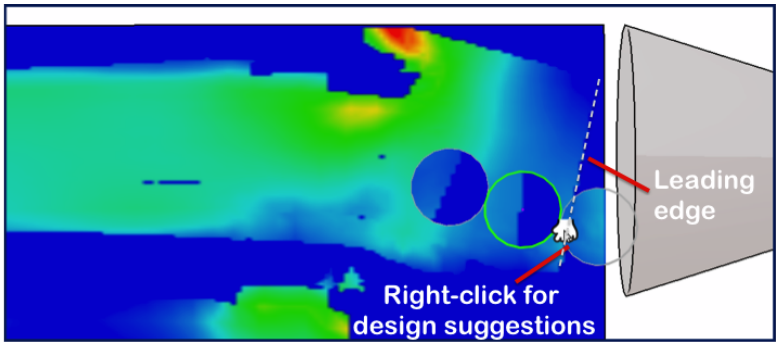
Figure 5.15: A use case of inverse design. Red, green and grey tabs indicate the weighting states of the parameters. In this case, we directly specified the design outcome and obtained optimal design solutions. The green arrows show the drag operations were performed on the three output attributes.

5.7.5 Direct Manipulation on the Visualization

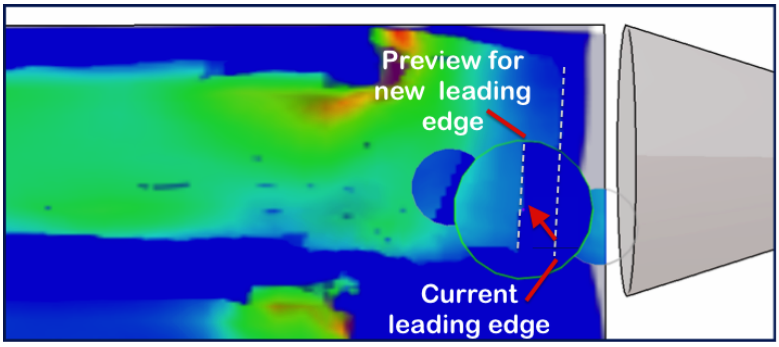
The direct manipulation on the visualization allowed us to directly interact with the contour of a output field, such as stress and strain distributions, to search for better designs. In this section, three use cases of improving cutting performance were demonstrated.

Use Case 1

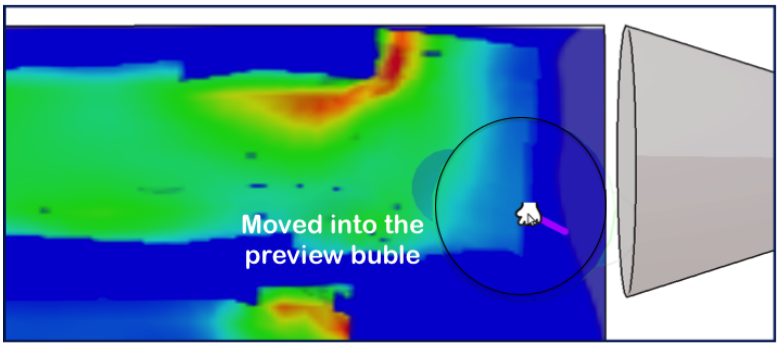
Studies have shown that slicing with pressing is a more efficient way to cut soft materials comparing to the pure pressing [33, 34]. The normal cutting force is largely reduced and the shear damage instead of tensile damage dominates the material fracture. If the normal cutting force is reduced more, the axial displacement of the tissue should be smaller. In order to examine how much the normal cutting force was reduced, we observed the total axial displacement of the tissue at the end of the cutting process. The direct manipulation was performed on the visualization of the deformed tissue shape with the von Mises stress field displayed on it. The stress contour was displayed at a cut plane which is the symmetric plane that cut outer cannula into two equivalent parts. Figure 5.16 shows a step-by-step manipulating process to reduce the axial displacement of the tissue. In step 1 (Fig. 5.16a), we located a design instance in which the tissue was pushed so far that its leading edge contacted the base of the needle tip (note that both the outer cannula and the inner cutter were suppressed from the view). To reduce this displacement, we right-clicked on a location of the leading edge and three preview bubbles were displayed. Each preview bubble was a design instance searched based on the predefined weighting condition set on the radar charts. The preview bubble in the middle clearly showed a design instance with its leading edge significantly moved backward. In step 2 (Fig. 5.16b), we switched to view that design instance by dragging to hit that bubble center. In step 3 (Fig. 5.16c), the new design instance was reached and the axial displacement of the tissue was shown largely reduced. Checking back to the radar charts, we learned that the slice-push ratio was increased from 0.7 to 4.0 while the translational cutting speed stayed unchanged.



(a) Suggested designs with magnified view shown in preview bubbles



(b) Move into a bubble to switch to that design instance



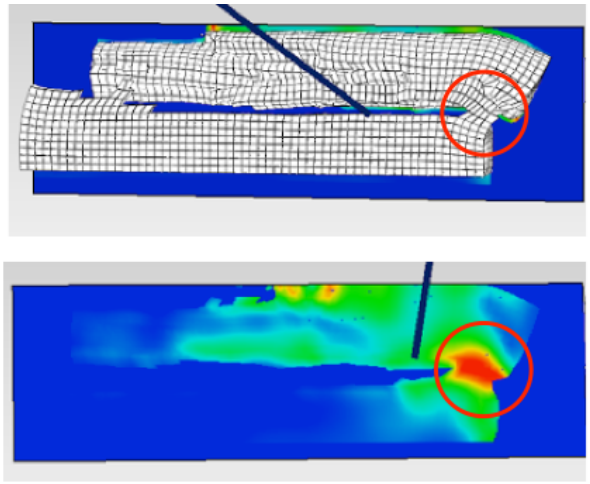
(c) The new design instance is shown when reaching the bubble center

Figure 5.16: Moving a leading edge by inverse manipulation on the visualization

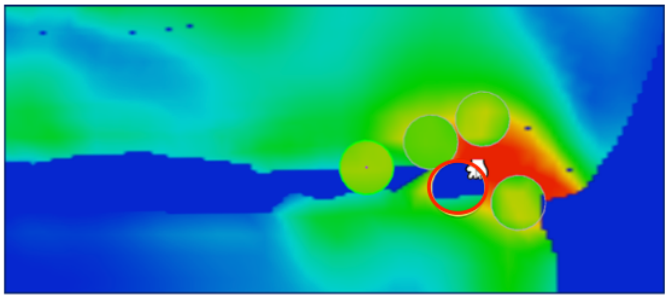
Use Case 2

A "dry tap" is a technical term that indicates a tissue sample is not fully separated from the main tissue. As a consequence, no tissue sample is retrieved when vacuum is applied.

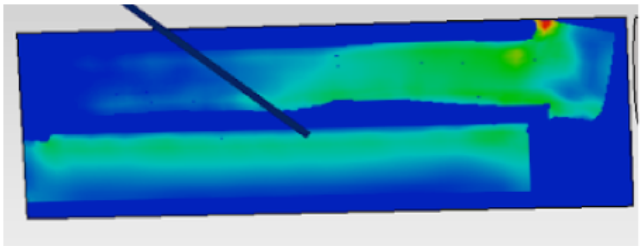
Although this problem was not discussed much in the literature, it was reported during our interviews with interventional radiologists. The dry tap problem was discovered in the tissue cutting simulations when cutting adipose tissue with a low rotational cutting speed. While the inner cutter kept moving forward to rapidly deform the tissue, none of the maximum shear and tensile damage criteria was satisfied so that the tissue fracture never occurred in the simulation. However, the cutting simulation eventually reached a point that the tissue model was not able to handle the large deformation, which caused the cutter's mesh penetrated the tissue. Although this mesh penetration was a numerical error, it was also an indicator of the dry tap because the cutter force was not large enough to break the tissue. Figure 5.17 illustrates how we found a design solution for the dry tap problem. The dry tap occurred at the design instance shown in Fig. 5.17a. The dry tap was identified in both displays of the deformed mesh and the cut plane. The main strategy to solve this problem was to replace the connected tissue (red region) with empty (in blue color). We right-clicked at a location on the red region so that five preview bubbles were shown near the click point, as shown in Fig. 5.17b. These preview bubble views are best design solutions searched by the system based on current weighting status. One of the preview bubbles (circled in red) showed a possible removal of the tissue connection. We then switched to that design instance for a further examination. The visualization clearly showed that the dry tap problem was resolved (see Fig. 5.17c). It should be noted that this was not a unique solution to the issue. More design solutions could be found by requesting more preview bubbles from other right-click points.



(a) Dry tap problem found in a design instance under a low cutting speed



(b) The preview bubble in red indicated a possible design solution



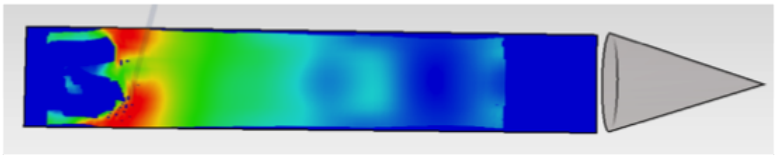
(c) A design solution was discovered

Figure 5.17: Avoiding "dry tap" by direct manipulation via the visualization

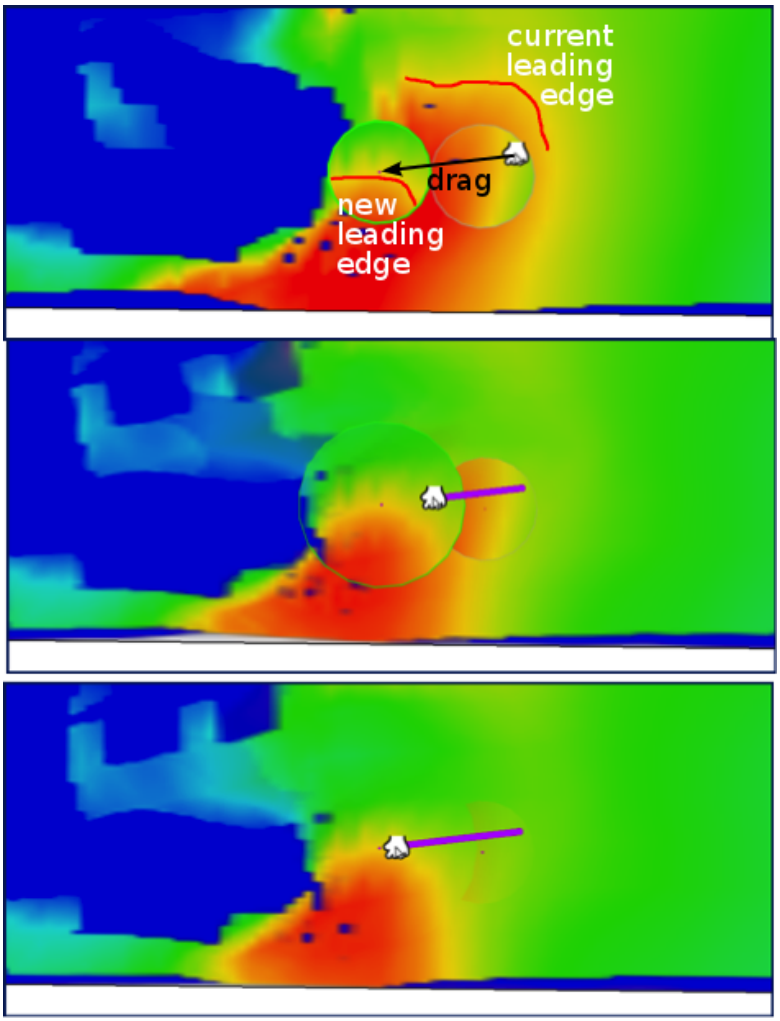
Use Case 3

When evaluating FEA simulation results, there are situations that the designer needs to not only concern nodal maximum/minimum values but also a spatially distributed field in a region. For example, when designing a plate with holes under loads, the designer needs to look at the stress distribution near the hole section to locate and measure the stress concentration. In this type of evaluation, the performance indicator is not identified from numbers, but from visualization (i.e. the location and the size of the stress concentration). Directly manipulating the visualization becomes very useful to this kind of design activities. In this use case, we demonstrate how to shrink a high-stress area via the visualization and how this direct manipulation strategy can help find desired design solutions.

