

Patient-Specific Vascular Model Construction and Modification for Blood Flow Simulation
and Analysis

by

Adam Robert Updegrave

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering-Mechanical Engineering

and the Designated Emphasis

in

Computational and Data Science and Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Shawn Shadden, Chair

Professor Grace O'Connell

Professor Per-Olof Persson

Summer 2018

Patient-Specific Vascular Model Construction and Modification for Blood Flow Simulation
and Analysis

Copyright 2018
by
Adam Robert Updegrave

Abstract

Patient-Specific Vascular Model Construction and Modification for Blood Flow Simulation
and Analysis

by

Adam Robert Updegrove

Doctor of Philosophy in Engineering-Mechanical Engineering

and the Designated Emphasis

in

Computational and Data Science and Engineering

University of California, Berkeley

Professor Shawn Shadden, Chair

Cardiovascular disease has remained the leading cause of death worldwide for the past 15 years, and organizations such as the American Heart Association (AHA) and the National Institute for Health (NIH) spend hundreds of millions of U.S. dollars annually to investigate heart disease and stroke. Local characteristics of blood flow in the heart and the rest of the cardiovascular system provide important information in both understanding progression of and diagnosis of cardiovascular diseases. Unfortunately, current medical imaging techniques cannot provide data with high enough temporal and spatial resolution to extract meaningful and accurate research conclusions. Thus, many researchers investigate cardiovascular diseases using a patient-specific blood flow simulation framework. In this framework, a patient's geometry is constructed on a computer from medical image data, and a numerical simulation, such as finite element analysis (FEA), is used to provide very high detail information. Typically, the most time consuming step and also one of the most crucial steps in this pipeline is constructing the geometry of interest from the image data. In addition, many of the tools to create an image-based model are commercial, not readily available, or dispersed amongst a variety of software packages. This dissertation discusses two main avenues of research: (1) the development of unique, customized, and open-source tools for vascular model construction and meshing for FEA and (2) the investigation into and the creation of novel model construction methods for an alternative of FEA called isogeometric analysis (IGA). All of the tools developed were implemented using open-source tools such as the Visualization Toolkit (VTK) and have been implemented into the software framework SimVascular. In addition, many of the methods developed were tested for applicability and

robustness on the open-source vascular model repository, which is a large database of over 100 vascular models provided by the Open Source Medical Software Corporation (OSMSC).

To my family and friends

Contents

Contents	ii
List of Figures	iv
List of Tables	vii
0.1 Introduction	1
1 SimVascular	4
1.1 Introduction	4
1.2 Pipeline and Architecture	5
1.3 Image Segmentation	9
1.4 Model Creation	12
1.5 Meshing	14
1.6 Simulation	14
1.7 Case Studies and Validation	19
1.8 Discussion	22
2 Vascular Modeling for Finite Element Analysis	26
2.1 Introduction	26
2.2 Solid Model Representations	27
2.3 Discrete Modeling	28
2.4 Parametric Modeling	54
2.5 Discussion	62
3 Vascular Meshing for Finite Element Analysis	65
3.1 Introduction	65
3.2 Uniform Isotropic Meshing	66
3.3 Boundary Layer Meshing	66
3.4 Spherical Mesh Refinement	67
3.5 Local Mesh Size Application	67
3.6 Multi-Domain Meshing	69
3.7 Meshing with Holes	69

3.8	Isotropic Mesh Adaption	70
3.9	Comparison	71
3.10	Discussion	73
4	Vascular Modeling for Isogeometric Analysis	76
4.1	Introduction	76
4.2	Isogeometric Analysis	77
4.3	Analysis Suitable Representations	80
4.4	Discussion	85
5	Vascular Meshing for Isogeometric Analysis	87
5.1	Introduction	87
5.2	Centerline Extraction	90
5.3	Centerline Processing	94
5.4	Graph Simplification	97
5.5	Polycube Generation	100
5.6	Graph Simplification	102
5.7	Surface Group Decomposition	105
5.8	Surface Patch Decomposition	106
5.9	Conformal Mapping	110
5.10	Volumetric Parameterization	112
5.11	Volumetric NURBS Creation	114
5.12	Results	115
5.13	Discussion	122
6	Summary	123
	Bibliography	125

List of Figures

1.1	SimVascular Pipeline	6
1.2	Inheritance Diagram	7
1.3	SimVascular 2.0 GUI	8
1.4	SimVascular 3.0 GUI	9
1.5	2D Segmentation Method	10
1.6	Level Set and Threshold Segmentation	11
1.7	Model Repair Operations	13
1.8	Mesh Examples	15
1.9	Boundary Conditions	17
1.10	Validation in Aortic Coartation	20
1.11	Repository Examples	21
1.12	Repository Content	22
1.13	Coronary Arteries Simulation	23
1.14	Pulmonary Arteries Simulation	23
1.15	Aortofemoral Arteries Simulation	24
1.16	Coronary Artery Bypass Graft Simulation	25
2.1	Constructive Solid Geometry and Boundary Representations	27
2.2	Oriented Bounding Box Trees	32
2.3	Line Line Intersection	33
2.4	Triangle Triangle Intersection	33
2.5	Intersection Line Points	34
2.6	Split Cell Loops	35
2.7	Intersection Loop Types	36
2.8	Cell Orientations	38
2.9	Sub-Surfaces of a Sphere Cylinder Boolean	38
2.10	Boolean between Aorta and Iliac Branch	39
2.11	Open Intersection Loops	40
2.12	Soft Closed Intersection Loops	41
2.13	Performance of Boolean Implementations	42
2.14	Vascular Network Union	43
2.15	Triangulated Surface Aspect Ratios	44

2.16	Triangulated Surface Normals Compare	45
2.17	Cell Selection Types	45
2.18	Constrained Smoothing of Aorta	49
2.19	Constrained Smoothing Example	49
2.20	Percent of Original Volume	50
2.21	Triangulated Surface Local Decimation	50
2.22	Butterfly Subdivision Scheme	51
2.23	Loop Subdivision Scheme	51
2.24	Triangulated Surface Local Subdivision	52
2.25	Remeshing Edge Operations	52
2.26	Triangulated Surface Uniform Remesh	53
2.27	Triangulated Surface Remesh with Refinement	54
2.28	Power Basis Curve	55
2.29	B-spline Basis Functions	57
2.30	NURBS with Control Point Grid	58
2.31	B-spline Curve Degree Elevation	62
2.32	B-spline Curve Knot Insertion	63
2.33	B-spline Curve Bezier Extraction	64
3.1	Uniform Isotropic Mesh	66
3.2	Boundary Layer Mesh	67
3.3	Sphere Refinement Mesh	68
3.4	Radius Based Mesh	68
3.5	Multi-Domain Meshing	69
3.6	Meshing with Holes	70
3.7	Adapted Mesh Example	72
3.8	Internal Adapted Mesh Example	73
3.9	MeshSim and TetGen Mesh Comparison	74
3.10	MeshSim and TetGen Simulation Comparison	74
4.1	Design and Analysis Timeline	82
5.1	Volumetric NURBS creation pipeline	90
5.2	Cell-thinning Procedure	92
5.3	Centerline Bifurcation and End Points	93
5.4	Centerline Merging	95
5.5	Centerline Coordinate System	97
5.6	Centerline Coordinate System Comparison	98
5.7	Centerline Coordinate System Update	98
5.8	Cuboid Types	101
5.9	Bifurcation Types	103
5.10	Clustering with Cell Normals and Positions	109

5.11	Surface and Polycube Matching	110
5.12	Angles for Harmonic Edge Weights	112
5.13	Input Output Diagram	115
5.14	Final NURBS Comparison	116
5.15	OSMSC Repository Examples	118
5.16	OSMSC Repository Content	118
5.17	Volumetric NURBS Creation - Pulmonary Example	119
5.18	Volumetric NURBS Creation - Aortofemoral Example	119
5.19	Volumetric NURBS Creation - Cerebrovascular Example	120
5.20	Volumetric NURBS Creation - Teddy Bear Example	120
5.21	Volumetric NURBS Creation - Bird Example	120
5.22	Volumetric NURBS Creation - Pliers Example	121
5.23	Volumetric NURBS Creation - Ant Example	121
6.1	SimVascular Global Map	124

List of Tables

3.1	Comparison of MeshSim (A.) and TetGen (B.)	73
4.1	Comparison of IGA Geometric Representations	84
5.1	Inputs and Outputs for Centerline Extraction	91
5.2	Inputs and Outputs for Centerline Processing	94
5.3	Inputs and Outputs for Graph Simplification	99
5.4	Inputs and Outputs for Polycube Generation	102
5.5	Inputs and Outputs for Graph Simplification	104
5.6	Inputs and Outputs for Surface Group Decomposition	105
5.7	Inputs and Outputs for Surface Patch Decomposition	107
5.8	Inputs and Outputs for Conformal Mapping	111
5.9	Inputs and Outputs for Volumetric Parameterization	113
5.10	Inputs and Outputs for Volumetric NURBS Creation	114
5.11	NURBS Conversion Error Metrics	117

Acknowledgments

I would first like to thank my PhD advisor, Dr. Shawn Shadden. He has been a continual source of knowledge, support, and kindness throughout my PhD. He has inspired and pushed me to become a better researcher, thinker, and person. I would also like to thank both Dr. Alison Marsden and Dr. Nathan Wilson for their continued support. Throughout my PhD, Alison has been a fabulous role model and mentor. She is always incredibly thoughtful and kind, and I am honored to have been able to work on a project in which she was involved. Nathan has also been an amazing mentor throughout my PhD. From day one, I was learning from Nathan, and although we may disagree on the best coding languages, I will always cherish Nathan's incredible insight, thought process, and comedic relief. Thank you for being the best set of advisors anyone could ask for.

I have been unfathomably lucky to have the best friends in the world who have supported me throughout my PhD and life. To the boys of 2240 (Tyler Iosue, Jake Tonkel, William Keddy-Hector, Jim Mulhern, Michael Bergamaschi, Ben Batista, Ryan Kelly, and Zach Fane), I think about the great times we had together almost every day, and I look forward to every time I get to see you all. To the monop crew (Shepherd Darquea, Scott Wandel, Tyler Davies, Greg Saunders, Christoph Moore, and Alex Updegrove), thanks for teaching me the value of having fun and living life to the fullest. To the NBYC (Patrick Lu and Pratik Sachdeva), I'm so happy that I have been able to cross paths with such an incredible institution. To Nick Schlager, my friend since the beginning, thanks for always being there. Our lizard catching days may be over, but our friendship is not. To Chris Fuller, thank you for always lending me an ear and listening to me, no matter what the topic. And to Christoph Moore, thanks for supporting me through the whole PhD. You made everything brighter, and we shared many a good time together. Lastly, I would like to thank the Shadden Lab and the Carotid Kids. It has been so great getting to know such wonderful people while at Berkeley, and I'll always remember the great times we had in lab as well as the countless Etcheverry Cups we've won.

And finally thanks so much to my family. My madre and padre, Diane and Kevin, there aren't enough words to describe what you mean to me. You are loving, caring, supportive, helpful, selfless, and just all around amazing. There is no way I could have done any of this without you. My brothers, Alex and Andrew, and my sister, Nicole, are also truly incredible. You are also the best siblings anyone could have asked for. I cherish every moment I get with you guys, and I look forward to all the fun we will have in the future.

0.1 Introduction

0.1.1 Motivation

Cardiovascular diseases are the leading cause of death worldwide [1, 2]. Many of these cardiovascular diseases are caused or affected by characteristics or disruptions of local blood flow. Unfortunately, current imaging techniques cannot provide blood flow information at a high enough resolution to be useful for in depth cardiovascular disease investigation. Thus, patient-specific hemodynamic or blood flow simulation was pioneered in the late 1990's [3, 4] to investigate the progression of cardiovascular diseases. Since then, patient-specific blood flow simulation has entered the clinical world [5], and also proven to be an effective method for studying a variety of cardiovascular diseases in the academic world [6, 7, 8, 9, 10, 11, 12, 13, 14]. In patient-specific blood flow modeling, a patient-specific model is "segmented" or constructed from medical image data, such as magnetic resonance imaging (MRI) or computed tomography (CT). This model is prepared for simulation historically through a meshing procedure that makes it suitable for analysis or blood flow simulation. Blood flow analysis is then carried out using numerical techniques such as finite element analysis (FEA), finite volume (FV) analysis, or isogeometric analysis (IGA).

Despite its effectiveness as both a research tool and as a supplement to clinical procedures, there are still many inefficiencies and bottlenecks in the blood flow simulation pipeline. It is still a major challenge to accurately and efficiently create vascular models from medical image data. This is due to a combination of challenges including a large collection of cardiovascular diseases, vast differences in patient vascular anatomies, and difficulty in distinguishing blood vessels from other artifacts and anatomical structures in image data. Researchers typically report modeling to be the most time-consuming step of the patient-specific blood flow pipeline. Though it can often take a significant amount of time to run a complex and large simulation, solvers run independently, and this is time that can be utilized elsewhere by the researcher. Thus, patient-specific model construction is a major roadblock in the blood flow simulation pipeline for both academic studies and use in the clinic. It has been proposed that a patient-specific geometry should be constructed in 40 minutes or less to be used for computer-integrated surgery [15]. In addition, the model or geometry is often a main source of error in simulations. Thus, it is important to both reduce the time it takes to create patient-specific models as well as make construction methods more reproducibly accurate.

An additional challenge for researchers looking to investigate a particular cardiovascular disease is the lack of easily available and usable tools to perform both anatomical construction and blood flow analysis. This is due to a variety of reasons:

1. Many software tools for image segmentation, modeling, and analysis are either general tools or intended for a different application. For example, SolidWorks is one of the most powerful computer-aided design (CAD) frameworks in the industry. Though SolidWorks excels at constructing complex mechanical parts and assemblies, a model

cannot be constructed from medical image data through SolidWorks, and it is difficult to import a vascular model and perform meaningful modifications to the geometry.

2. There are very few tools that provide a full pipeline of the vascular modeling workflow. Starting from image analysis and segmentation and ending with blood flow simulation may require a researcher to use up to four or five different software frameworks. It is then a major challenge to develop a procedure to pass data between these interfaces and also learn and develop the skills to use all of these different tools.
3. Powerful software packages are typically commercial, and even with an academic license, it can be expensive to purchase and use. This is a roadblock to researchers who may simply want to evaluate a tool to test its applicability for cardiovascular problems but cannot due to the cost.