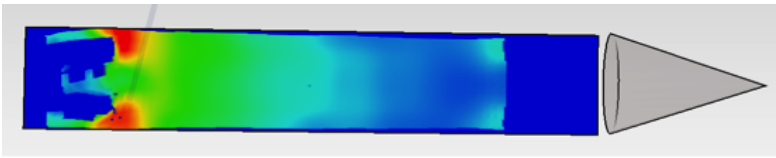
We hypothesized that the tissue cutting with a more concentrated high-stress area surrounding the cutter-tissue contact surface can produce tissue samples with a cleaner profile, which may contribute to more accurate diagnoses. Therefore, a narrower high-stress area near the cutter-tissue contact surface was desired. Here we focused on improving this performance indicator for cutting the adipose tissue. We directly manipulated the stress distribution on the cut plane of the tissue. Figure 5.18a shows the starting design instance: [slice-push ratio, linear cutting speed]=[0.4, 50 mm/s] at an early cutting stage (time step = 7/21). The goal was to shrink the high-stress area (red region). We first right-clicked on a location in the red region to request preview bubbles for design suggestions. As shown on the top of Fig. 5.18b, two preview bubbles were shown near the click point. The bubble on the left indicated that the leading edge of the red region shifted to lower left, which could lead to a narrower high-stress area. Thus, we switched to that design instance and obtained a design solution: [slice-push ratio, linear cutting speed]=[0.6, 10 mm/s] (Fig. 5.18c). This result suggested that by slightly increasing the slice-push ratio, lower translational and rotational cutting speeds could still lead to a better cutting performance. This type of design insights cannot be suggested by design optimization tools as this performance indicator is not quantified. However, using the direct manipulation method, we were able to intuitively and quickly approach a desired stress distribution.



(a) The starting design instance had a wide high-stress area (in red)



(b) Design suggestions were shown in preview bubbles (top). Dragging to move into the new design instance (middle). The view switched to the new design instance (bottom).



(c) The new design instance had a much narrower high-stress area than the original one.

Figure 5.18: Direct manipulation of the stress distribution.

5.8 Conclusion

We presented a more realistic, complex application of device-tissue interaction to the simulation-based design framework with DBD. The benchmark example of the VAB tissue cutting simulation was introduced to the design framework. 900 design instances with different cutting conditions and motor choices were generated in a semi-automatic manner. These design instances sparsely populated a four dimensional design space (10^6 possible solutions) and the DBD sofware was able to interpolate between them. The total time for populating the design space was 28.72 hours compared to more than 180 hours estimated for a state-of-the-art workstation. In addition, the database was stored in an open-source, high-efficiency NetCDF format, which largely reduced the post-processing time for sampling the design space (i.e. computing the warps) and for other visualization purposes. The capability of the design framework to deal with massive simulation data sets was also approved. The size of the native Abaqus output data files (ODB) was 90 GB in total. This size was reduce to half after converting to our custom NetCDF format. Reading data in the NetCDF format was much more efficient than directly reading from a native Abaqus ODB. When reading some large fields, e.g. displacements on 10^6 nodes, it can take up to minutes to return all the nodes, while in the NetCDF format it only takes seconds. The capabilities of compressing data size and providing high-efficiency data access are critical when the design space is significantly large and includes extensive simulations.

The ability of the design framework to integrate with different simulation modules was also demonstrated. In this example, the FEA simulation of the tissue cutting was linked to a Matlab program that evaluated the motor and overall device performance. Each of the 180 tissue cutting simulation results was used in this Matlab program to evaluate with five motor choices so that 900 design instances were finally generated. This scenario frequently occurs in real-world cases where pre- or post-calculations are required when running FEA simulations. Being able to link the simulation modules to each other, batch process through all design configurations, and provide an integrated view in DBD would significantly save the manual effort and potentially benefit the design process with more design insights.

Use cases were provided to show creative forward and inverse design approaches

using the radar chart widget. The forward design via the radar chart widget allowed quickly browsing through design instances by varying one design variable at a time to build understanding on how the performance metrics change with the varying variable. The trade-offs between the design variables were also learned when looking at the changes of the performance metrics. In this type of interaction, we obtained answers to questions similar to "what happens to the tissue sampling rate if I decrease the slice-push ratio?" In the inverse design via the radar chart widget, we were able to directly specify design outcomes to search for design solutions that best fits this specification. Unlike the forward design strategy, the question being asked here is similar to "what are the design configurations that would provide tissue sample volume larger than x and total device weight less than y?" or "If I want to decrease the motor overload factor without decreasing the tissue sampling rate, what would be the design options?"

Using direct manipulation via the visualization, we interacted with the stress contour, dragged to move a local region, and reached desired design solutions. We first improved the cutting performance by dragging to move the leading edge of the tissue sample backward. This reduced the amount of the total displacement of the tissue in the axial direction and provided a preferred solution that the tissue fracture was more dominated by the shear force. Next, we solved a dry tap problem, in which the tissue sample did not fully separate. We dragged to grab the region with the connected tissue toward a new design instance shown in one of the preview bubbles. After switching to the new design instance, the dry tap was confirmed to be resolved with a suggestion to increase the slice-push ratio. Finally, we found a better design solution that had a more concentrated high-stress region. This was achieved by using the same strategy to move the leading edge of the high-stress region inward. All of these direct manipulation use cases showed a process that involve human in the design loop, in which the user knowledge is input during each interaction with the visualization of the simulations. The insight gained through these interactions led the user to step-by-step approach optimal/desired solutions.

Chapter 6

Conclusion and Future Directions

This chapter concludes the dissertation and discusses future improvements and research directions.

6.1 The Simulation-Based Design Framework

6.1.1 Toward Seamless Integration with the Engineering Tools

In this research, we integrated the DBD interface into a workflow, including CAD, FEA and custom Matlab simulations, that are currently utilized by medical device engineers. We also automated this workflow to successfully complete 460 configurations for a bending needle simulation and 900 configurations for a VAB tissue cutting simulation. By outputting the simulation results into the NetCDF format, the entire design spaces were efficiently sampled to enable design manipulations in the DBD interface.

A real-time connection between the DBD software and the Solidworks CAD software was developed to keep a CAD model in sync with the design changes made in the DBD interface. This connection was made possible by SolidWorks API, which we believe is the best way to access a CAD model in the native format without information loss caused by model conversion. However, this real-time synchronization requires a manual pre-visualization step in DBD to create a base model with geometric features mapping to the CAD model. To fully enable a bi-directional synchronization, the research can focus on how to automatically identify key geometric parameters to describe a 3D CAD

model, then the information could be input to DBD to enable automatic creation of the shape of the CAD model and also to allow manipulating the model through the identified parameters. Some existing algorithms ([51, 52]) may provide an excellent starting point for this research direction.

In order to run a large number of FEA simulations, we developed an approach based on an Abaqus model template to generate simulation jobs by controlling identified simulation parameters. The generated simulation jobs are directly executed by an Abaqus kernel so that the interaction with the native Abaqus user environment is not required. We demonstrated two types of Abaqus model templates, which were a Python script (for the bending needle example) and an input file (for the tissue cutting simulation). Using a Python script, we were able to control the geometry of a biopsy needle to simulate 460 different geometric configurations under a perpendicular force applied at the needle tip. However, the communication of this type of model template would become much less manageable when more complicated simulation settings are involved. Basically a Python script is a macro that records a set of user commands required for building a FEA model. Running a Python script still requires an instance of Abaqus interactive session running behind to perform the requested user commands. This type of command executions via a background interactive session is less efficient, especially when this session is created remotely. The commands have to be executed one by one until the FEA model is created. This results in extra time cost in each simulation run. In addition, the Abaqus Python scripting interface does not fully support all simulation settings, for example, the tensile and shear damage criteria used in our VAB tissue cutting simulation cannot be set through a script. Using an input file as a model template is more efficient. This approach skips the process of establishing a FEA model in a backgroud Abaqus interactive session as the input file can be directly interpreted by the Abaqus kernel. The approach allowed us to control the values assigned to the linear and rotary velocities of the cutter and the pre-defined tissue properties to generated tissue cutting simulations under different conditions. The limitation of the input file based approach is that meshes must be generated beforehand. In our tissue cutting simulation, the geometry was fixed so that only an one-time mesh generation was required. However, if any geometric parameter is involved in the study, mesh generation would became necessary for every different geometric configuration. Future improvements on

this can focus on incorporating mesh generation technologies to the design framework to allow generating or regenerating meshes for updated geometry. Being able to view the generated meshes and assign boundary conditions/loads is also desired as a mesh update would result in changes on those nodes/elements that have prescribed conditions on them.

6.1.2 Supporting Other Types of FEA Simulations

In this research, we focused on structural simulations for tissue-device interaction between a VAB needle and soft tissue. There are many kinds of medical device design problems that can involve other types of FEA simulations, for example, computational fluid dynamics (CFD) simulations for blood flowing through a mechanical heart valve [53], thermal mechanical simulations for shape memory polymer used in expandable stents or other medical devices [54]. To extend the proposed design framework to support other types of simulations, development should focus on the following three aspects:

1. Enhancing the FEA interface to generically read simulation output fields- In structural analyses, displacements, stresses and strains are usually of interests. On the contrary, CFD analyses concern more about fluid flow velocities and pressure fields. An approach to implement this is making the FEA interface capable of reading any user-specified output fields.

2. Redefining the custom NetCDF format- The NetCDF format used in this research is also specifically designed for structural analyses. However, this format is very flexible to extend. For every new output field we can quickly add a new nodal/elemental variable correspondingly.

3. Adding the capability of visualizing new types of data fields- some of the data fields from new types of simulations can be very different from the data fields available in the structural analysis, e.g. flow velocity in CFD. The future research can focus on exploring new ways to better present the data and provide new design insights. To this end, our team has developed a new visualization method in a VR environment [55]. There is a potential to combine the DBD functionality into this VR environment in the future.

6.1.3 Increasing the Usability and Capability of DBD

Future directions for DBD would focus on two main goals. The first is to increase the applicability to other medical device design problems and ease the process of sampling a design space. The second is to improve the capability to contain a much larger design space and enable fast/real-time design space refinement.

A pre-computation is required to integrate the simulation data and sample a design space when applying a new problem to the current DBD. For example, in order to sample the design space for the VAB tissue cutting problem, 32,220 (180×179) warps between 180 FEA simulations were computed for describing the transition of the stress distribution from one simulation instance to another. This process is computationally expensive and involves several steps so that it has to be performed by the developer of the DBD software in a HPC system. The future direction to improve in this aspect could be to simplify this pre-computation process to a single-step process with standardized input and output data format. This eases the difficulty for regular users to apply new problems to the DBD interface.

Another long-term goal could be directly connecting the DBD system to HPC. The computing power will bring in many new possibilities:

1. Exploring a design space containing big data- The design space sampled for the VAB tissue cutting requires roughly 20GB of system memory, which has reached the upper limit of a desktop workstation used in this research. The HPC resources can significantly increase this limit to allow exploring a much larger design space.

2. Interactively refining a design space in real-time- Exploring current design space can lead to new needs to increase design samples in a certain parameter range or expand the design space boundary. The HPC connection would significantly reduce the time cost for new simulations and warp computation. The DBD interface could also intelligently detect the needs based on previous user activities. Updating a design space in real-time can be made possible via the HPC resource. This would provide an important step toward our ultimate goal of simulation-based design.

6.2 Modeling and Simulation of the VAB Cutting

6.2.1 Improving the Constitutive Breast Tissue Models

Obtaining more realistic constitutive breast tissue models is the key to improve the accuracy of the VAB tissue cutting simulation. The future improvements could focus on the following two aims:

1. Improving the tissue deformation models- As reviewed in Chapter 2. Many constitutive models for describing the behavior of the breast tissue deformation are available. However, these models are originally developed to predict the breast tumor displacement due to lying prone during surgery or being compressed during mammography, rather than being rapidly impacted during tissue cutting. Under the conditions of tissue cutting, the high strain rate and thermal effects should be considered. Furthermore, the assumption of isotropic property may not be the best fit for the breast tissue cutting simulation. This assumption could cause inaccurate predictions of the tissue behavior, especially for the fibroglandular tissue, which responses more differently along preferred directions. Constitutive tissue modeling for anisotropic tissue behavior has been an active research direction, but very few studies are conducted for the breast tissue. Existing studies for fibrous tissue/materials [56, 57] may provide excellent start point for working toward a better breast tissue deformation model.

2. Realistic tissue damage models: No breast tissue damage model is available in the literature. In our tissue cutting model we used traditional tensile and shear damage criteria for ductile materials and assumed the fracture points for the tissue. We successfully simulated the tissue cutting, but the simulations could only provide conservative predictions as some important tissue responses (e.g. tissue softening under excessive deformation) were not considered. Fracture toughness is one of the important characteristics that has been used to describe the tissue resistance to the fracture [58, 59, 60]. Identifying the fracture toughness for breast tissue could be a first step toward a realistic tissue damage model. The contact conditions during the cutter progressively fracturing the tissue are also important. Studies in the needle insertion would provide knowledge for developing the contact

force at tissue-cutter interface [61, 62, 63, 64] and the simulation technologies using custom codes [65, 66, 67]. Although the needle insertion is much slower than the VAB tissue cutting, these approaches could be utilized to further develop more realistic tissue-cutter contact conditions.

6.2.2 Investigating New Needle Designs and Cutting Methods

There are various designs for the cutting needle other than the flat tip type studied in this research. Examples are bevel tip needles with different bevel angles and Franseen tip needles. To investigate a new design concepts, geometric features of the needles have to be identified so that different design configurations can be generated. In the tissue cutting simulation, special attention should be paid to the singularity problems which could occur at the area where the sharp needle tip contacts the tissue.

New VAB cutting methods other than the rotation/translation type could be another future direction to be investigated into, such as the scissor cutting method utilized by Senorx EnCor^{TM} VAB device. This could require much more effort to adjust the current tissue cutting model to be able to simulate the new cutting method. Not only the geometry but also the motion of the cutting needle have to be redefined.

6.2.3 Validating the VAB Tissue Cutting Model

Setting up an exactly the same experiment to measure the tissue forces during cutting would be an ideal but a difficult approach to validate the model. First, acquiring breast lesion tissue samples and control them in a right condition for testing is a challenging task. Secondly, fixating the tissue sample to correctly mimic the breast under compression can also be tricky. These uncertainties have to be clarified before pursuing in this direction.

A more achievable approach would be to compare with existing experimental results, such as [34]. It is much easier to modify the simulation settings of our current model, for example, cutting speeds, tissue specimen geometry and tissue properties, to match an existing experimental setup. The variables being assumed in the model, such as the tissue fracture point and the friction coefficient of the tissue-cutter contact, could be calibrated through the comparison. This approach would not reach a fully validated

model, as the cutting condition is changed to match the experiment, but could increase the confidence of the model accuracy.

References

[1] Dane M. Coffey, Chi-Lun Lin, Arthur G. Erdman, and Daniel F. Keefe. Design by Dragging: An Interface for Creative Forward and Inverse Design with Simulation Ensembles. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2783–2791, 2013.

[2] Arthur G. Erdman, Daniel F. Keefe, and Randy. Schiestl. Grand Challenge: Applying Regulatory Science and Big Data to Improve Medical Device Innovation. *Biomedical Engineering, IEEE Transactions on*, 60(3):700–706, 2013.

[3] Daniel F. Keefe, Fotis Sotiropoulos, Victoria Interrante, Birali H. Runesha, Dane M. Coffey, Molly Staker, Chi-Lun Lin, Yi Sun, Iman Borazani, Nancy Row, and Arthur G. Erdman. A Process for Design, Verification, Validation, and Manufacture of Medical Devices using Immersive VR Environments. *Journal of Medical Devices*, 4(4):045002, 2010.

[4] Chi-Lun Lin, Dane M. Coffey, Arthur G. Erdman, and Daniel F. Keefe. A Framework for Medical Device Design Using CAD Synchronization and Remote High-performance FEA Computing. *Journal of Medical Devices*, 6(1):017577, 2012.

[5] Chi-Lun Lin, Ashutosh Srivastava, Dane M. Coffey, Daniel F. Keefe, Marc Horner, Mark Swenson, and Arthur G. Erdman. A System for Optimizing Medical Device Development Using Finite Element Analysis Predictions. *Journal of Medical Devices*, 8(2):020942, 2014.

[6] J Whyte, N Bouchlaghem, A Thorpe, and R McCaffer. From CAD to virtual reality: modelling approaches, data exchange and interactive 3D building design tools. *Automation in Construction*, 10(1):43–55, 2000.

[7] Theophilus Adjei-Kumi and Arkady Retik. A library-based 4d visualisation of construction processes. pages 315–321, 1997.

[8] Vassilis Bourdakis. The future of vrml on large urban models. *Future*, 11:30, 2003.

[9] Ghassan Aouad, Terry Child, Farhi Marir, and Peter Brandon. Developing a virtual reality interface for an integrated project database environment. pages 192–197, 1997.

[10] S. Jayaram, U. Jayaram, Yong Wang, H. Tirumali, K. Lyons, and P. Hart. Vade: a virtual assembly design environment. *Computer Graphics and Applications, IEEE*, 19(6):44–50, Nov 1999.

[11] L. Barbieri, F. Bruno, F. Caruso, and M. Muzzupappa. Innovative Integration Techniques Between Virtual Reality Systems and CAx Tools. *The International Journal of Advanced Manufacturing Technology*, 38(11-12):1085–1097, 2008.

[12] Liu Jiang-sheng, Yao Ying-xue, SA Pahlovy, and Li Jian-guang. A novel data decomposition and information translation method from cad system to virtual assembly application. *The International Journal of Advanced Manufacturing Technology*, 28(3-4):395–402, 2006.

[13] Patrick Bourdot, Thomas Convard, Flavien Picon, Mehdi Ammi, D. Touraine, and J-M Vézien. VR-CAD Integration: Multimodal Immersive Interaction and Advanced Haptic Paradigms for Implicit Edition of CAD Models. *Computer-Aided Design*, 42(5):445–461, 2010.

[14] L.X. Ng, S.W. Oon, S.K. Ong, and A.Y.C. Nee. GARDE: A Gesture-Based Augmented Reality Design Evaluation System. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 5(2):85–94, 2011.

[15] Sang Hun Lee. A CAD-CAE Integration Approach Using Feature-Based Multi-Resolution and Multi-Abstraction Modelling Techniques. *Computer-Aided Design*, 37(9):941–955, 2005.

[16] G.P. Gujarathi and Y-S Ma. Parametric CAD/CAE Integration Using a Common Data Model. *Journal of Manufacturing Systems*, 30(3):118–132, 2011.

[17] J.C.H. Goh, P.V.S. Lee, S.L. Toh, and C.K. Ooi. Development of an Integrated CAD-FEA Process for Below-Knee Prosthetic Sockets. *Clinical Biomechanics*, 20(6):623–629, 2005.

[18] Giorgio Colombo, Giancarlo Facoetti, and Caterina Rizzi. A Digital Patient for Computer-Aided Prosthesis Design. *Interface focus*, 3(2):20120082, 2013.

[19] N. J. B. McFarlane, X. Lin, Y. Zhao, G. J. Clapworthy, F. Dong, A. Redaelli, O. Parodi, and D. Testi. Visualization and Simulated Surgery of the Left Ventricle in the Virtual Pathological Heart of the Virtual Physiological Human. *Interface focus*, 1(3):374–383, 2011.

[20] Peter V. Coveney, Vanessa Diaz, Peter Hunter, Peter Kohl, and Marco Viceconti. The Virtual Physiological Human. *Interface Focus*, 1(3):281–285, 2011.

[21] Bernard W. Stewart and Christopher P. Wild. *World Cancer Report 2014*. International Agency for Research on Cancer. World Health Organization, 2014.

[22] Rebecca Siegel, Deepa Naishadham, and Ahmedin Jemal. Cancer Statistics, 2013. *CA: A Cancer Journal for Clinicians*, 63(1):11–30, 2013.

[23] N Howlader, AM Noone, M Krapcho, J Garshell, D Miller, SF Altekruse, CL Kosary, M Yu, J Ruhl, Z Tatalovich, A Mariotto, DR Lewis, HS Chen, EJ Feuer, and KA(eds) Cronin. "SEER Cancer Statistics Review, 1975-2011". National Cancer Institute. Bethesda, MD, 2013.

[24] E.A.M. O'Flynn, A.R.M. Wilson, and M.J. Michell. Image-Guided Breast Biopsy: State-of-the-Art. *Clinical Radiology*, 65(4):259–270, 2010.

[25] Ian Grady, Heidi Gorsuch, and Shelly Wilburn-Bailey. Ultrasound-Guided, Vacuum-Assisted, Percutaneous Excision of Breast Lesions: An Accurate Technique in the Diagnosis of Atypical Ductal Hyperplasia. *Journal of the American College of Surgeons*, 201(1):14–17, 2005.

[26] Enrico Cassano, Linei ABD Urban, Maria Pizzamiglio, Francesca Abbate, Patrick Maisonneuve, Giuseppe Renne, Giuseppe Viale, and Massimo Bellomi. Ultrasound-Guided Vacuum-Assisted Core Breast Biopsy: Experience with 406 Cases. *Breast Cancer Research and Treatment*, 102(1):103–110, 2007.

[27] Xinghua Liang, Markus Hahn, Mohamed S Ebeida, Karl Oliver Kagan, Yongjie Zhang, Katja Claudia Siegmann, Ute Krainick-Strobel, Bernhard Kraemer, Tanja Fehm, Eva Fischbach, Diethelm Wallwiener, and Ines Gruber. Mammotome® versus ATEC®: a comparison of two breast vacuum biopsy techniques under sonographic guidance. *Archives of Gynecology and Obstetrics*, 281(2):287–292, April 2009.

[28] Shambhavi Venkataraman, Vandana Dialani, Hannah L. Gilmore, and Tejas S. Mehta. Stereotactic core biopsy: Comparison of 11 gauge with 8 gauge vacuum assisted breast biopsy. *European Journal of Radiology*, 81(10):2613 – 2619, 2012.

[29] Steve H. Parker, Mark A. Dennis, A. Thomas Stavros, and Kevin K. Johnson. Ultrasound-Guided Mamtmeotomoy: A New Breast Biopsy Technique. *Journal of Diagnostic Medical Sonography*, 12(3):113–118, 1996.

[30] Steve H Parker, Anita J Klaus, Patrick J Mc Wey, Kathy J Schilling, Tommy E Cupples, Nathalie Duchesne, Mark A Guenin, and Jay K Harness. Sonographically Guided Directional Vacuum-Assisted Breast Biopsy Using a Handheld Device. *American Journal of Roentgenology*, 177(2):405–408, 2001.