To address these issues, the research discussed in this dissertation has been geared around providing usable, effective, and open-source tools for the software framework, SimVascular [16]. SimVascular is currently the only completely open-source software to enable researchers to go all the way from image data to blood flow simulation analysis. In the effort to provide a completely open-source pipeline for cardiovascular simulation, it became evident that there are a lack of complete, robust, and usable modeling and meshing tools, especially in specific to cardiovascular modeling. In this dissertation, SimVascular will be overviewed, and the specific contributions made as part of this doctoral work will be discussed. The algorithms and methods developed as part of this dissertation have been implemented in an open source library called vtkSV. The library includes algorithms for geometric manipulation, Boolean operations, NURBS and parametric modeling, surface decomposition or segmentation, and other miscellaneous and useful operations.

0.1.2 Background

Many software platforms focus on visualizing medical image data for identification of anatomic structures in clinical, academic, and educational settings; Osirix (<https://www.osirix-viewer.com>), Volview (<https://www.kitware.com/volview>), and Slicer (<https://www.slicer.org>) are all examples of softwares focussing on image visualization. Other software platforms, such as ITK-snap (<http://www.itksnap.org>), Seg3D (<http://www.sci.utah.edu/cibc-software/seg3d.html>), and MITK (<http://mitk.org>), focus on obtaining a usable segmentation of structures within medical images for a variety of applications including 3D printing, anatomical studies, and analysis. Though these can often times provide quality representations of the image data, it can be more efficient to work within a platform specializing on the area of interest. For example, VMTK (<http://www.vmtk.org>), CRIMSON (<http://www.crimson.software>), and SimVascular (<http://www.simvascular.org>) all provide tools focussed on segmentation of vasculature from medical image data for the purpose of blood flow simulation. VMTK has a variety of command line python tools for vascular segmentation, model creation, and meshing; however, VMTK is not directly linked to a solver, and an external solver must be

found or purchased for simulation. CRIMSON provides tools for the vascular segmentation, model creation, and meshing pipeline within a graphical user interface (GUI), but the solver is not open source and available for widespread use. SimVascular provides a complete blood flow simulation pipeline with an open-source GUI for vascular model creation and an open-source solver that can be called directly from the GUI, built and executed on a cluster, or run through a gateway interface [17].

SimVascular was originally developed in the lab of Charles Taylor at Stanford University. After significant developments in making the pipeline usable for researchers at Stanford, the software was open sourced in 2007. Despite being open source, multiple components of the software were still commercial, and it was very difficult to obtain and compile the source. With the work presented in this dissertation, significant improvements have been made towards creating a more accessible, robust, and user friendly pipeline.

0.1.3 Outline

A description of the new open-source SimVascular pipeline and the different components of the pipeline will first be provided in Chapter 1. This will help to give a basic understanding of the software as well as establish a context for the remainder of the work described in this dissertation. Following the overview of SimVascular, the contributions for this dissertation for vascular modeling and effective tools in constructing analysis suitable models will be discussed in Chapter 2. Next, meshing procedures and the challenges associated in meshing vascular models will be reviewed in Chapter 3. This will be followed by a small description of IGA along with an analysis of geometric representations that have recently been developed for use with IGA in Chapter 4. I will then describe methods developed during this doctoral work to tackle the complicated task of forming analysis suitable multi-variate spline representations in Chapter 5. In each section, the relevant literature will be reviewed, the developed methods and their applications will be described, and if there are direct results of the methods, those will be presented and discussed.

Chapter 1

SimVascular

Parts of this chapter are published in [16] and in collaboration with *Nathan M. Wilson, Jameson Merkow, Alison L. Marsden, and Shawn C. Shadden*.

1.1 Introduction

Cardiovascular disease is the leading cause of death and disability worldwide. Central to both the causes and consequences of cardiovascular disease are local and regional disruptions in blood flow. The relationships between hemodynamics and cardiovascular diseases are subtle and multifaceted. While qualitative understanding and correlations are well documented [18, 19, 20, 21, 22, 23], precise knowledge of hemodynamic conditions is needed to quantify risk and evaluate mechanisms. Simulation-based methods can provide a powerful framework in this regard. Advanced numerical methods are enabling increasingly realistic representations of cardiovascular physiology. Moreover, because the role of hemodynamics in any disease scenario is highly individualized, medical imaging and clinical data often forms the basis for patient-specific numerical simulations. These simulations can now provide a means to perform patient-specific treatment planning, virtual surgery and design optimization.

Image-based blood flow modeling was pioneered in the late 1990's and early 2000's [3, 4, 24, 25, 26, 27] and, in the years since, has proven to be a powerful tool in basic science and clinical research. Indeed, HeartFlow recently introduced the first FDA-approved simulation-based service for routine clinical evaluation of coronary stenoses [28]. In most image-based modeling applications, 3D angiographic data obtained from computed tomography (CT) or magnetic resonance imaging (MRI) is used to construct a geometric model of a vascular region. Image processing is used to construct a vascular model that is then imported into a computational fluid dynamics (CFD) package to generate a volumetric mesh and numerically simulate blood flow. While numerous image processing software packages exist, most are not designed to generate computer models well-suited for simulation purposes. And while numerous CFD packages exist, most are not designed to accommodate the sophisticated boundary conditions, physiologic models and physics specific to cardiovascular modeling.

The software package SimVascular was originally developed in the lab of Charles Taylor at Stanford University to provide a complete pipeline from medical image data segmentation to patient specific blood flow simulation and analysis. SimVascular provides boundary conditions that achieve physiologic levels of pressure, fluid structure interaction, and a highly accurate and efficient finite element flow solver. The software was released in 2007 and remained the only software package for cardiovascular simulation that includes the entire pipeline from model construction to simulation analysis. However, the need for commercial components and licenses previously hindered new user adoption and prevented complete open source release. Moreover, the infrastructure for continued software development was lacking as well as necessary features for wider use.

To overcome the above barriers, the SimVascular revitalization project was launched in 2013. A major goal of these efforts was the development and integration of open-source alternatives for a truly open-source SimVascular project. In addition, new functionality in nearly all facets of the pipeline has been added to enhance modeling accuracy, usability and efficiency. Examples of recent enhancements include direct 3D segmentation, discrete solid modeling, mesh repair tools, fluid-solid interaction with variable wall properties, closed-loop lumped parameter network modeling, and updates to the graphical user interface (GUI). In addition, case studies, online documentation, CMake compatibility, a user forum, binary packages for all major operating systems, and other infrastructure to support the open-source project have been brought online (www.simvascular.org). As part of the overhaul of SimVascular, two large updates have taken place. The first update included completing the open-source pipeline with the addition of open-source modeling and meshing tools and updates to many parts of the source code. This update culminated in the release of SimVascular 2.0 [16]. The next update involved the development of an entirely new GUI, which produced SimVascular 3.0 [29]. Both GUI architectures will be described briefly, and the general image-based patient-specific blood flow simulation pipeline will be described in detail.

1.2 Pipeline and Architecture

1.2.1 SimVascular Pipeline

The SimVascular pipeline starts with image processing and segmentation and continues all the way through to post-processing of simulation results. Figure 1.1 displays nominal steps of image-based modeling in SimVascular, although alternative and additional steps may be employed. We briefly describe the main steps in the pipeline, and then provide additional details in the subsequent sections.

Paths - The segmentation process typically starts by creating pathlines along the vessels of interest. It is possible to create models using a *lofted 2D* segmentation method (Section 1.3.1) or *direct 3D* segmentation methods (Section 1.3.2). When performing lofted 2D segmentation, the pathlines are used to resample the image data to a cross-sectional “in-

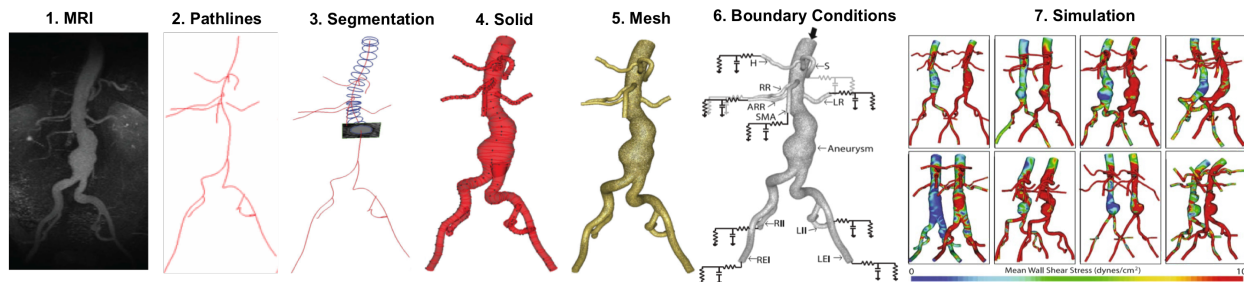


Figure 1.1: The SimVascular pipeline leads the user from visualization of image data to completion of blood flow simulations. Steps 2-4 correspond to the lofted 2D segmentation process. Adapted from [30].

tensity probe” window that can be moved along the vessel’s path (Fig. 1.5a). For direct 3D segmentation methods, paths are not necessary but can be used to help initiate region growing methods.

Segmentation - For the lofted 2D segmentation method, functionality is provided to move along each pathline and create a series of segmentations that delineate the luminal boundaries of the vessel (Fig. 1.5b). Alternatively, for direct 3D segmentation, functionality is provided to position seed surfaces (spheres) that will expand, merge and morph in 3D space to fill in the luminal boundary.

Model - After image segmentation, a solid model can be generated. Following the lofted 2D approach, the series of segmentations are lofted together with splines (Fig. 1.5c). For either the lofted 2D or direct 3D approach, functionality for manipulating the model and identifying faces of the model (e.g., for specifying boundary conditions) is provided. Additionally, surfaces from 2D (Fig. 1.5d) and 3D methods can be combined into a single model using custom boolean operations.

Meshing - A volumetric finite element mesh is created from the geometric model for numerical analysis. SimVascular supports construction of unstructured tetrahedral meshes as well as several advanced meshing features including boundary layer meshing, radius-based meshing, regional refinement and adaptive meshing.

Simulations - Simulation in SimVascular is broken down into three executables in which a presolver, solver and postsolver are used to generate simulation results. There is functionality to assign boundary conditions, material properties, and set parameters for the solver. The svSolver can be run through the GUI; however, it is common for simulation files to be generated on a desktop computer and then copied to a high performance computing (HPC) cluster where the svSolver can be run in parallel.

1.2.2 SimVascular Architecture

SimVascular’s core source code is comprised of two common coding languages: C++ and Fortran. The majority of the source code is written in C++, including code for segmentation, solid modeling, and meshing. The solver, `svSolver`, is the only portion of the source currently written in Fortran. The SimVascular source code is built upon a *repository* in which objects are stored, maintained, and tracked. The repository, which is a large hash table, facilitates memory management across the large scale software platform. The repository stores all data structures as a `cvRepositoryData` object. Figure 1.2 displays a simplified inheritance diagram for the SimVascular data structures. Stemming from the `cvRepositoryData` data structure, there are several objects used within SimVascular’s source code for data representation. These objects include `cvDataObject` (a general subclass of `cvRepositoryData`), `cvSolidModel`, and `cvMeshObject`. These are abstract base classes providing virtual functions for implementation in derived classes. They define a rigid structure for the derived classes that is important for the modularity of SimVascular. SimVascular uses external libraries for multiple Solid Model and Mesh Object classes (Fig. 1.2). Each of these packages are included in a derived class demonstrating SimVascular’s extensibility for new module plugins.

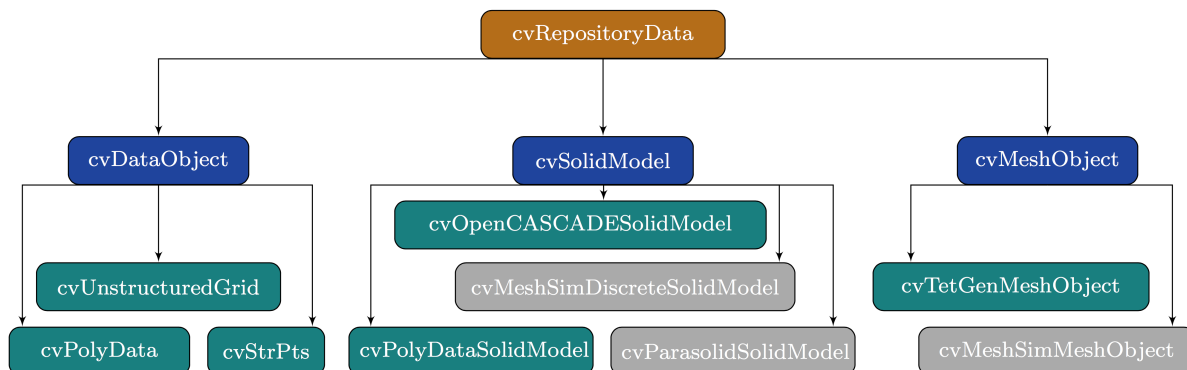


Figure 1.2: Inheritance Diagram of `cvRepositoryData`. Derived classes in aqua are open source while derived classes in grey are commercial and optional.

1.2.3 SimVascular 2.0 GUI

The SimVascular 2.0 GUI is comprised of an interactive display, as well as image and work tabs (Fig. 1.3). The image tabs provide control over how the image data is displayed. This includes functionality such as loading medical image data, point cloud visualization, and volume rendering. The work tabs encapsulate the core functionality of the model construction process. In SimVascular 2.0, the Tcl language is used for C++ bindings (for high level access to core functionality) and as the front end interpreter language. Tcl/Tk is used in

combination for the graphical user interface (GUI). In addition, SimVascular’s core C++ functions are wrapped with Tcl bindings and callable through a Tcl console or interpreter. The Tcl interpreter provides users the ability to use custom scripts, in addition to or in place of the GUI, to access a wide range of the SimVascular functionality and automate repeatable procedures.