[31] AG Atkins, Xianzhong Xu, and G Jeronimidis. Cutting, by 'Pressing and Slicing,'of Thin Floppy Slices of Materials Illustrated by Experiments on Cheddar Cheese and Salami. *Journal of Materials Science*, 39(8):2761–2766, 2004.

[32] Tony Atkins. *The Science and Engineering of Cutting*. Butterworth-Heinemann. Oxford, UK, Chap. 5, 2009.

[33] E Reyssat, T Tallinen, M Le Merrer, and L Mahadevan. Slicing Softly with Shear. *Physical Review Letters*, 109(24):244301, 2012.

[34] Peidong Han and Kornel Ehmann. Study of the Effect of Cannula Rotation on Tissue Cutting for Needle Biopsy. *Medical engineering & physics*, 35(11):1584–1590, 2013.

[35] Parris Wellman, Robert D Howe, Edward Dalton, and Kenneth A Kern. "Breast Tissue Stiffness in Compression is Correlated to Histological Diagnosis". Harvard Biorobotics Laboratory, Cambridge, MA, 1999.

[36] A Pérez Del Palomar, B. Calvo, J. Herrero, J. Lopez, and M. Doblaré. A Finite Element Model to Accurately Predict Real Deformations of the Breast. *Medical Engineering & Physics*, 30(9):1089–1097, 2008.

[37] P. Pathmanathan, D.J. Gavaghan, J.P. Whiteley, S.J. Chapman, and J.M. Brady. Predicting Tumor Location by Modeling the Deformation of the Breast. *Biomedical Engineering, IEEE Transactions on*, 55(10):2471–2480, 2008.

[38] Christine Tanner, Julia A. Schnabel, Derek L. G. Hill, David J. Hawkes, Martin O. Leach, and D. Rodney Hose. Factors influencing the accuracy of biomechanical breast models. *Medical Physics*, 33(6):1758–1769, 2006.

[39] Vijay Rajagopal, Poul M F Nielsen, and Martyn P Nash. Modeling breast biomechanics for multi-modal image analysis-successes and challenges. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 2(3):293–304, October 2009.

[40] Abbas Samani and Donald Plewes. A method to measure the hyperelastic parameters of ex vivo breast tissue samples. *Physics in Medicine and Biology*, 49(18):4395, 2004.

[41] Gerhard A Holzapfel. Biomechanics of soft tissue. *The handbook of materials behavior models*, 3:1049–1063, 2001.

[42] Dane M Coffey. *Scalable natural user interfaces for data-intensive exploratory visualization: designing in the context of big data.* PhD thesis, UNIVERSITY OF MINNESOTA, 2014.

[43] Unidata. Network Common Data Form (NetCDF). On the WWW, 2014. URL `http://www.unidata.ucar.edu/software/netcdf`.

[44] Abraham Savitzky and Marcel JE Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964.

[45] Maxon Motor. Maxon academy. On the WWW, July 2014. URL `http://www.maxonmotorusa.com/maxon/view/content/academy`.

[46] Simon P DiMaio and Septimiu E Salcudean. Needle Insertion Modeling and Simulation. *Robotics and Automation, IEEE Transactions on*, 19(5):864–875, 2003.

[47] L S Kaplan, L J Kelly, and D L Hamann. Biopsy needle instrument, February 2004.

[48] F.J. Viola, J. Dale, and A.C. Stickney. Biopsy instrument driver apparatus, April 2003. US Patent 6,554,779.

[49] J.A. Hibner, R.P. Nuchols, E.A. Rhad, and H.W. Craig. Handheld biopsy device with needle firing, September 2013. EP Patent App. EP20,110,838,420.

[50] J.R. Donahue and G. Smith. Surgical instrument, February 1997. US Patent 5,601,583.

[51] Ran Gal, Olga Sorkine, Niloy J Mitra, and Daniel Cohen-Or. iwires: an analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics (TOG)*, 28(3):33, 2009.

[52] Niloy J Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala. Illustrating how mechanical assemblies work. *ACM Transactions on Graphics (TOG)*, 29(4):58, 2010.

[53] Trung Bao Le and Fotis Sotiropoulos. Fluid–structure interaction of an aortic heart valve prosthesis driven by an animated anatomic left ventricle. *Journal of computational physics*, 244:41–62, 2013.

[54] P.R. Buckley, G.H. McKinley, T.S. Wilson, W. Small, W.J. Benett, J.P. Bearinger, M.W. McElfresh, and D.J. Maitland. Inductively heated shape memory polymer for the magnetic actuation of medical devices. *Biomedical Engineering, IEEE Transactions on*, 53(10):2075–2083, Oct 2006.

[55] Dane Coffey, Nicholas Malbraaten, Trung Le, Iman Borazjani, Fotis Sotiropoulos, and Daniel F Keefe. Slice wim: a multi-surface, multi-touch interface for overview+ detail exploration of volume datasets in virtual reality. In *Symposium on Interactive 3D Graphics and Games*, pages 191–198. ACM, 2011.

[56] V Alastrué, JF Rodríguez, B Calvo, and M Doblare. Structural damage models for fibrous biological soft tissues. *International Journal of solids and Structures*, 44(18):5894–5911, 2007.

[57] Michael S Sacks and Wei Sun. Multiaxial mechanical behavior of biological materials. *Annual review of biomedical engineering*, 5(1):251–284, 2003.

[58] CF Doran, BAO McCormack, and A Macey. A simplified model to determine the contribution of strain energy in the failure process of thin biological membranes during cutting. *Strain*, 40(4):173–179, 2004.

[59] A G Atkins. Toughness and cutting: a new way of simultaneously determining ductile fracture toughness and strength. *Engineering Fracture Mechanics*, 72(6):849–860, April 2005.

[60] David Taylor, Niamh O'Mara, Eoin Ryan, Michael Takaza, and Ciaran Simms. The fracture toughness of soft tissues. *Journal of the Mechanical Behavior of Biomedical Materials*, 6(C):139–147, February 2012.

[61] A M Okamura, C Simone, and M D O'Leary. Force Modeling for Needle Insertion Into Soft Tissue. *IEEE Transactions on Biomedical Engineering*, 51(10):1707–1716, October 2004.

[62] S Misra, K B Reed, A S Douglas, K T Ramesh, and A M Okamura. Needle-tissue interaction forces for bevel-tip steerable needles. pages 224–231, 2008.

[63] Mohsen Mahvash and Pierre E Dupont. Mechanics of Dynamic Needle Insertion into a Biological Material. *IEEE Transactions on Biomedical Engineering*, 57(4):934–943, 2010.

[64] Dennis J van Gerwen, Jenny Dankelman, and John J van den Dobbelsteen. Needle–tissue interaction forces – A survey of experimental data. *Medical engineering & physics*, 134(6):665–680, July 2012.

[65] Teeranoot Chanthasopeephan, Jaydev P Desai, and Alan CW Lau. Measuring forces in liver cutting: New equipment and experimental results. *Annals of biomedical engineering*, 31(11):1372–1382, 2003.

[66] S P DiMaio and S E Salcudean. Interactive Simulation of Needle Insertion Models. *IEEE Transactions on Biomedical Engineering*, 52(7):1167–1179, July 2005.

[67] Nuttapong Chentanez, Ron Alterovitz, Daniel Ritchie, Lita Cho, Kris K Hauser, Ken Goldberg, Jonathan R Shewchuk, and James F O'Brien. *Interactive simulation of surgical needle insertion and steering*, volume 28. ACM, 2009.

[68] M& M International. Tubing chart. On the WWW, November 2014. URL `http://mmtubing.com/tubing-chart/`.

[69] Q.-H. Wang, J.-R. Li, and H.-Q. Gong. A cad-linked virtual assembly environment. *International Journal of Production Research*, 44(3):467–486, 2006.

Appendix A

Needle Gauge Chart

A list of needle gauges is provided in Table A.1 [68]. A gauge number is used to define the outer radius (OD) of a hypodermic needle. Different wall thicknesses (W) are available for each needle gauge number. The most common configurations are regular wall (RW), thin wall (TW) and extra thin wall (XTW). Once the gauge and wall thickness are specified, the inner radius of a needle can be determined by Eqn. A.1.

$$ID = OD - 2W \tag{A.1}$$

Table A.1: List of needle gauges

G	Nominal OD (in)	Nominal ID (in)	Nominal W (in)
3G-RW	0.259	0.219	0.02
3G-TW	0.259	0.229	0.015
3G-XTW	0.259	0.239	0.01
4G-RW	0.238	0.198	0.02
4G-TW	0.238	0.208	0.015
4G-XTW	0.238	0.218	0.01
5G-RW	0.219	0.189	0.015
5G-TW	0.219	0.199	0.01
5G-XTW	0.219	0.205	0.007

Continued on next page

Table A.1 – continued from previous page

G	Nominal OD (in)	Nominal ID (in)	Nominal W (in)
6G-RW	0.203	0.173	0.015
6G-TW	0.203	0.183	0.01
6G-XTW	0.203	0.189	0.007
7G-RW	0.18	0.15	0.015
7G-TW	0.18	0.16	0.01
7G-XTW	0.18	0.166	0.007
8G-RW	0.165	0.135	0.015
8G-TW	0.165	0.145	0.01
8G-XTW	0.165	0.151	0.007
9G-RW	0.148	0.118	0.015
9G-TW	0.148	0.128	0.01
9G-XTW	0.148	0.135	0.0065
10G-RW	0.134	0.106	0.014
10G-TW	0.134	0.114	0.01
10G-XTW	0.134	0.118	0.008
11G-RW	0.12	0.094	0.013
11G-TW	0.12	0.1	0.01
11G-XTW	0.12	0.104	0.008
12G-RW	0.109	0.085	0.012
12G-TW	0.109	0.091	0.009
12G-XTW	0.109	0.097	0.006
13G-RW	0.095	0.071	0.012
13G-TW	0.095	0.077	0.009
13G-XTW	0.095	0.081	0.007
14G-RW	0.083	0.063	0.01
14G-TW	0.083	0.0673	0.0079
14G-XTW	0.083	0.072	0.0055
15G-RW	0.072	0.054	0.009

Continued on next page

Table A.1 – continued from previous page

G	Nominal OD (in)	Nominal ID (in)	Nominal W (in)
15G-TW	0.072	0.0605	0.0058
15G-XTW	0.072	0.0628	0.0046
16G-RW	0.065	0.047	0.009
16G-TW	0.065	0.0535	0.0058
16G-XTW	0.065	0.056	0.0045
17G-RW	0.058	0.042	0.008
17G-TW	0.058	0.0475	0.0053
17G-XTW	0.058	0.05	0.004
18G-RW	0.05	0.033	0.0085
18G-TW	0.05	0.0385	0.0058
18G-XTW	0.05	0.042	0.004
19G-RW	0.042	0.027	0.0075
19G-TW	0.042	0.0325	0.0048
19G-XTW	0.042	0.035	0.0035
20G-RW	0.0358	0.0238	0.006
20G-TW	0.0358	0.0263	0.0048
20G-XTW	0.0358	0.0278	0.004
21G-RW	0.0323	0.0203	0.006
21G-TW	0.0323	0.0233	0.0045
21G-XTW	0.0323	0.026	0.0031
22G-RW	0.0283	0.0163	0.006
22G-TW	0.0283	0.0198	0.0043
22G-XTW	0.0283	0.0213	0.0035
22.5G-RW	0.0263	0.0178	0.0043
22.5G-TW	0.0263	0.0203	0.003
22.5G-XTW	0.0263	0.0213	0.0025
23G-RW	0.0253	0.0133	0.006
23G-TW	0.0253	0.0173	0.004

Continued on next page

Table A.1 – continued from previous page

G	Nominal OD (in)	Nominal ID (in)	Nominal W (in)
23G-XTW	0.0253	0.0193	0.003
24G-RW	0.0223	0.0123	0.005
24G-TW	0.0223	0.0143	0.004
24G-XTW	0.0223	0.0163	0.003
25G-RW	0.0203	0.0103	0.005
25G-TW	0.0203	0.0123	0.004
25G-XTW	0.0203	0.0143	0.003
26G-RW	0.0183	0.0103	0.004
26G-TW	0.0183	0.0123	0.003
26G-XTW	0.0183	0.0138	0.0023
27G-RW	0.0163	0.0083	0.004
27G-TW	0.0163	0.0103	0.003
27G-XTW	0.0163	0.012	0.0021
28G-RW	0.0143	0.0073	0.0035
28G-TW	0.0143	0.0093	0.0025
28G-XTW	0.0143	0.0105	0.0019
29G-RW	0.0133	0.0073	0.003
29G-TW	0.0133	0.0085	0.0024
29G-XTW	0.0133	0.0095	0.0019
30G-RW	0.0123	0.0063	0.003
30G-TW	0.0123	0.0073	0.0025
30G-XTW	0.0123	0.0083	0.002
31G-RW	0.0103	0.0048	0.0028
32G-RW	0.0093	0.0043	0.0025

Appendix B

The SolidWorks API Applications for Synchronizing CAD Parameters

This appendix describes how to develop SolidWorks API software applications using C^{++} for directly communicating with the SolidWorks kernel.

B.1 SolidWorks API C^{++} Application

This section provides essential parts of the source code of the standalone SolidWorks API C++ application developed in this research.

To create an executable project of SolidWorks API application, two essential SolidWorks libraries: SolidWorks type library and constant type library must be imported. In our program, these libraries are imported in the standard system include files (*stdafx.h*):

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently,
   but
// are changed infrequently
//
```

```
#pragma once

#include "targetver.h"

#include <stdio.h>
#include <tchar.h>


#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS      // some CString
    constructors will be explicit

#include <atlbase.h>
#include <atlstr.h>

// Additional headers required
// SolidWorks type library
#import "C:\Program Files\SolidWorks Corp\SolidWorks\sldworks.
    tlb" raw_interfaces_only, raw_native_types, no_namespace,
    named_guids
// SolidWorks constant type library
#import "C:\Program Files\SolidWorks Corp\SolidWorks\swconst.
    tlb"  raw_interfaces_only, raw_native_types, no_namespace,
    named_guids
```

In this application, all of the SolidWorks CAD functions are implemented in a class called SWCore. The essential parts of this class and example functions are provided in the following cpp and header files:

```
File: SWCore.h
#pragma once

#include <iostream>
#include <fstream>
```

```
using namespace std;

class SWCore
{
public:
  double m_nOldRout, m_nOldRin, m_nMainLength, m_nOpnCntr,
      m_nOpnLeng;
  double m_nCutTubeRout, m_nCutTubeRin, m_nCutTubeLength;

  SWCore(void);
  ~SWCore(void);
  int CreateModel(double CenterPx, double CenterPy, double
      CenterPz, double Rout, double Rin, double Length, double
      OpnCntr, double OpnLeng, double CutRin, double CutRout,
      double CutLength);
  int ModifyModel(double CenterPx, double CenterPy, double
      CenterPz, double Rout, double Rin, double Length, double
      OpnCntr, double OpnLeng, double CutRin, double CutRout,
      double CutLength);
  int OutputTess(void);
  bool TraverseFeaturesAndOperate(CComBSTR sKeyword, int
      nOperation, IModelDoc2* swModel, BSTR bFeatType=L"", int
      bSubFeatType= -1);
  // Op1:Select but not append, Op2:Select and append, Op3:
      Rename, Op4.Delete
  BSTR GetSketchByFeatureName(BSTR bFeatName, ISketch**
      swSketch, IModelDoc2* swModel);
  int ModifySketchArc(double Radius, CComBSTR bFeatName,
      IModelDoc2* swModel);
  int ExtrusionCut_Rect(double P1x, double P1y, double P1z,
      double P2x, double P2y, double P2z, IModelDoc2* swModel);
```

```
  BSTR Extrusion(int nType, Point3 Para1, double nPara2, double
      nPara3, BSTR bBaseSurface, IModelDoc2* swModel); //
      return created feature name
  int SetExtrusionDepth(CComBSTR bFeatName, double nLength,
      IModelDoc2* swModel);
  BSTR ExtrusionCut(int nType, Point3 Para1, double nPara2,
      double nPara3, BSTR bBaseSurface, IModelDoc2* swModel);
  bool SaveToSTEP(BSTR bPath);
};
```

File: SWCore.cpp

```
#include "StdAfx.h"
#include "SWCore.h"

// ATL smart pointers
CComPtr<ISldWorks> m_swApp; // SolidWorks Application instance
    running behind
CComPtr<IModelDoc2> m_swModel; // A CAD model document
CComPtr<IPartDoc> m_swPart; // A part document
CComPtr<IModelDocExtension> m_swDocExt; // For additional model
    commands
VARIANT_BOOL m_bRetVal;

SWCore::SWCore(void)
{
  CoInitialize(NULL); //Initialize COM
  // Create an instance of SolidWorks
  m_swApp.CoCreateInstance(L"SldWorks.Application", NULL,
      CLSCTX_LOCAL_SERVER);
```

```
    // Get the active model document
    ////////////////////////////////////////////
    // Model 1
    CComPtr<IDispatch> swDispatch;
    BSTR bTitle;
    m_swApp->GetFirstDocument(&swDispatch);
    swDispatch->QueryInterface(IID_IModelDoc2, reinterpret_cast<
        void**>(&m_swModel1));
    swDispatch.Release();

    // Get the title and other objects belong to this part
    m_swModel1->GetTitle(&bTitle);
    m_swModel1->get_Extension(&m_swDocExt1);
    m_swModel1->get_ActiveView(&swDispatch);
    swDispatch->QueryInterface(IID_IModelView, reinterpret_cast<
        void**>(&m_swModelView1));
    swDispatch.Release();

    // Model 2
    m_swModel1->GetNext(&swDispatch);
    swDispatch->QueryInterface(IID_IModelDoc2, reinterpret_cast<
        void**>(&m_swModel2));
    swDispatch.Release();
}

SWCore::~SWCore(void)
{
    //---Clean-----------------------------------
    m_swPart.Release();
    m_swModel.Release();
    m_swApp->ExitApp(); // Shut down SolidWorks
    m_swApp.Release();  // Release COM reference
```

```
  CoUninitialize(); //Uninitialize COM
  //-----------------------------------------
}
int SWCore::ModifyModel(double CenterPx, double CenterPy,
    double CenterPz, double Rout, double Rin, double Length,
    double OpnCntr, double OpnLeng, double CutRin, double
    CutRout, double CutLength)
{
  // Example function: Modify inner radius of the outer cannula
  ModifySketchArc(Rin, m_strMainCut.GetString(), m_swModel1);
  ModifySketchArc(Rout, m_strMain.GetString(), m_swModel1);

  if (OpnCntr != m_nOpnCntr || OpnLeng != m_nOpnLeng)
  {
    // Delete the old openning
    TraverseFeaturesAndOperate(m_strOpnCut.GetString(), 4,
        m_swModel1);
    // Create a opening
    double L1 = OpnCntr-OpnLeng;
    double L2 = OpnLeng*2;

    if (ExtrusionCut_Rect(CenterPx-Rout,-L1,0,CenterPx+Rout,-(
        L1+L2),0, m_swModel1))
    {
      m_nOpnCntr = OpnCntr;
      m_nOpnLeng = OpnLeng;
    }
  }

  if (m_nMainLength != Length)
  {
```

```
    if (SetExtrusionDepth(m_strMain.GetString(),Length,
        m_swModel1))
      m_nMainLength = Length;
  }

  if (m_nCutTubeLength != CutLength)
  {
    if (SetExtrusionDepth(m_strCutTube.GetString(),CutLength,
        m_swModel2))
      m_nCutTubeLength = CutLength;
  }

  return 1;
}

// Example function: create the outer cannula
int SWCore::CreateModel(double CenterPx, double CenterPy,
    double CenterPz, double Rout, double Rin, double Length,
    double OpnCntr, double OpnLeng, double CutRin, double
    CutRout, double CutLength)
{
  // Create main cynlider
  BSTR bFeatName;
  bFeatName = Extrusion(1, pCenter, Rout, Length, L"Front_Plane
      ", m_swModel1);
  m_strMain = CString(bFeatName);
  m_nOldRin = Rin;
  m_nOldRout = Rout;
  m_nMainLength = Length;

  // Create a extrusion cut
  CString strTmp;
```

```
  strTmp.Format(_T("FACE@%s"), m_strMain);
  bFeatName = ExtrusionCut(1,pCenter,Rin, Length, strTmp.
      AllocSysString(), m_swModel1);
  m_strMainCut = CString(bFeatName);

  // Create a opening
  double L1 = OpnCntr-OpnLeng;//OpnCntr-0.5*OpnLeng;
  double L2 = OpnLeng*2;

  if (ExtrusionCut_Rect(CenterPx-Rout,-L1,0,CenterPx+Rout,-(L1+
      L2),0,m_swModel1))
  {
    m_nOpnCntr = OpnCntr;
    m_nOpnLeng = OpnLeng;
  }

  return 1;
}

// Example function: traverse the features in the model tree
bool SWCore::TraverseFeaturesAndOperate(CComBSTR sKeyword, int
    nOperation, IModelDoc2* swModel, BSTR bFeatType, int
    bSubFeatType)
{
  // Traverse FeatureManager design tree to get the specified
      feature
  CString strKeyword = CString(sKeyword);
  CString strFeatType = CString(bFeatType);
  CComBSTR sGetFeatName(L"");
  CComBSTR sGetFeatType(L"");
  bool bFound = false;
```

```
// ATL smart pointers
CComPtr<IFeature> swFeat;
CComPtr<IFeature> swSubFeat;

HRESULT hres = swModel->IFirstFeature(&swFeat);
VARIANT_BOOL RetVal;

// If the name of the feature is "Boss-Extrude1"
// then select the feature
while (swFeat)
{
  hres = swFeat->get_Name(&sGetFeatName);
  CString strGetFeatName = CString(sGetFeatName);
  hres = swFeat->GetTypeName(&sGetFeatType);
  CString strGetFeatType = CString(sGetFeatType);

  if (strGetFeatName.Find(strKeyword) != -1)
  {
    if (nOperation == 1 && strGetFeatName == strKeyword) //
        Select but not append
    {
      hres = swFeat->Select2(false, 1, &RetVal);
      bFound = true;
      break;
    }
    else if (nOperation == 2 && strGetFeatName == strKeyword)
        // Select and append
    {
      hres = swFeat->Select2(true, 1, &RetVal);
      bFound = true;
      break;
    }
```

```
else if (nOperation == 3 && strGetFeatType == strFeatType
    ) // Rename
{
  bFound = true;
}
else if (nOperation == 4 && strGetFeatName == strKeyword)
    // Delete this feature and all of the subfeatures
{
  // Assume the case: one feature based on one sketch
  bool bDelete = false;
  hres = swFeat->IGetFirstSubFeature(&swSubFeat);    //
     Get the subfeature
  if (swSubFeat)
  {
    BSTR bTmp;
    hres = swSubFeat->GetTypeName(&bTmp);
    CString strTmp = CString(bTmp);
    if (strTmp == "ProfileFeature") // main sketch
    {
      CComPtr<IUnknown> swUnknown;
      CComPtr<ISketch>  swSketch;
      CComPtr<ISketchSegment>  swSktSeg;

      // Get the Sketch interface
      hres = swSubFeat->IGetSpecificFeature(&swUnknown);
      hres = swUnknown->QueryInterface(__uuidof(ISketch),
          reinterpret_cast<void**>(&swSketch));
      swUnknown.Release();

      if ( (hres == S_OK) && (swSketch != NULL) )
      {
        VARIANT vSegments;
```