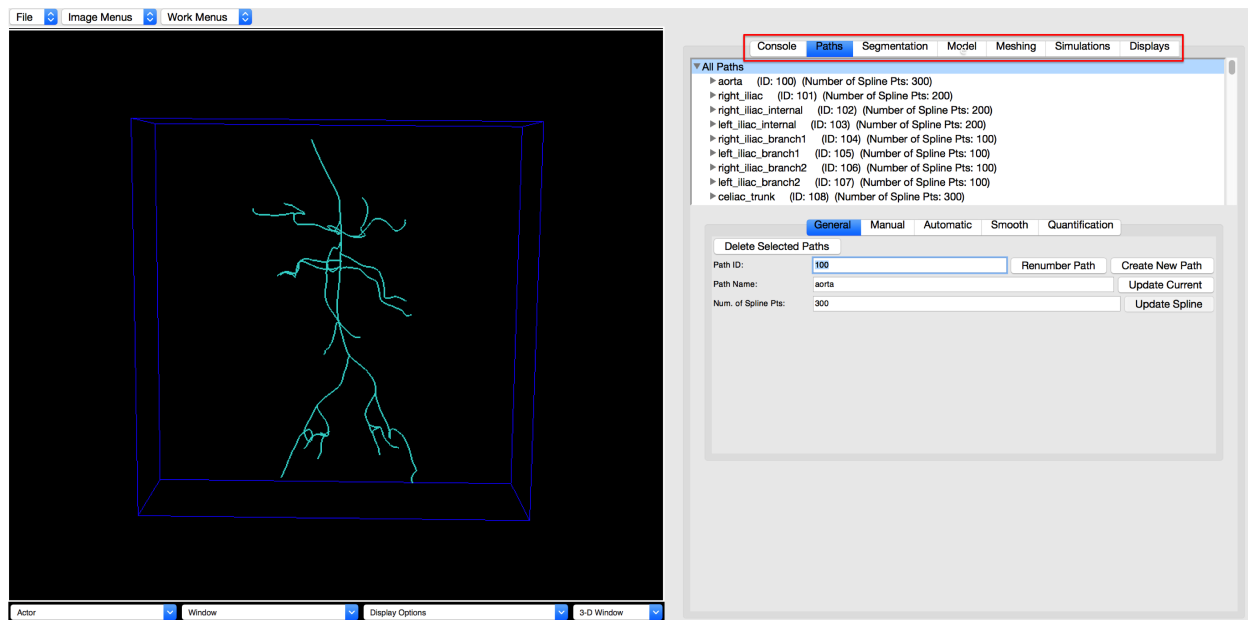


Figure 1.3: The SimVascular pipeline is mirrored in the main work tabs of the GUI of SimVascular 2.0 (enclosed in red box): Paths → Segmentation → Model → Meshing → Simulations.

1.2.4 SimVascular 3.0 GUI

The SimVascular 3.0 GUI was developed in 2016 and also provides functionality for all steps of the SimVascular pipeline, but in a more up-to-date interface using Qt and derived from the Medical Imaging Interaction Toolkit (MITK) (Fig. 1.4). Care has been taken to improve the usability and design of the interface to facilitate the workflow. An interactive data manager allows creation, editing, and removal of all data types within the SimVascular pipeline. Each step of the pipeline also contains its own separate module tab which is activated upon selection of an item of that specific data type in the data manager. In this manner, it is obvious what tools and functions can be performed for each data type. Python bindings have also been created for SimVascular’s core C++ functions to complement the new GUI. For more information on the development and implementation of the SimVascular 3.0 GUI, see [29].

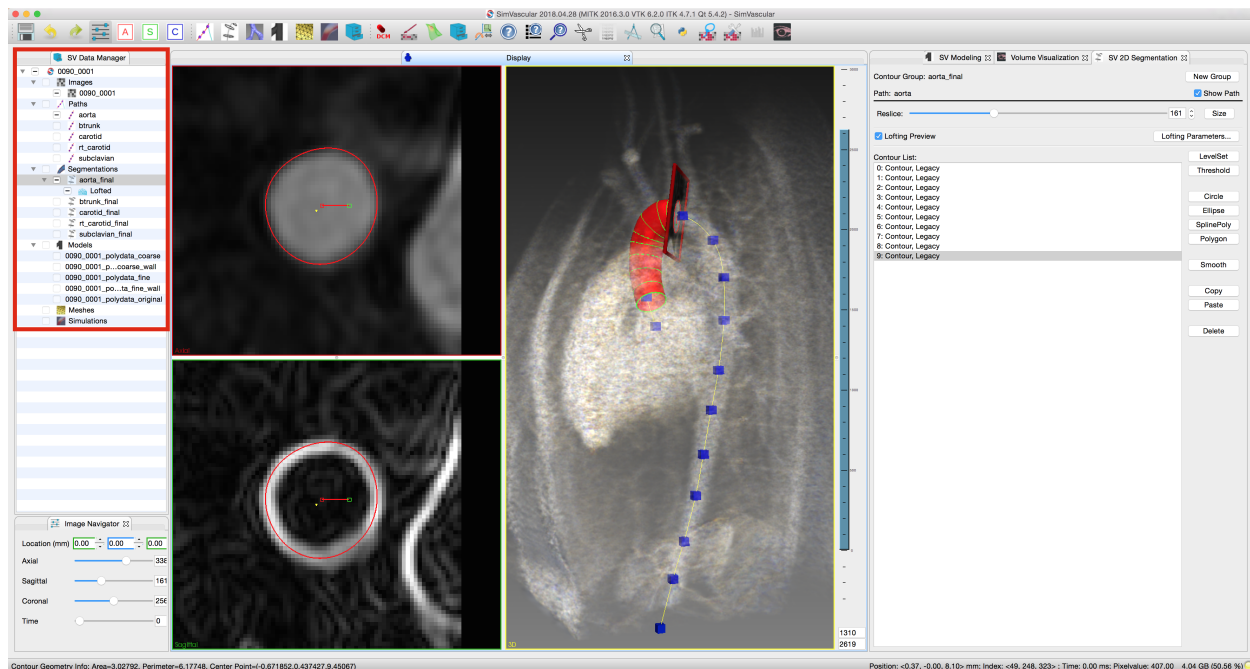


Figure 1.4: The SimVascular pipeline is mirrored in the SimVascular 3.0 Data Manager (enclosed in red box). Images \rightarrow Paths \rightarrow Segmentation \rightarrow Model \rightarrow Meshing \rightarrow Simulations.

1.2.5 SimVascular Maintenance

The source code is maintained as a github repository (<https://www.github.com/SimVascular>). Maintenance and development is enhanced with multiple modern software development tools. CMake is used to build and test functionality of the source, while Travis CI is used for automated building on various versions of Linux with different library versions, and CTest/CDash are used for nightly builds. Stable binary releases of the software are posted on Simtk (<https://www.simtk.org/home/simvascular>). Simtk also currently hosts user forums, email lists, and a bug tracker for SimVascular.

1.3 Image Segmentation

Image data consists of a set of scalar values defined on a structured grid. The scalar values represent an intensity field in 3D space and objects or material within the image are identified by different intensity values or ranges. The first step in image-based modeling is to segment the image data in a region of interest (ROI) to extract the boundary or structure of an object from the intensity field. With SimVascular, the segmentation process is most commonly used to identify the luminal surface of a blood vessel; however other anatomical structures may be similarly segmented and modeled. Extensive research has been conducted in the field

of image segmentation [31, 32, 33, 34] and SimVascular utilizes established techniques that incorporate both 2D and 3D image segmentation techniques.

1.3.1 2D Segmentation Methods

Figure 1.5 displays the steps to generate an individual vessel using the lofted 2D segmentation approach. First, an approximate centerline is generated along the vessel (Fig. 1.5a). Along this path, a series of segmentations is generated by stepping a 2D cross-sectional imaging window along the vessel (Fig. 1.5b). Finally, the segmentations are lofted together to give a surface representing the lumen (Fig. 1.5c). Lofting is performed through generation of spline interpolating functions. To create a vascular network, multiple vessels are created sequentially, and then unioned via a set of boolean operations (Fig. 1.5d).

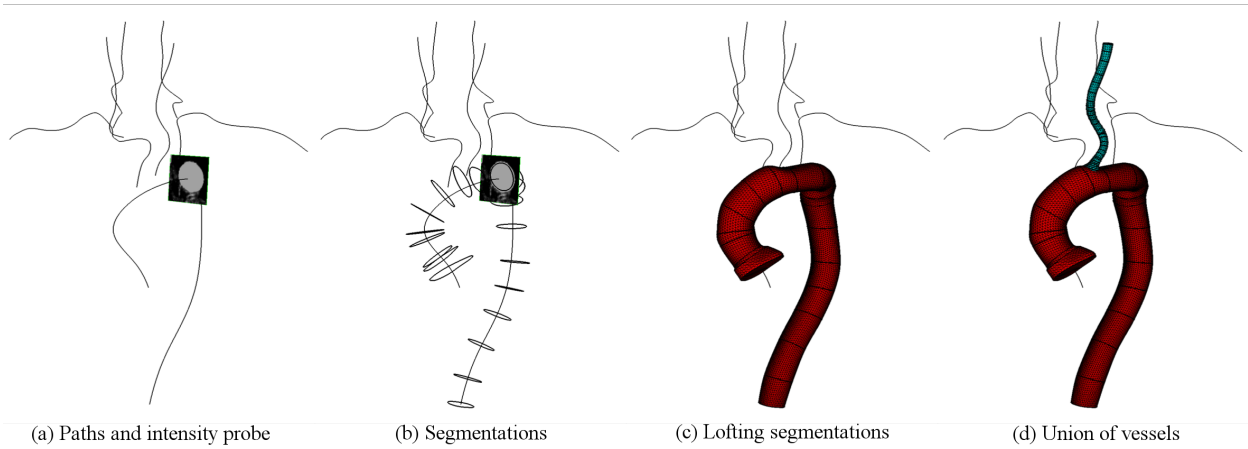


Figure 1.5: Creation of a vascular geometry using the lofted 2D segmentation approach involves moving a cross-sectional image window along each vessel path to create a series of segmentations (b) that are lofted to form each vessel (c). A solid model is generated by the union of individual vessel models (d).

The 2D cross-sectional imaging window (Fig. 1.5a,b) is limited to a region around the path so that peripheral image data does not interfere with the local segmentation of the vessel of interest. There are a variety of methods implemented to segment the lumen, though the two main approaches are based on level set and threshold techniques.

Level Set - A contour is initialized by a seed point (small disk) and grows in the directions of changing intensity values to find the location of sharpest change. This picks out the vessel wall as a complete contour region (Fig. 1.6, left). A pair of windows displaying the image intensity and magnitude of the intensity gradient facilitates segmentation creation and editing. A moving level set front is governed by

$$\phi_t = -v|\nabla\phi| - \nabla g \cdot \nabla\phi, \quad (1.1)$$

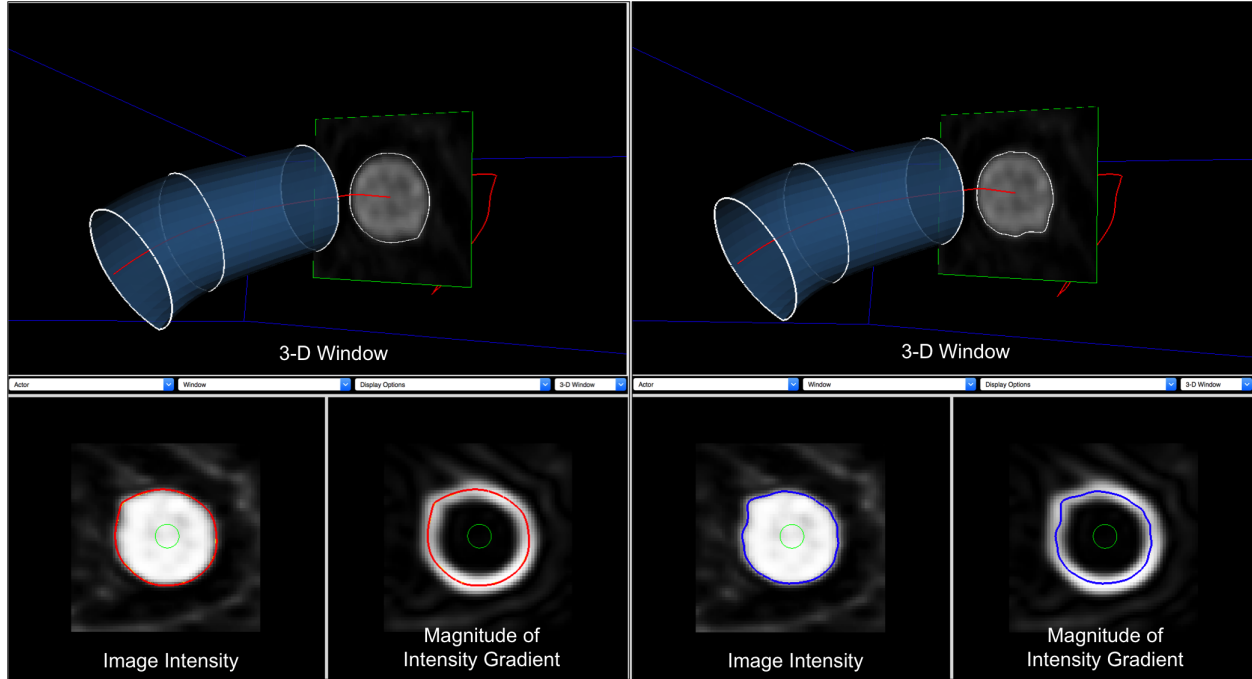


Figure 1.6: Left: A slice along the vessel pathline is segmented using level set segmentation techniques. Right: The same slice is segmented using threshold techniques.

where ϕ_t represents the front, v is the velocity normal to the front, and g is an edge detection function. The velocity term, v , is represented differently for two different stages. In the first stage, the velocity is represented by exponentially decaying functions. In the second stage, the velocity is represented by edge attraction functions. Parameters in the SimVascular GUI correspond to scalars governing the decay in the first stage and attraction in the second stage. For more details, see [32].

Threshold - Image intensity values are assumed to be centered at each pixel. A bilinear interpolation function is then used to create isocurves of a specified threshold value on the image. These isocurves are potential contours of the lumen. A circle of specified radius is centered on the path, and the smallest closed isocurve that completely encapsulates the circle is identified as the lumen boundary (Fig. 1.6, right).

Manual - Manual segmentation is useful for noisy data sets, images with complicated features, or cases where the automated methods fail to converge. Points along the lumen are manually selected by the user and (automatically) connected with a closed spline to represent the geometry.

Analytic - A 2D segmentation is created using a circle or ellipse of user specified dimensions. This can be helpful for ideal geometries, noisy data, or locations where no image data exists.

Segmentations created by any of the above methods can be smoothed post creation.

Also, in some applications multiple segmentations can be created using the same set of level set or threshold parameters, allowing for “batch mode” segmentation. In batch mode, for a specified range along the vessel path, segmentations are automatically generated with specified settings. Segmentations can then be checked and modified as needed using the editing or smoothing tools provided in SimVascular. In cases of appropriate image quality, this can be an efficient way to automatically generate a set of segmentations along each path.

1.3.2 3D Segmentation Methods

Direct 3D segmentation methods are also available in SimVascular, which are useful for segmenting vessel sections that do not lend themselves well to 2D cross section segmentation, such as aneurysms, and vessel junctions. This process begins by placing 3D seed “points” (small spheres) within vascular locations. These act as initial surfaces for active contours and level set algorithms. Seeds can be positioned by manual selection in the 3D window using coordinate position sliders, or along SimVascular pathlines.