```
        VariantInit(&vSegments);
        SAFEARRAY *psa;
        swSketch->GetSketchSegments(&vSegments);
        psa = vSegments.parray;

        CComPtr<IUnknown>* pData;
        ::SafeArrayAccessData(psa, (void **)&pData);
        swSktSeg = pData[0];
        //swUnknown->QueryInterface(__uuidof(
            ISketchSegment), reinterpret_cast<void**>(&
            swSktSeg));
        long nType;
        swSktSeg->GetType(&nType);
        if (nType == swSketchLINE) // swSketchLINE,
            swSketchELLIPSE, swSketchSPLINE, ...
          bDelete = true;

        ::SafeArrayUnaccessData(psa);
      }
    }
  }
  if (bDelete)
  {
    // Delete feature first
    hres = swFeat->Select2(false, 1, &RetVal);
    swModel->EditDelete();
    swModel->ForceRebuild3(VARIANT_TRUE, &RetVal);
    // Then delete subfeature
    swSubFeat->Select2(false, 1, &RetVal);
    swModel->EditDelete();
    swModel->ForceRebuild3(VARIANT_TRUE, &RetVal);
    bFound = true;
```

```
        }
      }
    }

    CComPtr<IFeature> swFeatureNext;
    hres = swFeat->IGetNextFeature(&swFeatureNext);
    swFeat.Release();
    swFeat = swFeatureNext;
    swFeatureNext.Release();
    swSubFeat.Release();

  }
  swFeat.Release();
  return bFound;
}

// Example function: Get the base sketch of a feature by a
   specified feature name.
BSTR SWCore::GetSketchByFeatureName(BSTR bFeatName, ISketch**
   swSketch, IModelDoc2* swModel)
{
  HRESULT hres;
  BSTR bSubFeatName(L"");      // the sketch name to be returned
  CString strPrint;
  CString strFeatName(bFeatName); // for comparison with
     current feature name

  // ATL Smart Pointers
  CComPtr<IFeature> swFeat;
  CComPtr<IFeature> swSubFeat;
  CComPtr<IUnknown> swUnknown;
  CComPtr<IFeature> swNextFeature;
```

```
CComPtr<IFeature> swNextSubFeature;

// Get first feature in document
hres = swModel->IFirstFeature(&swFeat);

// Start to browse features
while (swFeat)
{
  BSTR bCurFeatName;
  hres = swFeat->get_Name(&bCurFeatName);          // Get the
      feature name
  strPrint = CString(bCurFeatName);
  //printf("Feature:%25S\n", strPrint);
  hres = swFeat->IGetFirstSubFeature(&swSubFeat);    // Get
      the subfeature

  CString strCurFeatName(bCurFeatName);
  // start to browse subfeatures
  while (swSubFeat)
  {
    BSTR bFeatureTypeName;
    BSTR bSubFeatureName;
    hres = swSubFeat->get_Name(&bSubFeatureName);
    strPrint = CString(bSubFeatureName);
    //printf("Subfeature:%25S\n", strPrint);

    // Get the subfeature type
    hres = swSubFeat->GetTypeName(&bFeatureTypeName);
    CString strFeatTypeName(bFeatureTypeName);
    // If the sub-feature is a Sketch
    if (strFeatTypeName == "ProfileFeature" && strCurFeatName
        == strFeatName)
```

```
    {
      // Get the Sketch interface
      hres = swSubFeat->IGetSpecificFeature(&swUnknown);
      ISketch* swSelectedSketch;
      hres = swUnknown->QueryInterface(__uuidof(ISketch),
          reinterpret_cast<void**>(&swSelectedSketch));
      *swSketch = swSelectedSketch;
      if ( (hres == S_OK) && (swSketch != NULL) )
      {
        // Get the sub-feature name
        hres = swSubFeat->get_Name(&bSubFeatName);
      }
    }
    // Get next subfeature
    hres = swSubFeat->IGetNextSubFeature(&swNextSubFeature);
    swSubFeat.Release();
    swSubFeat = swNextSubFeature;
    swNextSubFeature.Release();
  }

  // Get next feature
  hres = swFeat->IGetNextFeature(&swNextFeature);
  swFeat.Release();
  swFeat = swNextFeature;
  swNextFeature.Release();
}

// Release
swFeat.Release();
swSubFeat.Release();
swUnknown.Release();
swNextFeature.Release();
```

```
  swNextSubFeature.Release();

  return bSubFeatName;
}

// Example function: Modify a sketch arc with a specified
    radius
int SWCore::ModifySketchArc(double Radius, CComBSTR bFeatName,
    IModelDoc2* swModel)
{
  CComPtr<ISketch>           swSketch;
  CComPtr<ISketchArc>        swSketchArc;
  CComPtr<ISketchSegment>    swSktSeg;
  CComPtr<IUnknown>          swUnknown;

  BSTR bSketch = GetSketchByFeatureName(bFeatName, &swSketch,
      swModel);
  CString tmp = CString(bSketch);
  if (tmp == "")
  {
    printf("Error: No conrresponding feature found.");
    return 0;
  }

  VARIANT vSegments;
  VariantInit(&vSegments);
  SAFEARRAY *psa;

  // May try IEnumSketchSegments
  swSketch->GetSketchSegments(&vSegments);
  psa = vSegments.parray;
```

```
  CComPtr<IUnknown>* pData1;
  ::SafeArrayAccessData(psa, (void **)&pData1);
  swUnknown = pData1[0];
  swUnknown->QueryInterface(__uuidof(ISketchSegment),
      reinterpret_cast<void**>(&swSktSeg));
  long nType;
  swSktSeg->GetType(&nType);

  // Prepare SketchArcOut for later change
  if (nType == swSketchARC) // swSketchLINE, swSketchELLIPSE,
      swSketchSPLINE, ...
  {
    swModel->ClearSelection2(VARIANT_TRUE);
    swModel->SelectByName(0, bSketch);
    swModel->EditSketch();
    swSketchArc = swSktSeg;
    swSketchArc->SetRadius(Radius, &m_bRetVal);
    swModel->ClearSelection2(VARIANT_TRUE);                    //
        exit sketch mode
    swModel->InsertSketch2(VARIANT_TRUE);
  }
  else
    printf("Error: Not a arc sketch.");

  swSketchArc.Release();
  swSktSeg.Release();
  swUnknown.Release();

  ::SafeArrayUnaccessData(psa);

  return 0;
}
```

```
// Example function: Create a rectangle shape opening on a part
    object
int SWCore::ExtrusionCut_Rect(double P1x, double P1y, double
    P1z, double P2x, double P2y, double P2z, IModelDoc2* swModel
    )
{
  CComPtr<ISketchManager>  swSketchMgr;
  CComPtr<ISketch>         swSketch;
  CComPtr<ISelectionMgr>   swSelMgr;
  CComPtr<IDispatch>       swDispatch;
  CComPtr<IFeature>        swFeat;
  BSTR bFeatType, bFeatName;
  HRESULT hr;
  VARIANT vRet;

  try
  {
    // Prepare selection manager
    swModel->get_ISelectionManager(&swSelMgr);

    // Create a rectangle sketch
    swModel->InsertSketch2(VARIANT_TRUE);
    TraverseFeaturesAndOperate(L"Top_Plane", 1, swModel); //
        Select the plane
    swModel->get_SketchManager(&swSketchMgr);
    hr = swSketchMgr->CreateCornerRectangle(P1x,P1y,P1z,P2x,P2y
        ,P2z,&vRet);
    if (hr != S_OK)
    {
      printf("Failed to create the sketch.");
      return 0;
```

```
}

// Get the active sketch name for future usage
if (swSketchMgr->get_ActiveSketch(&swSketch) == S_OK)
{
  swSketch->QueryInterface(__uuidof(IFeature),
      reinterpret_cast<void**>(&swFeat));
  swFeat->GetNameForSelection(&bFeatType, &bFeatName);
  m_strOpnCutSkt = CString(bFeatName);
}
else
{
  printf("Failed to get the active sketch.");
  return 0;
}

swModel->InsertSketch2(VARIANT_TRUE);
swModel->ClearSelection2(VARIANT_TRUE);

// Create a extrusion feature
TraverseFeaturesAndOperate(bFeatName, 1, swModel); //
    Select the sketch
hr = swModel->FeatureCut(1,0,0,1,0,0.01,0.01,\
  0,0,0,0,0.01745329251994,0.01745329251994,0,0);
if (hr != S_OK)
{
  printf("Failed to create the cut feature.");
  if (m_strOpnCutSkt != "")
    TraverseFeaturesAndOperate(m_strOpnCutSkt.GetString(),
        4, swModel); // delete the created sketch
  return 0;
}
```

```
    // Clean
    swFeat.Release();
    swSketch.Release();

    // Save the feature name
    swSelMgr->GetSelectedObject5(1, &swDispatch);
    swDispatch->QueryInterface(__uuidof(IFeature),
        reinterpret_cast<void**>(&swFeat));
    swFeat->GetNameForSelection(&bFeatType, &bFeatName);
    m_strOpnCut = CString(bFeatName);
    swModel->ClearSelection2(VARIANT_TRUE);

    swSketchMgr.Release();
    swSelMgr.Release();
    swDispatch.Release();
    swFeat.Release();

    return 1;
  }
  catch (char* str)
  {
    cout << "Exception raised: " << str << '\n';
    return 0;
   }
}

// Example function: create a extrusion
BSTR SWCore::Extrusion(int nType, Point3 Para1, double nPara2,
    double nPara3, BSTR bBaseSurface, IModelDoc2* swModel)
{
  // Declaration
```

```
CComPtr<ISketchManager>          swSketchMgr;
CComPtr<ISketch>                 swSketch;
CComPtr<IFeature>                swFeat;
CComPtr<IFeatureManager>         swFeatMgr;
CComPtr<ISketchSegment>          swSktSeg;
CComPtr<ISelectData>             swSelData;
BSTR bFeatType, bFeatName;
VARIANT_BOOL bRet;

// Prepare sketch manager
swModel->get_SketchManager(&swSketchMgr);

if (nType == 1) // Circle
{
  // Sketch
  TraverseFeaturesAndOperate(bBaseSurface, 1, swModel);
  //swModel->SelectByName(0, bBaseSurface);
  swModel->InsertSketch2(VARIANT_TRUE); //Enter sketch mode
  swSketchMgr->CreateCircleByRadius(Para1.x,Para1.y,Para1.z,
      nPara2, &swSktSeg);
  swModel->ClearSelection2(VARIANT_TRUE);
  swModel->InsertSketch2(VARIANT_TRUE);

  if (!swSktSeg)
  {
    printf("Func-Extrusion:Failed to create circle");
    return L"";
  }
  swSktSeg->Select4(VARIANT_FALSE, swSelData, &bRet);
  swModel->get_FeatureManager(&swFeatMgr);          // get
      feature manager for this sketch
```

```
    swFeatMgr->FeatureExtrusion2(1,0,0,swEndCondBlind,
        swEndCondBlind,nPara3,0,0,0,0,\
      0,0.0,0.0,0,0,0,0,1,0,1,0,0.0,0,&swFeat); // extrude
  }

  if (swFeat) swFeat->get_Name(&bFeatName);

  // Clean
  swSketchMgr.Release();
  swSketch.Release();
  swFeat.Release();
  swFeatMgr.Release();
  swSktSeg.Release();
  swSelData.Release();

  return bFeatName;
}

// Example function: change the depth of a extrusion feature
int SWCore::SetExtrusionDepth(CComBSTR bFeatName, double
    nLength, IModelDoc2* swModel)
{
  CComPtr<ISelectionMgr>              swSelMgr;
  CComPtr<IDispatch>                  swDispatch;
  CComPtr<IFeature>                   swFeat;
  CComPtr<IExtrudeFeatureData2>       swFeatData;