In addition to specification from seeds, initial contours from which a 3D surface is grown can be specified via several alternative methods. These methods include initialization from previous level set surfaces, surfaces created through 2D segmentation, and even 3D iso-surfaces of the image data. After positioning seed points or selecting an initial contour surface, a 3D surface is grown using one of two level set algorithms: (1) a Laplacian fast edge grower, or (2) a geodesic smoothing level set. Both level set types are implemented by modifying the terms in Equation (1.1). These level set algorithms propagate segmentation labels through an energy minimization of appearance, curvature, and propagation terms. Appearance features are controlled by modifying the parameters shown in the following equation:

$$E = \frac{1}{1 + (\Delta(I * f) \cdot \kappa)^m}, \quad (1.2)$$

where $*$ is the convolution operator, f represents a Gaussian smoothing kernel, and k and m represent contrast parameters for contrast scaling and proximity.

1.4 Model Creation

The segmentation process results directly, or indirectly, in a *boundary representation* of the blood flow domain (or other physical region of interest). The output of the 3D segmentation process is a triangulated surface that serves as a discrete boundary representation. For the lofted 2D segmentation approach, the segmentations must be lofted to construct a solid model as described below, which can be represented as either a triangulated surface, or parametric (CAD) model. See Chapter 2 for more details on modeling. Additional procedures are often needed to make the solid model compatible with computational modeling. SimVascular supports four different solid modeling approaches: (1) PolyData, (2) OpenCASCADE, (3) Parasolid, and (4) Discrete.

PolyData - The most extensive solid modeling package in SimVascular is PolyData. Combining custom-developed procedures with filters available in VTK (www.vtk.org) and VMTK (www.vmtk.org), the PolyData kernel provides multiple ways to create and manipulate a geometry. The PolyData kernel is first used to generate a model from the ordered 2D segmentations. Splines are formed along the length of the vessel that connect the 2D segmentations, resampled to a specified number of points, and then connected and triangulated to form a complete PolyData surface. Each set of segmentations results in one lofted vessel. Additional lofted vessels are then combined using an ordered Boolean addition. A customized Boolean operation for triangulated surfaces is used for this operation [35]. Other PolyData operations provided in SimVascular include smoothing, blending, decimation, subdivision, remeshing, clipping, deleting cells, and filling holes. Many of these are available as localized operations, which can confine operations to a subset of the model. Selection options include picking a spherical region, using single or multiple faces, identifying the region between two faces (e.g., vessel junctions), or even clicking on individual cells on the model.

A PolyData model is an unstructured triangulated surface. Discrete models generated in other segmentation programs (e.g. in STL format) can be imported into the SimVascular modeling pipeline as a PolyData model (Fig. 1.7). After importing, one can identify faces of the discrete model and perform the same set of operations that are available for models created in SimVascular.

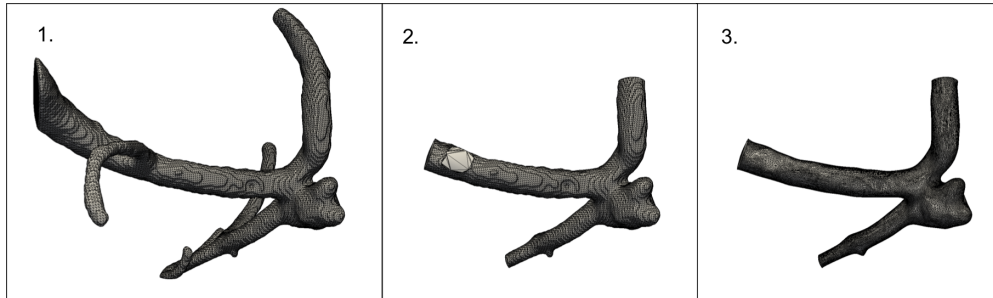


Figure 1.7: A geometry imported into SimVascular and prepared for meshing using the PolyData module. (1) The imported geometry, (2) extra and undesired portions of the geometry are removed and holes are filled, and (3) the geometry is smoothed, decimated, and subdivided.

OpenCASCADE - The OpenCASCADE (www.opencascade.org) modeling kernel provides 3D solid modeling functionality found in most CAD software. OpenCASCADE is the solid modeling package utilized in FreeCAD (www.freecadweb.org). Using this component, one can use SimVascular to create a CAD model by lofting the 2D segmentations into a non-uniform rational B-spline (NURBS) surface. This parametric format lends well to typical CAD procedures such as blending, cutting, and Boolean operations. These functions are accessible through SimVascular’s GUI and console.

Parasolid - Parasolid (Siemens PLM Software, Plano, TX, USA) is an *optional* licensed solid modeling plugin, which is the solid modeling package utilized in SolidWorks (www.solidworks.com). Using this licensed component, one can also perform lofting of 2D segmentations into a NURBS surface and access typical CAD procedures through SimVascular's GUI and console. In general, much of the functionality between Parasolid and OpenCASCADE is similar; however, Parasolid has generally been found to be more robust.

Discrete - The last solid modeling package, Discrete, is an *optional* plugin that simply provides a way to represent a discrete PolyData surface as a model that is usable by SimVascular's commercial mesher, MeshSim (Simmetrix, Inc., Clifton Park, NY, USA).

At the end of the model creation step, faces on the model are labeled with a user specified name and identifier (*ModelFaceID*). These identifiers can later be used to specify boundary conditions or material properties in the simulation steps. When a model is created using 2D or 3D segmentation approaches in SimVascular, names and *ModelFaceIDs* are automatically prescribed. Customized naming and prescription of faces can be accomplished using built in functionality, which is also helpful if the model is created using an external program and imported into SimVascular.

1.5 Meshing

After image segmentation and model construction, the next step for image-based blood flow modeling is discretizing the volumetric domain through mesh generation. The most robust meshing packages have traditionally been commercial codes, though in the past decade, high quality open-source meshing tools have also become available.

SimVascular supports two meshing kernels for the user to choose from: (1) TetGen and (2) MeshSim. The open-source TetGen kernel is actually a combination of functionality from TetGen (www.tetgen.org), as well as custom code for adaptive meshing, code from VMTK (www.vmtk.org) for boundary layer meshing and radius-based meshing, and MMG (www.mmgtools.org) for fast and robust surface remeshing. The *optional* MeshSim kernel is a licensed mesher by Simmetrix (www.simmetrix.com). Both MeshSim and TetGen kernels provide a broad and similar range of meshing options (Fig. 1.8). See Chapter 3 for a more in detail description of the meshing possibilities in SimVascular. Surface remeshing, local mesh refinement, and cylindrical mesh refinement are a few of the options available in both packages. Also, boundary layer meshing is supported, which enables smaller, thinner elements over near-wall regions where the gradient of the velocity normal to the surface changes most drastically. Additionally, mesh adaption based on *a-posteriori* error estimates is supported to provide a more efficient discretization strategy. This is achieved by computing the second directional derivative, or the Hessian, of the solution (e.g., blood velocity magnitude). The eigenvalues of this Hessian matrix at each mesh point are used as an indication of how much the solution is changing around this point, and the mesh is locally refined/coarsened accordingly [36]. This enables the intelligent placement of mesh points that are both refined

enough to capture the dynamics of the solution where necessary, but coarse enough to be computationally efficient where possible. See Chapter 3 for more details.

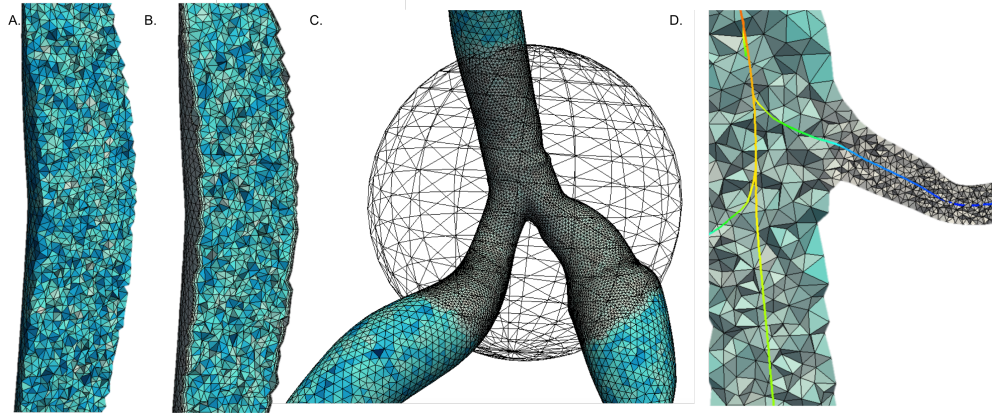


Figure 1.8: A variety of meshing options are available in SimVascular. (A) Uniformly prescribed element size on a mesh, (B) a boundary layer mesh, (C) a mesh with spherical refinement, (D) and a radius-based mesh.

1.6 Simulation

The meshing procedure produces an unstructured volumetric mesh that can be used as the computational domain for simulation of blood flow and pressure. The SimVascular simulation module includes three parts: (1) Presolver (svPre), (2) Flowsolver (svSolver), and (3) Postsolver (svPost).

1.6.1 Boundary Conditions

Boundary conditions are essential to obtaining valid, physiologically realistic cardiovascular simulation results. The foremost boundary condition is the traction (no-slip, no-penetration) boundary condition applied at the lumen surfaces (“walls”). The other boundaries can be considered inflow (“inlet”) and outflow (“outlet”) boundaries, and it is important that boundary conditions specified on these surfaces represent the physiology of the vasculature *outside* the 3D computational domain. For example, boundary conditions are essential for obtaining realistic values of pressure required for accurate fluid structure interaction simulations. SimVascular provides different options for boundary condition assignment at the three boundary types. Dirichlet or Neumann boundary conditions can be applied at either inlets or outlets of the model, which enables a broad range of options for boundary condition specification. These values can be directly prescribed, or implicitly prescribed from reduced order models of the upstream or downstream vasculature. Along these lines, inflow and

outflow boundary conditions can be prescribed in an “open-loop” or “closed-loop” manner (Fig. 1.9). For the latter, inflow conditions become coupled to the dynamics of the model itself, which can be necessary in surgical planning applications where one virtually changes the vascular geometry in a manner that may alter inflow conditions from the nominal or measured values.

Inlets - At the inlet (or inlets), a flow rate or pressure waveform is typically prescribed. The waveform is fit to a truncated Fourier series so that the flow rate (or pressure) at arbitrary time points can be queried. When a volumetric flow rate is specified, it is mapped to the inlet plane using a specified profile; plug, parabolic and Womersley profiles are currently supported in SimVascular. This mapping can account for non-circular inlet planes. In addition, SimVascular supports the ability to map planar phase contrast magnetic resonance imaging (PCMRI) velocity measurements to the inlet plane of the model for scenarios where such measurements are available [37].

Outlets - There are a number of techniques used in SimVascular for outflow boundary conditions that model the effects of the downstream vasculature, including impedance boundary conditions, Windkessel-type boundary conditions (resistance, RC circuit, RCR circuit, etc.), and more complicated lumped parameter network (LPN) models like coronary boundary conditions [38]. These boundary conditions effectively model the pressure-flow relationship at each outlet due to the respective downstream vascular bed. Impedance and Windkessel-type boundary conditions are coupled implicitly to the 3D computational domain by prescribing pressure in a weak manner in the flowsolver as described in detail in [39]. In addition, the coupled LPN network can be modified without needing to recompile the solver, thus making it very simple to implement a variety of boundary conditions.

We can view outlet boundary conditions as being specified by a lumped parameter (LP) model of the downstream vascular domain. Unlike distributed models (such as the 3D computational domain) that are governed by PDEs, LP models are governed by ODEs. Therefore, providing the ability to couple ODEs that represent the dynamics of the downstream vascular domain opens vast possibilities for modeling downstream physiology. In fact, such models can represent the entirety of the circulation outside the 3D computational domain, in which case one achieves a “closed-loop” model and the ODEs serve to both modulate outflow conditions and inflow conditions (Fig. 1.9, right). Except for very simple LP models, the ODEs cannot be solved analytically, and must be solved numerically. For such couplings, SimVascular contains an efficient and stable numerical scheme for implicitly coupling ODE models with the flowsolver for the 3D domain without significantly increasing the overall simulation cost [40, 41], which has been used in several recent applications [42, 43].

Walls - No-penetration, no-slip boundary conditions are applied for rigid wall simulations. Alternatively, the flowsolver can be used to model fluid structure interaction (FSI). For FSI simulations, the fluid and solid domains are coupled using the coupled momentum method (CMM) [44], with the wall modeled as a linear elastic material, which can have uniform or variable elastic modulus and thickness along each vessel.

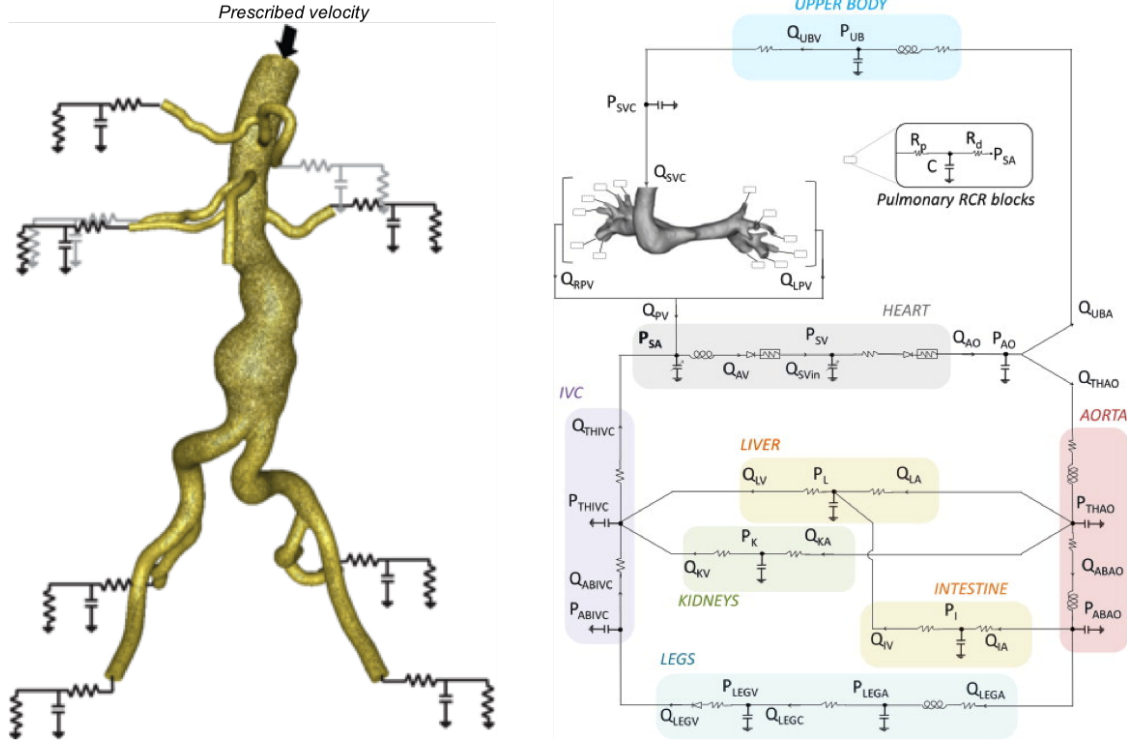


Figure 1.9: On the left, “open-loop” boundary conditions are prescribed on a model of an aorta (from [30]). RCR circuits are applied to represent the downstream vasculature. On the right, “closed-loop” boundary conditions are applied to a Hemi-Fontan model (from [43]).

1.6.2 Solver Methodology

Blood flow is modeled using the incompressible Navier-Stokes equations,

$$\begin{aligned} \rho \dot{v}_i + \rho v_j v_{i,j} - p_{,i} - \tau_{ij,j} &= 0, \\ v_{i,i} &= 0, \end{aligned} \quad (1.3)$$

where ρ is blood density, v_i is the i^{th} component of the fluid velocity and \dot{v}_i its time derivative, p is the pressure, and τ_{ij} is the viscous portion of the stress tensor. The flowsolver inside of SimVascular evolved from the academic finite element code PHASTA (Parallel, Hierarchical, Adaptive, Stabilized, Transient Analysis) for solving the Navier-Stokes equations in an arbitrary domain with the streamline-upwind/Petrov-Galerkin (SUPG) and pressure-stabilizing/Petrov-Galerkin (PSPG) methods [45].

The SUPG/PSPG formulation is defined on the finite-dimensional trial solution and weight function spaces \mathbf{S}_h^k , \mathbf{W}_h^k , and P_h^k . The domain is denoted by $\Omega \in \mathbb{R}^3$, and its boundary by $\Gamma = \Gamma_D \cup \Gamma_N$. Dirichlet boundary conditions are applied on Γ_D , and, Neumann, or flux type, boundary conditions are applied on Γ_N . Ω is discretized by n_{el} linear elements, Ω_e .

The weak form of Eq. (1.3) becomes

$$B_G(w_i, q; v_i, p) = \int_{\Omega} \{w_i(\rho \dot{v}_i + \rho v_j v_{i,j}) + w_{i,j}(-p\delta_{ij} + \tau_{ij}) - q_i v_i\} d\Omega + \int_{\Gamma_N} \{w_i(p\delta_{in} - \tau_{in}) + q v_{in}\} d\Gamma = 0, \quad (1.4)$$

where $\mathbf{w} \in \mathbf{W}_h^k$ and $q \in P_h^k$.

Momentum stabilization is required for advection dominated flows, and pressure stabilization is otherwise required to support the use of linear tetrahedral elements (P1-P1) in the SimVascular flowsolver for velocity and pressure, which is computationally efficient in terms of memory and mesh size. The following stabilized weak form is thus considered

$$B(w_i, q; v_i, p) = \underbrace{B_G(w_i, q; v_i, p)}_{\text{Eq. 1.4}} + \sum_{e=1}^{n_{el}} \int_{\Omega_e} \left\{ \underbrace{\tau_M (v_j w_{i,j} + q_i) L_i}_{\text{momentum and pressure stabilization}} + \underbrace{\tau_C w_{i,i} v_{j,j}}_{\text{incompressibility constraint stabilization}} \right\} d\Omega + \sum_{e=1}^{n_{el}} \int_{\Omega_e} \left\{ \underbrace{w_i \bar{v}_j v_{i,j} + \bar{\tau} \bar{v}_j w_{i,j} \bar{v}_k v_{i,k}}_{\text{compensation for the stabilization terms}} \right\} d\Omega = 0, \quad (1.5)$$

where $\mathbf{w} \in \mathbf{W}_h^k$ and $q \in P_h^k$. L_i represents the residual of the i^{th} momentum equation,

$$L_i = \dot{v}_i + v_j v_{i,j} + p_{,i} - \tau_{ij,j}. \quad (1.6)$$

This formulation includes both the momentum and pressure stabilization [46], which are controlled by the stabilization parameter, τ_M . The incompressibility constraint is also stabilized and is controlled by the stabilization parameter, τ_C . The addition of these stabilization terms causes inconsistencies in the conservation of momentum, so Taylor et al. [47] introduced the final term of the weak form to compensate for the momentum imbalance. For further details, see [45].

The above weak form contains stabilization terms for momentum, pressure, and the incompressibility constraint. In addition, backflow stabilization as described in [48] has been added to the SimVascular solver to prevent instabilities at Neumann boundaries that may experience backflow due to flow reversal (either total reversal due to conservation of mass, or partial reversal due to vortical structures near the outlet). This backflow stabilization method has been shown to provide a more robust and effective way to deal with numerical divergence caused by flow reversals at Neumann boundaries compared to more common methods [48].

The stabilized FEM formulation of Navier Stokes is discretized in time using the generalized alpha time-stepping scheme in the SimVascular flowsolver. A recently developed

linear solver with specialized preconditioners tailored to handle large vascular resistances coupled at outflow boundaries is used to handle cardiovascular simulations and reduce solution time [49], providing an alternative to the original commercial linear solver LesLib (Altair, Inc., Mountain View, CA). The flowsolver can be run with a single core or with multiple cores using the Message Passing Interface (MPI). A related version of the flowsolver has demonstrated excellent scalability on large clusters [50], which can enable the study of transiently or transitionally turbulent flow conditions.

1.6.3 Flow Analysis

SimVascular can post-process the simulation files to extract or calculate relevant hemodynamic quantities such as velocity, pressure, wall shear stress (WSS), and oscillatory shear index. Files can be exported to VTK formats to facilitate visualization of the data in leading open-source scientific visualization softwares such as ParaView and VisIt, as well as more custom post-processing by applying VTK classes and filters, which can be scripted using Python. For example, recent studies have analyzed flow fields produced by SimVascular to compute Lagrangian coherent structures [51, 52, 53, 54], residence times in aneurysms [55] [56], and turbulent kinetic energy [30, 8]. Flow fields have also been used to perform particle tracking [57, 58] and to model surface transport processes relevant for thrombosis [59, 60].

1.7 Case Studies and Validation

SimVascular has been used in a wide range of applications from studying blood flow in the heart, brain, and lungs and for various disease and surgical planning scenarios (e.g., [6, 7, 8, 9, 10, 11, 12, 13, 14] among others). In vitro validation in the thoracic aorta compared flow measurements from PCMRI in deformable phantoms to SimVascular FSI simulations [61]. The average difference between measured and simulated flow was approximately 13% (mean). The difference between the measured and simulated mean pressure was approximately 1.8%. Similar validation efforts were carried out in the coronary arteries and found similar agreement between measured and simulated flows [62]. In vivo validation has been performed by comparing fluctuating/turbulent kinetic energy computations obtained with SimVascular with measurements obtained using 4D flow imaging in an aortic coarctation in [8] as shown in Figure 1.10. In that study the quantified mean differences between in vivo measurements and CFD predictions of fluctuating kinetic energy were on the order of 10% and within expectations due to modeling and measurement errors. In addition, in vivo measurements of flow velocities in abdominal aortic aneurysms were shown in [54] to compare well to computations obtained using SimVascular, and simulated predictions of flow in Y-graft Fontan procedures were compared to in vivo clinical data in [63]. SimVascular has also been used in several recent CFD challenges for image-based hemodynamics modeling, e.g. [64]. Moreover, simulation-derived designs from SimVascular have been translated to clinical use, an example of which is the pilot study of the Fontan Y-graft [65, 66].

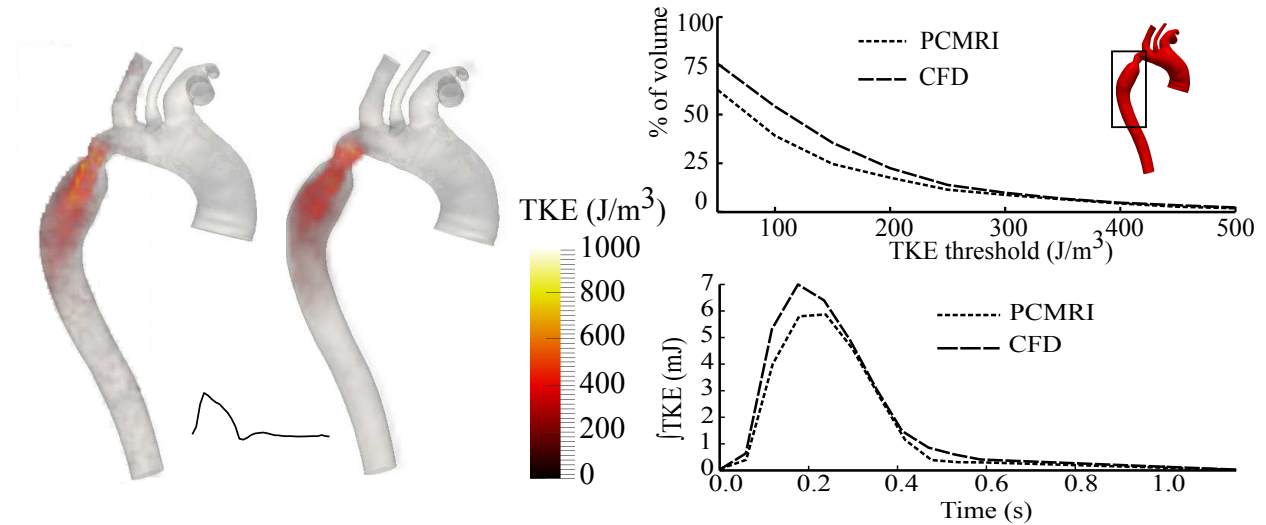


Figure 1.10: In vivo validation of SimVascular’s finite element flow solver for an aortic coarctation. (Left) Comparison of fluctuation intensity (TKE) fields from PCMRI and from SimVascular (CFD) during systole. (Right, top) Percentage of the descending aorta (boxed region) with fluctuation intensity above various thresholds at systole. (Right, bottom) Integral of the fluctuation intensity field over the descending aorta (boxed region) vs. time. Figures adapted from [8].

As a compelling exposition of SimVascular’s application to image-based hemodynamics modeling, the Open Source Medical Software Corporation (OSMSC) has compiled models and results from over 100 unique studies using SimVascular (Fig. 5.15). Figure 5.16 breaks down the contents of the cardiovascular and pulmonary model repository available to users at www.vascularmodel.com. We briefly present the results from two of these studies, as well as an example where FSI has been used for simulation of a coronary artery bypass graft (CABG).

Coronary Arteries: Coronary artery disease is the leading cause of death worldwide. Flow in the coronary arteries is unique in that it is distinctly out of phase with other systemic arterial beds due to high intra-myocardial pressure during systole, which effectively increases vascular resistance in the coronary beds. Figure 1.13 displays a model of the coronary arteries built in SimVascular from cardiac-gated CT data of a 63 year old female. The model includes the proximal aorta, and a typical aortic flow waveform was imposed at the aortic root, taken from [67]. Including the aorta and specifying aortic flow, rather than coronary flow has several advantages; aortic flow can be measured or estimated more easily; the left and right coronary flows are more naturally coupled; and with this geometry a heart model can be more naturally coupled [38]. The aortic waveform was scaled to a mean volumetric flow rate of 5 L/min with a period of 1 second (60 beats per minute). At the aortic outlet, a three element Windkessel or RCR model was applied. The parameters in the RCR model were found by setting the mean flow of the aortic outlet to 96% of the cardiac output, and 60/40 split for the

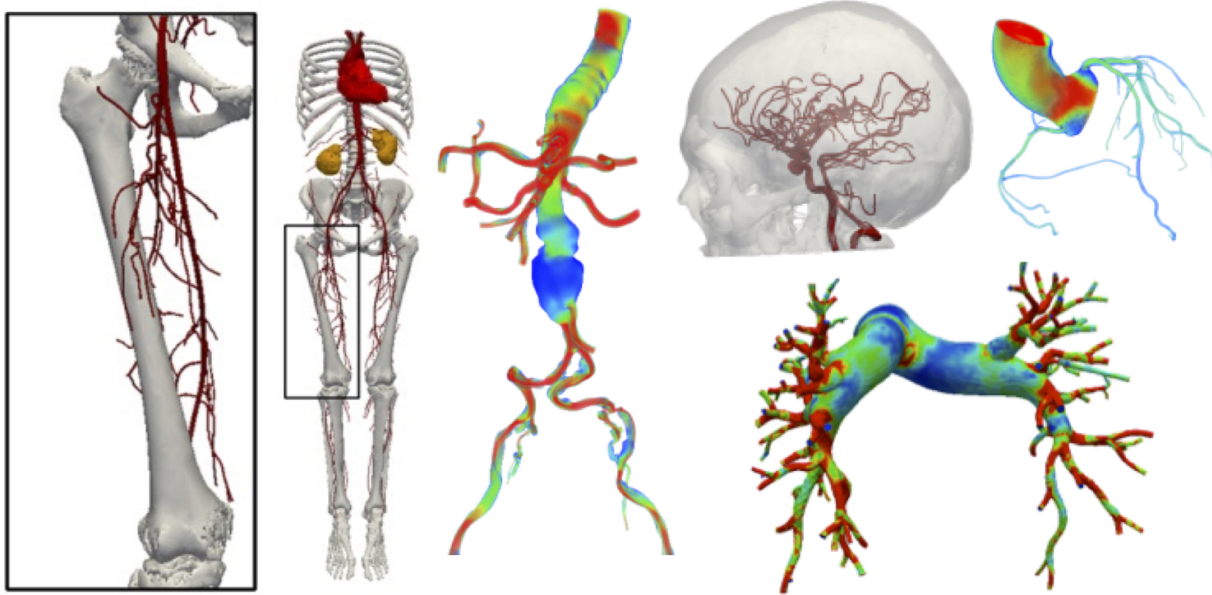


Figure 1.11: A sampling of the wide variety of model categories and simulation results available online in the vascular model repository at www.vascularmodel.com.

left and right coronary arteries. Lumped parameter coronary bed models (Fig. 1.13, upper left) were coupled at the model outlets using the coupled multidomain method [39, 68] with time dependent intra-myocardial pressure $P_{im}(t)$. Realistic systolic and diastolic pressures for a healthy adult of 120 and 80 mmHg were achieved. Volume rendering of the velocity field magnitude is displayed in Figure 1.13. One can observe highest flows in the coronaries during diastole. Similar methods have been applied to model Kawasaki disease and coronary bypass graft surgery using both open loop and closed loop circulation models [68, 9].