  HRESULT hr;
  VARIANT_BOOL bRet;

  // Prepare Selection Manager
  hr = swModel->get_SelectionManager(&swDispatch);
```

```
if (hr != S_OK) return 0;

swDispatch->QueryInterface(__uuidof(ISelectionMgr),
    reinterpret_cast<void**>(&swSelMgr));
swDispatch.Release();

// Get the feature by requested feature name
CString strFeatName = CString(bFeatName);
TraverseFeaturesAndOperate(bFeatName, 1, swModel);
hr = swSelMgr->GetSelectedObject5(1, &swDispatch);
if (hr != S_OK) return 0;
swDispatch->QueryInterface(__uuidof(IFeature),
    reinterpret_cast<void**>(&swFeat));
swDispatch.Release();

// Access ExtrusionFeatureData
swFeat->GetDefinition(&swDispatch);
swDispatch->QueryInterface(__uuidof(IExtrudeFeatureData2),
    reinterpret_cast<void**>(&swFeatData));
swDispatch.Release();
hr = swFeatData->AccessSelections(swModel, NULL, &bRet);
if (hr != S_OK || bRet == VARIANT_FALSE) return 0;

// Modify the depth
hr = swFeatData->SetDepth(VARIANT_TRUE, nLength);
if (hr != S_OK) return 0;

hr = swFeat->ModifyDefinition(swFeatData, swModel, NULL, &
    bRet);
if (hr != S_OK || bRet == VARIANT_FALSE)
{
  swFeatData->ReleaseSelectionAccess();
```

```
    return 0;
  }

  // Clean
  swSelMgr.Release();
  swFeat.Release();
  swFeatData.Release();

  return 1;
}

// Example function: General extrusion cut
BSTR SWCore::ExtrusionCut(int nType, Point3 Para1, double
    nPara2, double nPara3, BSTR bBaseSurface, IModelDoc2*
    swModel)
{
  CComPtr<IFeatureManager>          swFeatMgr;
  CComPtr<IFeature>                 swFeat;
  CComPtr<ISketchSegment>           swSktSeg;
  CComPtr<ISketchManager>           swSketchMgr;
  CComPtr<ISketch>                  swSketch;
  CComPtr<ISelectData>              swSelData;

  BSTR bFeatName = L"";
  VARIANT_BOOL bRet;

  // Prepare sketch manager
  swModel->get_SketchManager(&swSketchMgr);

  // Create a cut feature
  swModel->SelectByID(bBaseSurface,L"FACE",0,0,0,&m_bRetVal);
                                                  //pick a face
```

```
//TraverseFeaturesAndOperate(bBaseSurface, 1);
                                                    // Pick a face
//swModel->SelectByID(L"",L"FACE",0,0,0,&m_bRetVal);
swModel->InsertSketch2(VARIANT_TRUE);
                                                                //
    Enter sketch mode

if (nType == 1) // Create a circle for the cut
{
  swSketchMgr->CreateCircleByRadius(Para1.x,Para1.y,Para1.z,
      nPara2, &swSktSeg);
  swModel->ClearSelection2(VARIANT_TRUE);
  swModel->InsertSketch2(VARIANT_TRUE);
}

if (!swSktSeg)
{
  printf("Func-ExtrusionCut:Failed to create circle");
  return L"";
}

swSktSeg->Select4(VARIANT_FALSE, swSelData, &bRet);
//swModel->SelectByID(L"",L"SKETCHSEGMENT",0.5,0,0, &
    m_bRetVal);                          // Select the sketch
swModel->get_FeatureManager(&swFeatMgr);
                                                        // Get
    feature manager for this sketch
swFeatMgr->FeatureCut3(-1,0,0,swEndCondThroughAll,
    swEndCondBlind, nPara3,\
  0.0,0,0,0,0,0.0,0.0,0,0,0,0,0,1,1,0,0,0,swStartSketchPlane
      ,0,0,&swFeat); //cut
```

```
  if (swFeat) swFeat->get_Name(&bFeatName);

  // Clean
  swSketchMgr.Release();
  swSketch.Release();
  swFeat.Release();
  swFeatMgr.Release();
  swSktSeg.Release();
  swSelData.Release();

  return bFeatName;
}

// Example function: Save to a file in STEP format
bool SWCore::SaveToSTEP(BSTR bPath) // SW API not working,
    moved this function to VB
{
  BSTR bName;
  long nErr, nWarn;
  m_swModel1->GetTitle(&bName); // Get the part name
  CComPtr<IModelDoc2> swModel; // the activated doc
  CComPtr<IDispatch> swDispatch;
  m_swApp->ActivateDoc3(bName, false, 1, &nErr, &swDispatch);
  //swDispatch.Release();
  //m_swApp->get_ActiveDoc(&swDispatch);
  swDispatch->QueryInterface(IID_IModelDoc2,reinterpret_cast<
      void**>(&swModel));

  // Start to save
  VARIANT_BOOL bRet;
  BSTR file = L"C:\\Users\\Cyrus\\Desktop\\Part.STEP";
  CComPtr<IModelDocExtension> swDocExt;
```

```
    swModel->get_Extension(&swDocExt);

  swDocExt->SaveAs(file, swSaveAsStandardDrawing,
      swSaveAsOptions_Silent, NULL, &nErr, &nWarn, &bRet);

  swModel.Release();
  swDispatch.Release();
  swDocExt.Release();
  return true;
}
```

Appendix C

The Job Packet of the Abaqus Bending Simulation

C.1 File List

A simulation job packet is created for an individual design instance, which contains three files:

- *RunFEA.py*; A copy of the Abaqus Python script. The source code is provided in section C.3.
- *CADModel.STEP*: The geometry of the biopsy needle cannula exported by DBD through the CAD interface.
- *command.txt*: A text file that includes command lines that will be executed in the Linux-based HPC system. An example is provided in section C.2.

C.2 Linux Command Lines

A sample text file is shown below:

```
# command.txt
module load abaqus
```

```
abaqus cae noGUI=RunFEA.py -- "<Part><ModelPath>/project/
    simdevices/Biopsy/Run1/CADModel.STEP</ModelPath><Material><
    Name>StainlessSteel</Name><Elasticity>200000</Elasticity><PR
    >0.25</PR></Material><Mesh><Size>2</Size></Mesh><InnerRadius
    >2.531</InnerRadius><OuterRadius>2.581</OuterRadius><BCs><
    LoadPos><x>2.581</x><y>0</y><z>0</z></LoadPos><Load><x>2</x
    ><y>0</y><z>0</z></Load></BCs></Part>"
```

The first command line initiates the Abaqus module in a Linux system. The second command line executes *RunFEA.py* with a XML string as an input argument. The XML string contains all the required input variables, including a path to the STEP file, material properties, meshing, geometric information, load and boundary conditions.

C.3 Abaqus Python Script

The Abaqus Python script takes a XML string as input so that multiple input variables can be combined in one input argument. In the main entry of the code, the string is parsed to obtain all required input variables. Next the function *RunSimulation* is called to create the FEA model and run the simulation. Complete source code is provided below:

```
from abaqus import *
from abaqusConstants import *
from odbAccess import *
from sys import argv,exit
from xml.etree.ElementTree import *
import __main__

import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
```

```
import step
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import os

###
def RunSimulation(MyPath,MyMaterial,MyMatE,MyMatPR,x,y,z,Fx,Fy,
    Fz,MeshSize,IR,OR):
# Print initial conditions
    print x
    print y
    print z
    print Fx
    print Fy
    print Fz
# Other predefined variables
    MyModel = 'Model-1'
    MyPart = 'Cannula'
    MySection = 'Section-1'
    MyInstance = MyPart + '-1'
    MyJob = 'Biopsy_Sim' + '-1'

###
# Import model
```

```
    step = mdb.openStep(MyPath, scaleFromFile=OFF)
    mdb.models['Model-1'].PartFromGeometryFile(name=MyPart,
        geometryFile=step,
         combine=False, stitchAfterCombine=False, dimensionality
             =THREE_D,
         type=DEFORMABLE_BODY)

###
# Set up materials and create a section and assign it
    myPart = mdb.models[MyModel].parts[MyPart]
    myCells = myPart.cells

    mdb.models[MyModel].Material(name=MyMaterial)
    mdb.models[MyModel].materials[MyMaterial].Elastic(table=((
        MyMatE, MyMatPR), ))
    mdb.models[MyModel].HomogeneousSolidSection(name=MySection,
        material=MyMaterial, thickness=None)
    cells = myCells.getSequenceFromMask(mask=('[#1 ]', ), )
    myRegion = regionToolset.Region(cells=cells)
    myPart.SectionAssignment(region=myRegion, sectionName=
        MySection, offset=0.0, offsetType=MIDDLE_SURFACE,
        offsetField='', thicknessAssignment=FROM_SECTION)

###
# Assembly- creat a instance
    myAsm = mdb.models[MyModel].rootAssembly
    myInstance = myAsm.Instance(name=MyInstance, part=myPart,
        dependent=ON)

###
# Create the initial condition
    myFaces = myInstance.faces
```

```
    print IR
    print OR
    proximalFace = myFaces.findAt(((0.5*(IR+OR),150.0,0),))
    myAsm.Surface(side1Faces=proximalFace, name='Fixed')
    myRegion = regionToolset.Region(faces=proximalFace) # the
        region of proximal face
    mdb.models[MyModel].EncastreBC(name='Fixed', createStepName
        ='Initial', region=myRegion, localCsys=None)

###
# Create datum point and partition the point for load set up
    datumP = myPart.DatumPointByCoordinate(coords=(x,y,z))
    print datumP.name
    datumIndex = datumP.name[-1]
    print datumIndex
    print int(datumIndex)

###
# Setup the concentrated load
    myAsm.regenerate()
    mdb.models[MyModel].StaticStep(name='Step-1', previous='
        Initial')
    e1, d1 = myPart.edges, myPart.datums
    myPart.PartitionEdgeByPoint(edge=e1.findAt(coordinates=(x,y
        ,z)),point=d1[datumP.id])
    v1 = myAsm.instances['Cannula-1'].vertices
    verts1 = v1.findAt(((x,y,z), ))
    region = regionToolset.Region(vertices=verts1)
    mdb.models['Model-1'].ConcentratedForce(name='Load-1',
        createStepName='Step-1',
         region=region, cf1=-Fx, cf2=Fy, cf3=Fz,
            distributionType=UNIFORM,
```

```
        field='', localCsys=None)

###
# Mesh
    c1 = myAsm.instances[MyInstance].cells
    pickedRegions = c1.getSequenceFromMask(mask=('[#1 ]', ), )
    myPart.setMeshControls(regions=pickedRegions, elemShape=TET
        , technique=FREE)
    elemType1 = mesh.ElemType(elemCode=C3D20R)
    elemType2 = mesh.ElemType(elemCode=C3D15)
    elemType3 = mesh.ElemType(elemCode=C3D10)
    cells = c1.getSequenceFromMask(mask=('[#1 ]', ), )
    pickedRegions =(cells, )
    myPart.setElementType(regions=pickedRegions, elemTypes=(
        elemType1, elemType2, elemType3))
    partInstances =(myAsm.instances[MyInstance], )
    myAsm.generateMesh(regions=partInstances)
    myPart.seedPart(deviationFactor=0.1, minSizeFactor=0.1,
        size=MeshSize)
    myPart.generateMesh()

###
# Create a job and submit it
    mdb.Job(name=MyJob, model=MyModel, description='',
    type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0,
        queue=None, memory=90,
    memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
        explicitPrecision=SINGLE,
    historyPrint=OFF, userSubroutine='', scratch='',
        multiprocessingMode=DEFAULT, numCpus=1)
    CurrentJob = mdb.jobs[MyJob]
    CurrentJob.submit(consistencyChecking=OFF)
```