Pulmonary Arteries: The pulmonary arteries supply blood from the heart to the lungs for oxygenation. Pulmonary arterial hypertension (PAH) and pulmonary embolisms are common diseases associated with the pulmonary arteries. Image data from a woman aged 67 was used to construct an extensive model of the pulmonary arteries from the main pulmonary artery to various levels of branching in the left and right pulmonary pathways (Fig. 1.14). A total of 100 arteries were modeled. The inflow waveform was adapted from [69] to represent a typical resting pulmonary waveform. Resistance boundary conditions were used at all outlets. Resistance values were distributed inversely to outlet area and with total values chosen to match physiologic flow splits and pulmonary pressures. Wall shear stress values in the proximal arteries were observed to match values in previous studies, and flow rates through the main pulmonary arteries were consistent with measured values from PCMRI [70]. This application was used to evaluate wall shear stress and other quantities in normal and PAH patient specific models, revealing significant differences between healthy

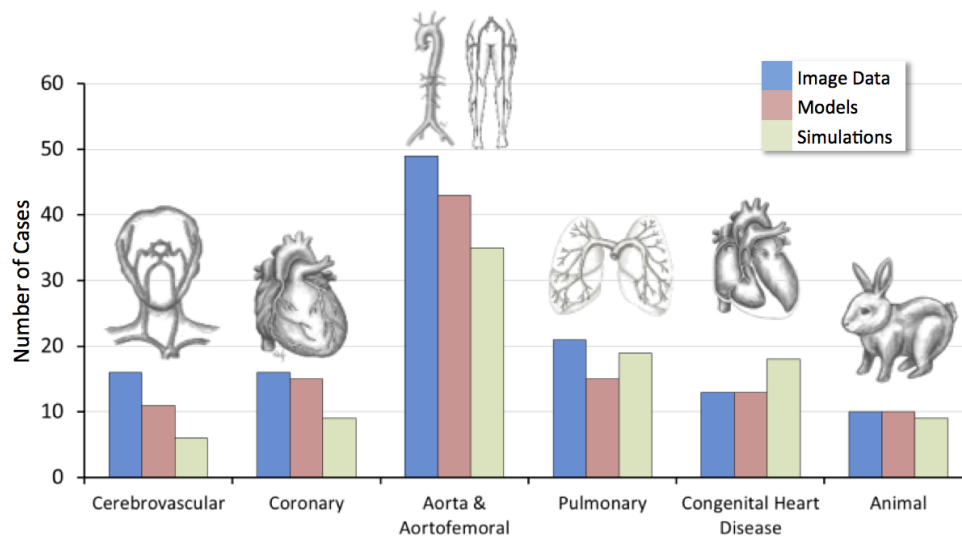


Figure 1.12: The vascular mode repository combines the results of over 100 studies of varying image data, model complexity, and simulation type.

and diseased states, which helped to reveal mechanisms for PAH progression [71].

Aortic and Femoral Arteries: The second example is a model extending from the aortic root to the femoral arteries (Fig. 1.15). This model was constructed from a large CT dataset of a 21-year old female subject. The aortic inflow waveform was taken from [72] and averaged to a mean cardiac output of 4.6 L/min. Three element Windkessel models were applied at each outlet with resistances and capacitances tuned to achieve desired flow distribution amongst the various outlets and physiologic pressure pulse. Various literature sources were used to support the distribution ratios to each model outlet. For example, 13% of the cardiac output was distributed to the carotid arteries, 65% to the descending thoracic aorta, and 22% to the subclavian arteries. Descending thoracic flow was further divided to the remaining arterial beds based on target flow rates from the literature. Target arterial pressures were based on typical pressures for a young healthy adult. The simulation results match a target diastolic pressure of 80 mmHg and systolic pressure of 120 mmHg.

Coronary Artery Bypass Graft: Coronary artery bypass graft surgery is performed in roughly 400,000 patients annually in the United States [74]. Vein graft failure continues to be a major clinical challenge in patients post CABG surgery. Simulations including material wall properties and vessel wall deformation may give insight into flow and wall mechanics leading to vein graft failure and optimal choice of surgical method. In this example, a model including the aorta, the coronary arteries, and the graft were constructed from CT images with SimVascular. Vessel wall thickness and material properties were prescribed based on literature values [75, 76]. For the boundary conditions at the aortic inlet and outlet, a closed-loop LPN was used that included circuit blocks for the heart, the systemic circulation, and the coronary circulation. Because flow in the coronary arteries is out of

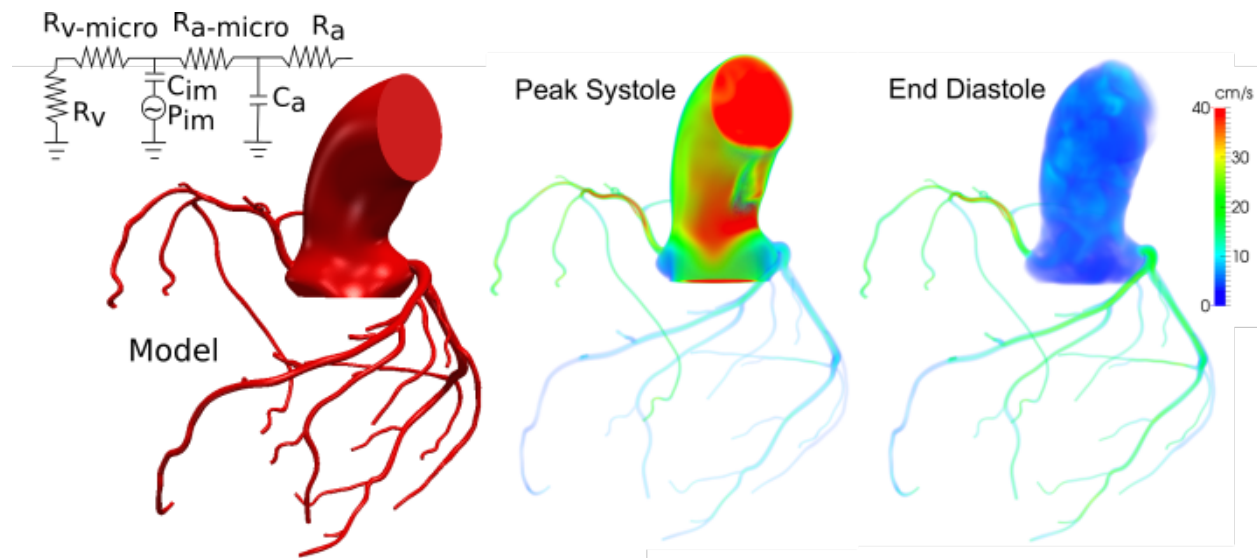


Figure 1.13: Model of the coronary arteries of a 63 year old female reconstructed from CT. The upper left inset is a schematic of a LP distal coronary bed model, one of which is coupled at each coronary outlet. The right two images show volume renderings of the computed velocity magnitude fields at peak systole and end diastole.

phase with the systemic circulation, coronary specific boundary conditions were applied at each of the coronary artery outlets. The model was tuned to send 4% of the cardiac output to the coronary arteries [77]. In addition, target pressures and flow splits were matched using values taken from literature. Velocity during end diastole and wall displacement during peak systole are displayed in Figure 1.16. Significant differences in biomechanical conditions between venous and arterial grafts were identified [78].

1.8 Discussion

SimVascular provides a complete pipeline for image-based hemodynamics simulation. Many custom features have recently been developed to enable efficient and flexible computer model construction from medical image data. While this software has benefited from more than a decade of development and use in state-of-the-art cardiovascular modeling research studies, the recent redevelopment of SimVascular has expanded and hardened its functionality and ease of use. Moreover, these recent efforts have made SimVascular completely open source, documented, and available on all major operating systems, which enables community use for research and education for the first time. Many of these efforts are documented in following chapters of this dissertation.

Because geometric fidelity and boundary conditions are of critical importance in accurate cardiovascular simulation, new features have focused on providing enhanced functionality

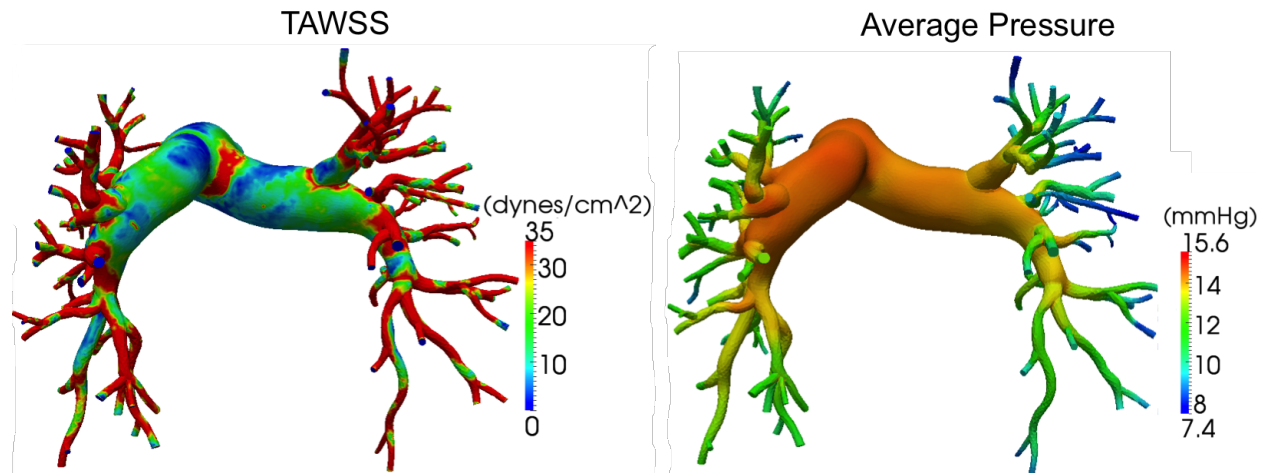


Figure 1.14: Time averaged wall shear stress and average pressure over one cardiac cycle for the pulmonary arteries of a 67 year old woman.

for model construction, manipulation, and repair, as well as the specification and numerical treatment of physiologic boundary conditions, multidomain modeling, and fluid structure interaction. Many of these features are not possible in other softwares. The SimVascular solver has undergone significant development to include support for multiscale boundary conditions, backflow stabilization, and a new linear solver with specialized preconditioning to improve performance. Significant efforts have also been made to refactor and harden the SimVascular code for stable releases and standardized development.

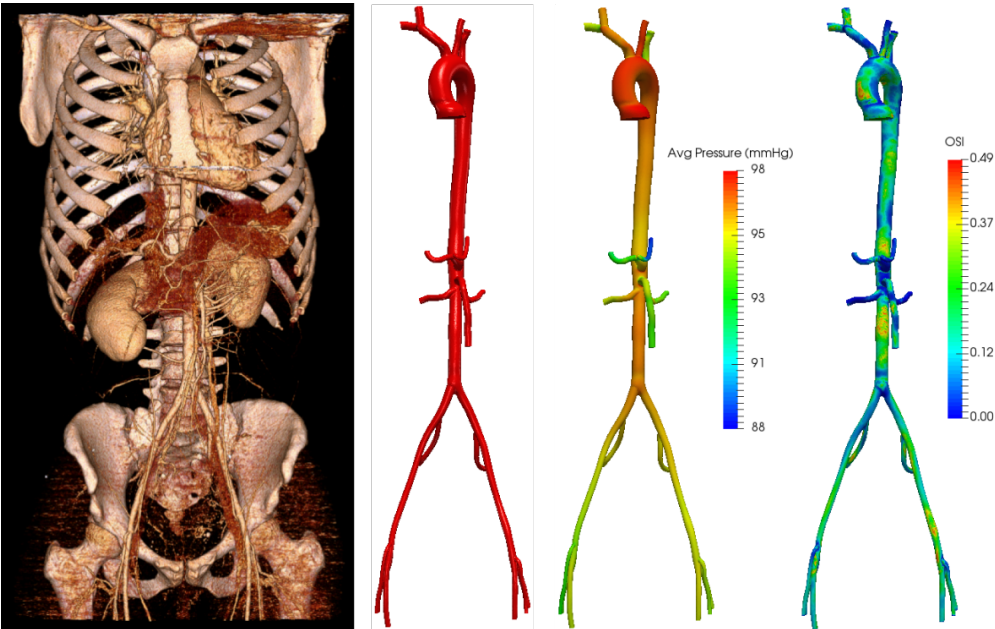


Figure 1.15: The image volume and the constructed model for the aortic and femoral arteries of a 21 year old female (left two panels). Representative simulation results of the time-averaged pressure field and oscillatory shear index (OSI) [73] field are shown (right two panels).

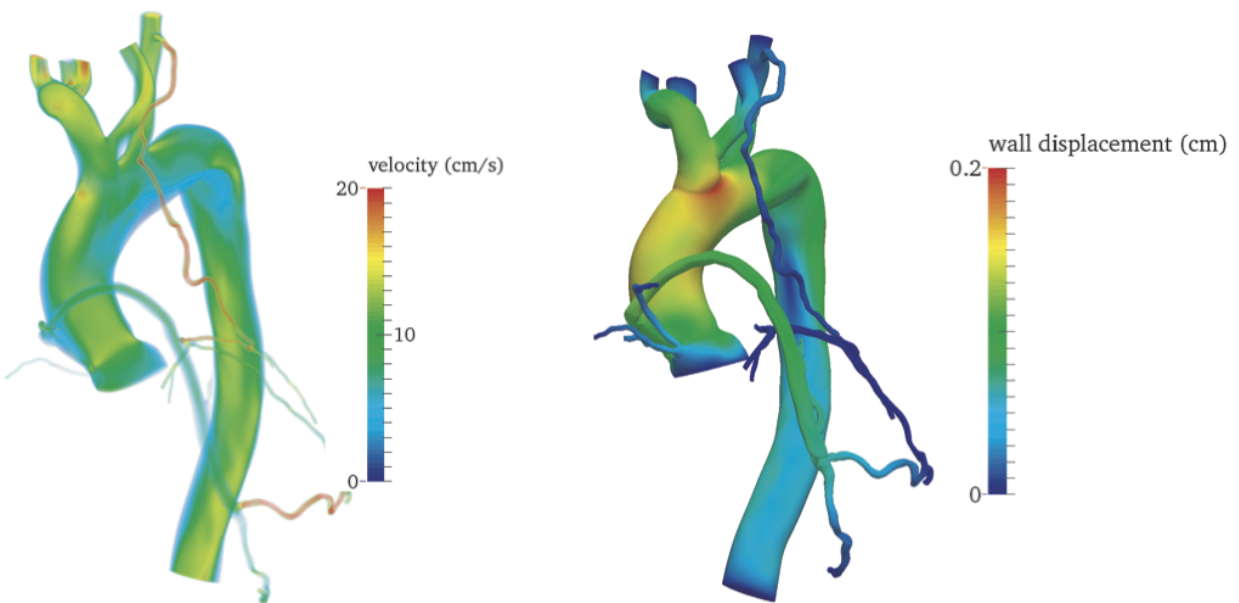


Figure 1.16: Model of the aorta, coronary arteries, and a bypass graft constructed from CT. Velocity during end diastole (left) and wall displacements during peak systole calculated from FSI simulation (right). Adapted from [78].

Chapter 2

Vascular Modeling for Finite Element Analysis

Parts of this chapter are published in [35] and in collaboration with *Nathan M. Wilson* and *Shawn C. Shadden*.

2.1 Introduction

Patient-specific vascular models constructed from medical image data are geometrically complex due to the natural winding and branching present in vascular structures. It is also very common for vascular models to have large variations in size scale. For example, anatomic models simulating blood flow in the coronary arteries typically include a portion of the aortic arch which can be an order of magnitude larger than the coronary arteries. In addition, many constructed geometries are diseased and contain abnormal anatomical structures such as an aneurysm (vessel expansion) or a stenosis (vessel thinning). Another challenge in vascular modeling arises from varying image modalities and quality. As an example, when using CT image data, it may be difficult to differentiate bone from vasculature. Additionally, image quality can be poor with many obstructing artifacts, and image resolution can also occasionally be greater than one millimeter, making it difficult to identify smaller vessel boundaries. All of these factors combine to make modeling patient-specific vascular geometries a difficult challenge.

Moreover, the constructed vascular model has a profound impact on blood flow simulation results and can be a major source of error. A manual construction process requires the modeler to make many decisions that affect the resulting model, which makes the model highly dependent on the user. There are a variety of tools that have been developed to help automate the process, [79, 80, 81, 82], and reduce the number of modeling decisions being made. The downside to automation is the ability to customize a model is lost. For example, adding a stent graft or varying the angle of a bifurcation would not be possible in a more automated framework where the output model depends primarily on the image data. Thus,

a set of efficient, well-developed, and semi-automated tools is the best possible framework for academic studies of the cardiovascular system. This chapter describes techniques for open-source vascular model creation through the lofted 2D segmentation approach and a variety of model modification tools. As described in section 2.2, there are many different solid model representations. The two representations that are used heavily in SimVascular will be discussed separately with an in depth discussion of the operations built around the representations.

2.2 Solid Model Representations

Once a vascular geometry has been segmented through either the lofted 2D segmentation technique, the direct 3D segmentation technique, or a combination of the two [83], the geometry can be represented in many different ways. There are a variety of different geometric representations, and these are typically broken down into two main categories (Fig. 2.1).

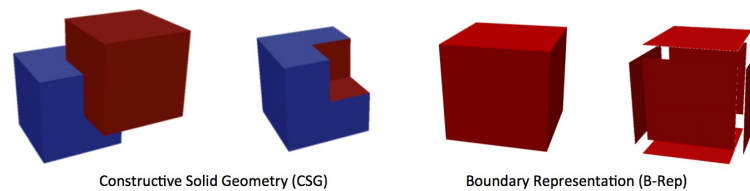


Figure 2.1: Constructive Solid Geometry and Boundary Representations are fundamentally different in object representation.

- Constructive Solid Geometry (CSG) - Constructive solid geometry is the combination of geometric primitives (e.g. cube, cone, cylinder) through boolean operations (union, intersect, difference) to construct a solid.
- Boundary Representation (B-Rep) - A boundary representation is the union of a set of non-overlapping geometric elements that define the exterior of a solid volume.

Constructive solid geometry can be useful when creating geometries of regular shapes; however, it is much more difficult to represent a free form geometry like that of the human vasculature with CSG. Though CSG was utilized in many early CAD frameworks, B-Rep is by far the more popular representation today. The B-Rep will be the focus for the remainder of this work. There are many representations that can be used to make up the set of non-overlapping geometric elements defining a B-Rep. Two of the most popular representations are triangulated surfaces and non-uniform rational B-splines (NURBS). Triangulated surfaces are typically used for representation of free-form geometries, while NURBS are the standard in many CAD frameworks and are commonly used in mechanical design. Although

these representations can be used to represent the exact same geometry, they are very different in nature. Triangulated surfaces are defined *discretely* as a set of points and connectivity of those points as triangles, while NURBS are defined *parametrically* as a set of splines defined on piece-wise basis functions. Thus, it is important to note some of the differences in discrete and parametric modeling.

- Discrete modeling - A model is represented using a point set and some sort of connectivity of that point set. This is sometimes referred to as a piece-wise linear complex. The most common representation is a triangulated surface mesh, although quadrilateral surface meshes are also quite common. The discrete model focus here will be the triangulated surface mesh. Typical file types used to store triangulated surfaces are *.stl, *.ply, *.obj, and *.vtp.
- Parametric modeling - A model is represented as a set of multi-dimensional parametric splines. This is typically what is used in many Computer Aided Design (CAD) frameworks. It is exact and models can be modified easily making it ideal for design. The CAD standard is the non-uniform rational B-spline (NURBS) surface, thus the parametric model of focus here will be NURBS. Typical NURBS file types are *.step and *.iges.

2.3 Discrete Modeling

Many image-based segmentation methods result in a discrete surface representation of the region of interest. For example, all direct 3D segmentation methods by nature result in a discrete representation. However, due to the nature of the lofted 2D segmentation technique, it is possible to form a discrete or parametric surface representation. As mentioned previously, the representation of choice for discrete modeling is the triangulated surface.

2.3.1 Lofting 2D Segmentations

When forming a tubular surface from the group of segmentations, the segmentations need to be connected in some manner to form a surface from the segmentations. The simplest way would obviously be combining the segmentations linearly to form what is referred to as a ruled surface. There are many advanced techniques for combining the segmentations and are described in literature [84]. The following steps are taken to form a triangulated surface from image data using the lofted 2D segmentation approach.

1. A pathline is created along the vessel/s of interest (Fig. 1.5(a)).
2. Each vessel pathline is traversed and a perpendicular slice plane called the segmentation slice plane is used to create a local 2D segmentation of the vessel (Fig. 1.5(b)). There are a variety of segmentation methods including thresholding, level set, and analytic.

3. The group of segmentations for a given vessel are sampled (s1) to a uniformly distributed set of points given a user defined resolution. The points are then reordered for each segmentation such that the initial point on each curve is the closest point in 3D space to the next adjacent curve. This leads to a series of “streaklines”, or piecewise linear paths, along the length of the vessel. (Fig. 1.5(c))
4. The group of segmentations are either connected with a series of splines or a parametric surface is solved for using the ordered segmentation points. The surface is triangulated at a user-defined spacing and caps are added to the end of the vessel. More details of these different methods are provided in Section 2.3.1.
5. If the model contains multiple vessels, the vessels are lofted individually and then combined with a union operation. A common mistake in vascular modeling with lofted 2D segmentations is to not completely enclose a child vessel in a parent vessel. The child vessel must be completely enclosed in the parent vessel in order to form a complete union and form a valid vessel network.

2.3.2 Boolean Procedure

Step five above is a complicated procedure for discrete surfaces. This section expands on the Boolean procedure and describes algorithms developed to perform Boolean operations on triangulated surfaces. This custom Boolean operation was developed in order to have a robust and succinct implementation that retained necessary information during the Boolean procedure. There are a variety of Boolean algorithms for B-Reps described in the literature. They can be classified in four categories by the computational approach: (1) tolerance and exact arithmetic, (2) approximate arithmetic, (3) volumetric, and (4) image space techniques [85]. Tolerance and exact arithmetic methods both compute the intersection between two solids on their exact boundary, but contain different techniques for dealing with geometric robustness. Tolerances restrict floating point numbers to a specified decimal place for geometric tests; whereas, exact integer arithmetic methods convert floating point numbers to an integer-based system in which computations can be carried out exactly [86, 87, 88, 89]. Approximate arithmetic methods reduce the computational complexity before running the geometric algorithm [90, 91]. This will make the computation simpler and quicker, but lacks the exactness of the previous approaches. Volumetric techniques represent the solids first as a volume [92, 93], and then perform the Boolean [94]. This results in a robust implementation; however, the boundary between the two solids is typically not resolved well in the output and a loss of geometric detail is seen [95]. Lastly, image space techniques take advantage of graphics hardware to quickly provide a boundary evaluation of the Boolean [96, 97]; these methods are typically used for object collision detection. Many of these algorithms use Layered Depth Images (LDIs) to store information about the depth piercing of ray tracing from the viewpoint and have the same pitfall as the volumetric methods in which the geometric detail is not exact; however, they do provide a fast and robust method to obtain a visual of the Boolean boundary [98, 99].

The Boolean implementations in literature also differ by the type of data used in the computation. Many implementations focus on NURBS surfaces [100, 85]; however, there are others that investigate the procedure for polygonal surfaces [101], which have been the leading representation for discrete solid models derived from image data. There are a limited number of libraries providing open-source, available, and usable Boolean operations for polygonal surfaces. The Visualization and Computer Graphics Library (VCG) has an implementation of the Boolean operation, which is implemented within the software MeshLab [102]. This is a volumetric implementation, and thus lacks the exactness of geometric detail described above. Another implementation is the GNU Triangulated Surface Library (GTS, <http://gts.sourceforge.net>), which is robust, but this package is no longer maintained and is difficult to include in software projects that require customization. VTK maintains an implementation for the Boolean of triangulated surfaces [103], however the implementation is not robust and often fails in various manners described below.

Rarely does the result of a Boolean give a surface that is ready for meshing and simulation, and typically other surface preparation methods must be performed. Besides defining faces for specification of boundary conditions or material properties, methods providing smoothing, blending, and manipulation of surfaces are necessary to give a solid that both accurately represents the image data and is valid for computational modeling. Moreover, discrete solid models obtained through image segmentation are often limited in quality by the resolution of the image data and inherent noise. Therefore surface manipulation tools are also necessary to improve the quality and representation of discrete solid models that serve the basis for quantitative postprocessing and simulation.

2.3.2.1 Boolean Overview

A Boolean procedure takes as input two objects and outputs some combination of these objects. Henceforth, the first input object is denoted as A and the second object as B . The possible Boolean operations between two objects are their **union** ($A \cup B$), **intersection** ($A \cap B$), and **difference** ($A - B$ or $B - A$). For this work, the objects are assumed to be two triangulated surface meshes. Moreover, these surfaces are considered B-Reps, or exterior surfaces, of volumetric objects in 3D space. Thus, one may view the Boolean as the union/intersection/difference of the enclosed volumes, subsequently restricted to the surface mesh.¹

To explain the Boolean procedure, it is helpful to introduce a couple of definitions **Intersection loops** define where one surface crosses the other. The **sub-surfaces** are the portions of each surface that are separated by the intersection loops. The Boolean in general consists of (1) finding the intersection loops between the two objects, (2) separating the objects into appropriate sub-surfaces, and (3) determining the appropriate combination of the sub-surfaces for the desired Boolean. For a discrete polygonal Boolean, an additional step is required. After the intersection loops are found, the two input surfaces are re-triangulated

¹This interpretation holds for water-tight surfaces. As discussed further below, for open surfaces the Boolean is strictly between the surface objects.

to conform to the intersection loops. In this way, the respective sub-surfaces can be combined to give a valid output once the Boolean is performed. The following summarizes the computational steps in the Boolean process for discrete polygonal surfaces:

Intersection. Determine where the input surfaces intersect in space. This step creates the intersection loops that are used for re-triangulation and sub-surface determination (see Section 2.3.2.2).

Re-triangulation. Re-triangulate each surface near the intersection loops. The intersection loops are comprised of intersection points and lines on each surface, and each surface is re-triangulated separately (see Section 2.3.2.3).

Boolean. Determine the correct combination of sub-surfaces for output. The sub-surfaces are extracted based on their orientation relative to the intersection loops incident on the surfaces (see Section 2.3.2.4).

2.3.2.2 Intersection

The first step in the Boolean is finding intersection loops. The methods used to find intersection loops of two surfaces follows the work described in [103] with some modifications. Finding the intersection loops between two discrete surfaces is difficult because all individual intersecting cells between surfaces A and B must be found. Thus, intersections are found with the help of *vtkOBBTree*, an oriented bounding box tree class found in VTK. An oriented bounding box tree partitions space occupied by a discrete surface into subregions to enable better location queries. An oriented bounding box (OBB) is the smallest volume box that encloses a specified number of cells. The *vtkOBBTree* class contains a function *::IntersectWithOBBTree()* that takes as input another *vtkOBBTree* to determine which OBBs from each surface intersect. For intersecting OBBs, the callback function *::FindTriangleIntersections()* uses a function *::TriangleTriangleIntersection()*, defined below, to find triangle intersections. An example of two oriented bounding box trees is displayed in Fig. 2.2.

The function *::TriangleTriangleIntersection()* is an existing function in the VTK class *vtkIntersectionPolyDataFilter*, however this function does not provide information regarding intersection points or origin surfaces, which are necessary for most Boolean implementations. An **intersection point** is a point where two surfaces meet that is incident on the edge of a triangle from either surface. The **origin surface** is the input surface containing the triangle edge that the intersection point lies on (cf. Fig. 2.5). A new *::TriangleTriangleIntersection* function was developed in order to track and maintain the intersection points and origin surfaces for later use, as described next.

The function *::TriangleTriangleIntersection()* takes as input 2 sets of 3 points, with each set representing a triangle from each surface. A check is first performed to determine if all three points from one triangle lie on one side of the supporting plane of the other triangle, excluding the possibility of intersection. The **supporting plane** is the plane containing the triangle of interest. If intersection is possible, the intersection line is found and restricted

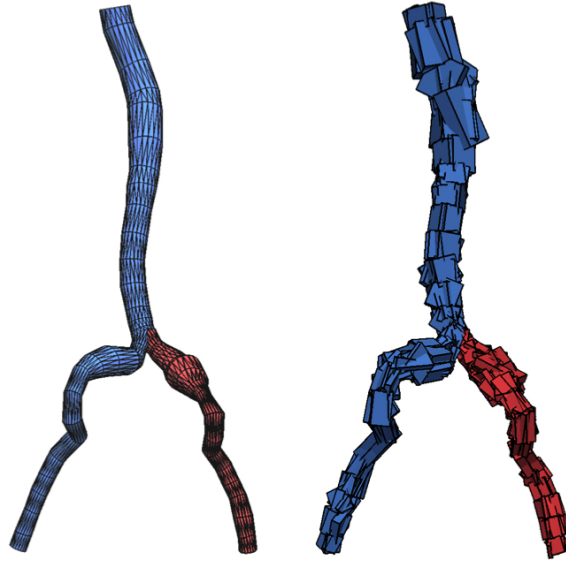


Figure 2.2: Left: The triangulated surface representation of a vascular model that was created by first developing a parent artery (blue) and branch artery (red) to be unioned. Right: oriented bounding boxes for each respective surface mesh. Each bounding box is specified to contain at most 10 surface triangles.

to the OBBs of the two triangles. The **intersection line**, \mathbf{l} , is the line where the supporting planes of the two triangles intersect. Each edge, \mathbf{e} , of each triangle from surface A is intersected with \mathbf{l} to give parametric values describing the intersection as shown in Fig. 2.3. Equations (2.1) and (2.2) give the intersection point, P , of the two line segments, and the parametric values (α_1, α_2) describe where the intersection occurs on each line segment. The same procedure is followed with the triangles from surface B. The above procedure is only performed for triangles contained in intersecting OBBs.

$$\underline{P} = \underline{a} + \alpha_1(\underline{b} - \underline{a}) \quad (2.1)$$

$$\underline{P} = \underline{c} + \alpha_2(\underline{d} - \underline{c}) \quad (2.2)$$

A necessary, but not sufficient, condition for two triangles to intersect is that \mathbf{l} must be intersected by at least 2 edges from each triangle. This can be determined by checking if the parametric values α_1 and α_2 are between 0 and 1 for two edges on each triangle. While this ensures both triangles intersect \mathbf{l} , this does not ensure the triangles intersect each other. Let α_A and β_A denote the α_2 value for the two intersecting edges from the triangle on surface A. Similarly, let α_B and β_B denote the α_2 value for the two intersecting edges from the triangle on surface B. Triangle intersections are determined by comparing these values as shown in Fig. 2.4 and Algorithm 1.