```
    CurrentJob.waitForCompletion()


###
# Main entry of the code
if __name__ == '__main__':

    argList = argv
    argc = len(argList)

    #Sample input (XML format)
    #<Part>\
    #<ModelPath>CADModel.STEP</ModelPath>\
    #<Material>\
    #<Name>StainlessSteel</Name>\
    #<Elasticity>200000</Elasticity>\
    #<PR>0.25</PR>\
    #</Material>\
    #<Mesh><Size>5</Size></Mesh>\
    #<InnerRadius>2.531</InnerRadius>\
    #<OuterRadius>2.581</OuterRadius>\
    #<BCs>\
    #<LoadPos><x>2.581</x><y>0.00</y><z>0.00</z></LoadPos>\
    #<Load><x>2</x><y>0</y><z>0</z></Load>\
    #</BCs>\
    #</Part>

    MyPart = fromstring(sys.argv[-1]) # Read arguments into a
        XML element

###
# Parse model info.
```

```
    MyPath = MyPart.find("ModelPath")
    print "Model Path: " + MyPath.text
###
# Parse material data
    MyMatName = MyPart.find("Material/Name")
    MyElast = MyPart.find("Material/Elasticity")
    MyPR =  MyPart.find("Material/PR")
    print "Material---"
    print "Name: " + MyMatName.text
    print "Elasticity: " + MyElast.text
    print "Poisson ratio: " + MyPR.text

###
# Parse Mesh Info
    MeshSize = MyPart.find("Mesh/Size")

###
# Parse Radii
    InnerRadius = MyPart.find("InnerRadius")
    OuterRadius = MyPart.find("OuterRadius")

###
# Parse BC data
    MyLoadX = MyPart.find("BCs/Load/x")
    MyLoadY = MyPart.find("BCs/Load/y")
    MyLoadZ = MyPart.find("BCs/Load/z")
    print "Load---\n"
    print "Fx: " + MyLoadX.text
    print "Fy: " + MyLoadY.text
    print "Fz: " + MyLoadZ.text

    MyLoadPosX = MyPart.find("BCs/LoadPos/x")
```

```
    MyLoadPosY = MyPart.find("BCs/LoadPos/y")
    MyLoadPosZ = MyPart.find("BCs/LoadPos/z")
    print "Position---\n"
    print "x: " + MyLoadPosX.text
    print "y: " + MyLoadPosY.text
    print "z: " + MyLoadPosZ.text

    RunSimulation(MyPath.text,MyMatName.text,float(MyElast.text
        ),float(MyPR.text),\
float(MyLoadPosX.text),float(MyLoadPosY.text),float(MyLoadPosZ.
    text),float(MyLoadX.text),float(MyLoadY.text),float(MyLoadZ.
    text), float(MeshSize.text),float(InnerRadius.text),float(
    OuterRadius.text))
```

Appendix D

The Abaqus-NetCDF Interface

D.1 Description

NetCDF is a open-source, cross-platform data format for creating scientific data. A set of software libraries is provided by the NetCDF to efficiently access the data content.

In this dissertation, a customized NetCDF data hierarchy is developed for containing the Abaqus simulation output data, as seen in Fig. D.1. An Abaqus-Python program is developed to export data fields from the Abaqus output data file (.odb) to a NetCDF file (.nc). The source code of the Abaqus-Python program is included in section D.2.

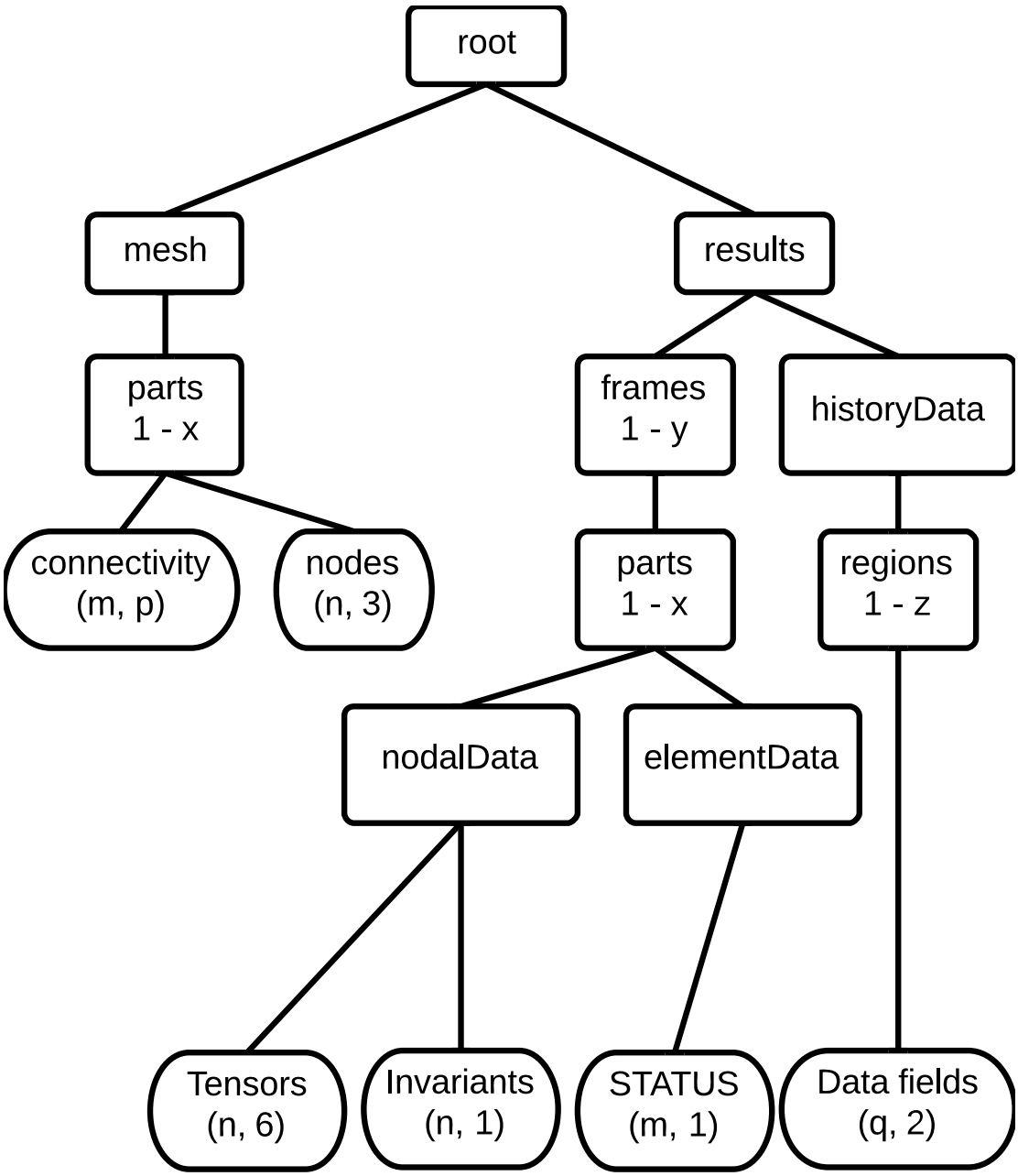


Figure D.1: Customized NetCDF data hierarchy

D.2 The Abaqus-Python Program

This section provides the source code of the Abaqus-Python Program. Only the essential parts of the source code is provided here due to its the extremely long length.

```
# Essential Abaqus and NetCDF libraries to import
from odbAccess import *
from netCDF4 import Dataset

# Open an Abaqus ODB file and access main data groups
odb = openOdb(odbPath)
asm = odb.rootAssembly
instances = asm.instances

# Create a NetCDF database
root = Dataset(outFile, 'w', format='NETCDF4') # Create a
    netCDF file
root.description = 'ABAQUS Output Database, version: ' + odb.
    jobData.version
root.source = odb.name
root.history = 'Created ' + odb.jobData.creationTime

# Looping through part instances
for inst in instances.keys():
  # Create a group to store mesh data of this instance
  grpInst = grpMesh.createGroup(inst)
  nodeNo = len(instances[inst].nodes)
  # Create a variable named 'nodes' for storing nodal
      coordinate
  grpInst.createDimension('nodeNo', nodeNo)
  grpInst.createDimension('coordinate', 3) # (x,y,z)
  nodes = grpInst.createVariable('nodes', 'f4', ('nodeNo', '
      coordinate'))
```

```
    for n in instances[inst].nodes:
      index = n.label
      nodes[index-1] = n.coordinates



# Iterate simulation steps
for sKey in steps.keys():
  # Iterate time frames in this simualtion step
  for frame in steps[sKey].frames:
    print 'Frame Step Time : ', frame.frameValue
    # Examples of reading field data
    # U: Displacement:
    field = 'U'
    set = frame.fieldOutputs[field].getSubset(position=NODAL,
        region=odb.rootAssembly.instances[inst])
    # S: Stress tensor:
    field = 'S'
    set = frame.fieldOutputs[field].getSubset(position=
       ELEMENT_NODAL, region=odb.rootAssembly.instances[inst])
    # STATUS (for element removal):
    field = 'STATUS'
    set = frame.fieldOutputs[field].getSubset(position=
       WHOLE_ELEMENT, region=odb.rootAssembly.instances[inst])
    # YPLUS (for CFD)
    field == 'YPLUS': # This field only contains values on the
        surface elements
    set = frame.fieldOutputs[field].getSubset(position=
       ELEMENT_FACE, region=odb.rootAssembly.instances[inst])
    # Other scalar fields
    set = frame.fieldOutputs[field].getSubset(position=NODAL,
        region=odb.rootAssembly.instances[inst])
```

```
# If this is a tensor (stress, strain)
# Process tensor components (ex: S11, S12, etc)
for label in set.componentLabels:
  subset = set.getScalarField(componentLabel=label)
  data = []
  for value in subset.values:
    data.append((value.nodeLabel, value.data))

# Get invariants (ex: von Mises)
for var in set.validInvariants:
  subset = set.getScalarField(invariant=var)
  data = []
  for value in subset.values:
    data.append((value.nodeLabel, value.elementLabel, value.
        data))

# If this is a scalar field
data = []
for value in set.values:
  data.append((value.nodeLabel, value.data))

# If this is history request output field (ex: reaction
   force, moment)
# grpHistData = grpStep.createGroup('historyData')
desiredHistRegions = ['Node CUTTER_ROUNDED-1.13057']
desiredHistFields = ['RF3', 'RM3']
histRegionKeys = []
for key in steps[sKey].historyRegions.keys():
  if any(key == str for str in desiredHistRegions):
    histRegionKeys.append(key)
print 'History regions to be processed : ', histRegionKeys
```

```
    #Begin reading regional data
    for regionName in histRegionKeys:
      print 'Process history region : ', regionName
      grpHistRegion = grpHistData.createGroup(regionName)
      histRegionData = steps[sKey].historyRegions[regionName]
      histFieldKeys = []
      print 'Available history output fields : ',
          histRegionData.historyOutputs.keys()
      for key in histRegionData.historyOutputs.keys():
        if any(key == str for str in desiredHistFields):
          histFieldKeys.append(key)
      print 'History output fields to be processed : ',
          histFieldKeys

      ##
      # Begin reading history field output
      for fieldName in histFieldKeys:
        print 'Process history data field : ', fieldName
        data = readHistoryOutput(odb.steps[sKey], regionName,
            fieldName)
        dimensionName = 'timeSteps'
        if dimensionName not in grpHistRegion.dimensions:
          grpHistRegion.createDimension(dimensionName, len(data
              ))
          grpHistRegion.createDimension('timeData', 2)
        histFieldData = grpHistRegion.createVariable(fieldName,
            'f4', ('timeSteps', 'timeData'))
        histFieldData[:] = data

# Close files
odb.close()
```

```
root.close()
```