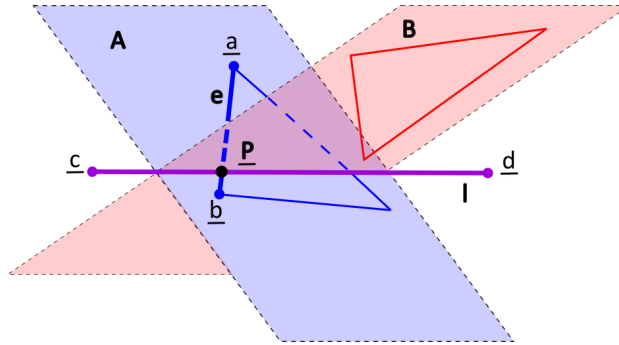


Figure 2.3: Line segment e , which is an edge of the triangle from surface A , has endpoints a and b . This intersects with intersection line l with endpoints c and d . The intersection gives two parametric values to determine where the intersection occurs along each respective line. Although the triangle from surface A intersects l , the triangles do not intersect because the triangle from surface B does not also intersect l .

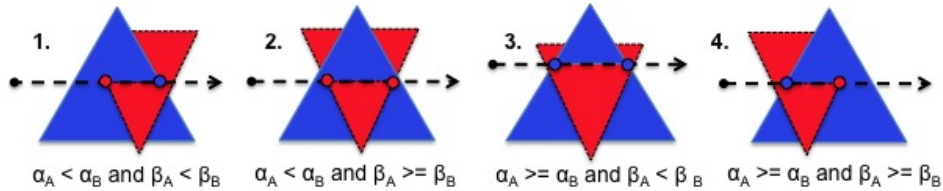


Figure 2.4: The different intersection types and how the parametric values determine the origin surface of the intersection points. The dashed line, l , is the intersection line, and the arrow indicates the order in which the points are considered. The blue triangle is from surface A and the red triangle is from surface B .

Algorithm 1 ::TriangleTriangleIntersection()

```

if  $\alpha_A < \alpha_B$ 
  if  $\beta_A < \beta_B$ 
    1. First point originates from surface  $B$ , second point originates from surface  $A$ 
  else if  $\beta_A \geq \beta_B$ 
    2. Both points originate from surface  $B$ 
  end if
else if  $\alpha_A \geq \alpha_B$ 
  if  $\beta_A < \beta_B$ 
    3. Both points originate from surface  $A$ 
  else if  $\beta_A \geq \beta_B$ 
    4. First point originates from surface  $A$ , second point originates from surface  $B$ 
  end if
end if

```

An intersection point may originate from: (1) surface A , (2) surface B , or (3) both surfaces. Fig. 2.5 demonstrates the *origin surface IDs* given to intersection points from the intersection of two discrete spheres. These surface IDs determine whether a point is part of the boundary of an intersected triangle, which is needed in the re-triangulation step described below. The prior VTK implementation did not go through the process of finding the surface IDs. Instead, a distance calculation was used in the re-triangulation to determine whether a point is part of the triangle boundary. This calculation is an unnecessary step and often leads to an incorrect surface origin determination. This approach provides a more robust solution to this problem. Each intersection point is stored in a *vtkPoints* object with its designated Origin Surface ID, and each intersection line is stored in a *vtkCellArray*. In addition, intersected triangles are marked for re-triangulation.

For a Surface ID of 3, edges from each surface intersect. In this case, there are at least two triangle-triangle intersections that will both locate this intersection point. Therefore, duplicate intersection points are removed. A point is flagged as duplicated if its distance from a prior intersection point is less than some specified tolerance (default $1e-6$).

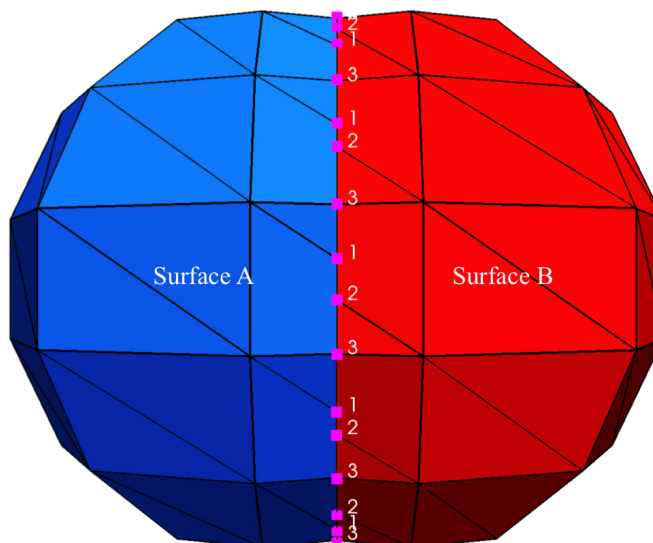


Figure 2.5: The intersection of two discrete spheres defined by intersection points with labeled origin surface IDs. 1 - intersection originates from Surface A; 2 - intersection originates from Surface B; 3 - intersection originates from both surfaces.

2.3.2.3 Re-Triangulation

The second step in the Boolean is the re-triangulation of both surfaces. The function `::SplitMesh()` is the overarching re-triangulation function within the *vtkIntersectionPolyDataFilter* and the basic steps performed by this function are listed in Algorithm 2. This function is called separately for each input surface after the intersection loops have been

found. It calls a function $::SplitCell()$, which carries out the re-triangulation for intersected triangles. Algorithm 3 lists the general steps in the re-triangulation process. This new triangulation is found by first finding the **cell loops** of the split triangle. Fig. 2.6 and Algorithm 4 demonstrate the process of finding cell loops for an intersected triangle.

Algorithm 2 $::SplitMesh()$

```

for Each Input Surface  $S_{in}$ 
  for Each Cell ( $C_i$ ) in  $S_{in}$ 
    if  $C_i$  is not an intersected triangle
      Copy  $C_i$  to  $S_{out}$ 
    else
      Call  $SplitCell(C_i)$ 
      Output Re-triangulated  $C_i$  to  $S_{out}$ 
    end if
  end for
end for

```

Algorithm 3 $::SplitCell()$

```

Call  $GetLoops(C_i)$ 
for Each Loop ( $L_j$ ) in  $C_i$ 
  Re-triangulate  $L_j$  with Delaunay 2D or Ear Clipping
end for
Attach information about new triangles in  $C_i$  to intersection lines
Return new triangles of  $C_i$ 

```

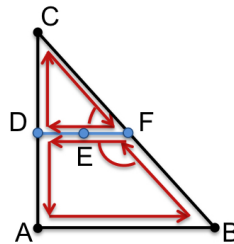


Figure 2.6: A triangle with an intersection splitting the cell into two loops. The loops are found using Algorithm 4.

The intersected cells are re-triangulated according to Algorithm 3, and these are added to the re-triangulated surface along with the existing non-intersected cells. The IDs of all cells adjacent to the intersection line are saved as $vtkCellData$ for later use. At this point, the intersection is finalized, and the surfaces can be checked to ensure a proper triangulation

Algorithm 4 *GetLoops()*

```

while There are untouched cell vertices or lines
  Start at one of the cell's original vertices ( $v_A$ ) and move to next vertex ( $v_{next}$ )
  while The next vertex ( $v_{next}$ ) is not vertex  $v_A$ 
    if  $v_{next}$  is connected to only two lines
       $l_{next} \rightarrow l_{connected}$  ( $l$  that is not  $l_{next}$ )
      Follow along next line ( $l_{next}$ ), comprised of  $l_{next}[v_1]$  and  $l_{next}[v_2]$ 
       $v_{next} \rightarrow l_{next}[v_2]$ 
    else
      if The loop does not yet have an orientation
        Find  $l$  that makes a minimum angle with current  $l_{next}$ 
        Calculate the orientation of the loop (CW or CCW)
      else
        Find  $l$  with minimum angle that follows the loops orientation
      end if
       $l_{next} \rightarrow l_{min}$ 
       $v_{next} \rightarrow l_{next}[v_2]$ 
    end if
  end while
end while
Return Cell Loops

```

was formed, i.e., that each edge adjoins an appropriate number of triangles. At the end of re-triangulation, the VTK class *vtkSVLoopIntersectionPolyDataFilter* is completed.

2.3.2.4 Union, Subtraction, Intersection Determination

To obtain the correct Boolean output, a VTK class *vtkSVLoopBooleanPolyDataFilter* was developed. The existing *vtkBooleanOperationPolyDataFilter* was found to have several deficiencies, and a new class was written in its place using alternative methods as described here. To begin the Boolean, the intersection loops are pre-processed to determine the type of intersection occurring between the surfaces. There are 3 intersection types for discrete polygonal surfaces [104], which are shown in Fig. 2.7. Each intersection loop is considered to be discretely represented by intersection points connected by intersection lines, which correspond to nodes and edges of triangles on each respective surface due to the re-triangulation step above.

- **Hard Closed Loop Intersection** - Every intersection point is connected to two intersection lines, and the beginning point of each loop is the end point. There can be any number of intersection loops, but no intersection point can be connected to more or less than two intersection lines.

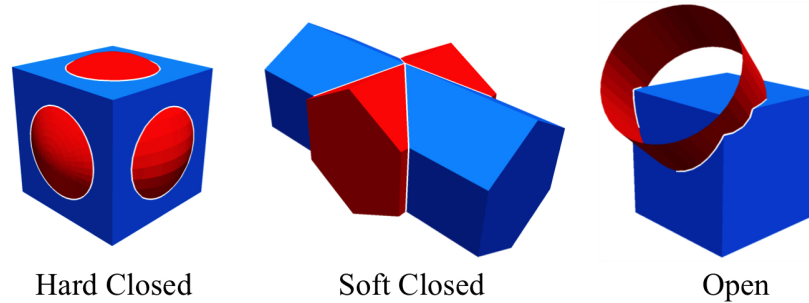


Figure 2.7: The different intersection loop types possible. Left: An intersection containing 6 hard closed loops, Middle: An intersection containing 2 soft closed loops, Right: An intersection containing 1 open loop.

- **Soft Closed Loop Intersection** - Every intersection loop is closed; however, an intersection point can be connected to more than two lines. For example, in the intersection between two cylinders of the same radius, there are two **soft closed intersection points** that are connected to four intersection lines. The beginning point of each loop is still the end point of the loop, but points within the loop may be attached to more than, but not less than, two lines.
- **Open Loop Intersection** - Intersections do not form complete loops. There are points on the intersection lines that are only connected to one intersection line, and thus, are the ends of that loop. The beginning point is not the end point of the loop as the intersection loop is not closed.

It is possible to have a Boolean that gives rise to multiple intersection types. It is also worth noting that if the surfaces are water-tight it is not possible to have an Open Loop intersection. This is only possible for surfaces with free edges. A **water-tight** surface is one in which every cell edge has two neighbor triangles. An **open surface** has at least one edge that has only one neighbor triangle; since portions of the surface are open, it may not be considered as enclosing a volume.

As pre-processing, the number of intersection loops and the type of intersection loops are found and stored. Then, each intersection loop is run through in an oriented manner to obtain the **intersected sub-surfaces** partitioned by the intersection loops. The VTK class *vtkSVLoopBooleanPolyDataFilter* has a function *::GetBooleanRegions()* that obtains the intersected sub-surfaces for each input, and is described in Algorithm 5.

The cells adjacent to the intersection loop lines are given an orientation based on their alignment when the intersection loop is transversed in a specified direction. For example, in Fig. 2.8, the intersection line is oriented bottom to top, making the cell to the left CCW and the cell to the right CW. All cells attached to this first cell but not outside the containment of the intersection lines are assigned this orientation with the use of a **flood fill** algorithm that designates all connected elements inside the subsurface the same orientation. The boundaries

Algorithm 5 *GetBooleanRegions()*

```

for Each intersection loop ( $L_i$ )
  for Each surface ( $S_j$ )
    for Each intersection line ( $l_i$ ) of loop  $L_i$ 
      Get connected surface cells ( $C_1$  and  $C_2$ ) to  $l_i$ 
      for Each connected surface cell ( $C_k$ )
        if Cell has not been given an orientation,  $O$ 
          Set cell orientation (CW or CCW) of  $C_k$ 
          Flood fill to assign  $O$  to all cells in respective subsurface of  $S_j$ 
        end if
      end for
    end for
  end for
end for

```

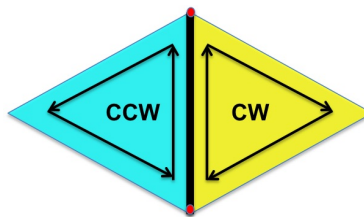


Figure 2.8: Orientation of cells adjacent to an intersection line. Cell to the left is oriented CCW and cell to the right is oriented CW

for the flood fill are the intersection loops. The flood fill algorithm only runs once for every sub-surface—as initiated once the first element in that subsurface is considered. Fig. 2.9 displays the sub-surfaces defined by the intersection between a sphere and a cylinder. The desired Boolean (union, intersection, subtraction) is some combination of the sub-surfaces. The determination of which sub-surfaces to use is dependent on both the intersection loop types and the Boolean operation being performed. For the case when all intersection loops are hard closed loops, the determination is straightforward and follows the description in Fig. 2.9. To demonstrate a more practical application, Fig. 2.10 demonstrates the union of two arterial segments created from 2D segmentation [105] of medical image data.

2.3.2.5 Special Cases: Open Loops and Soft Closed Loops

The Boolean procedure is well defined for water-tight B-Reps since the inside and outside of these surfaces are known. For open surfaces (i.e., there are cell edges with no neighbors), the “inside” is less obvious. Therefore, for ease of understanding and to allow the possibility to perform a Boolean on open surfaces, the outside of an open surface is defined by the direction the normals are facing. The Boolean procedure implemented assumes that the direction of

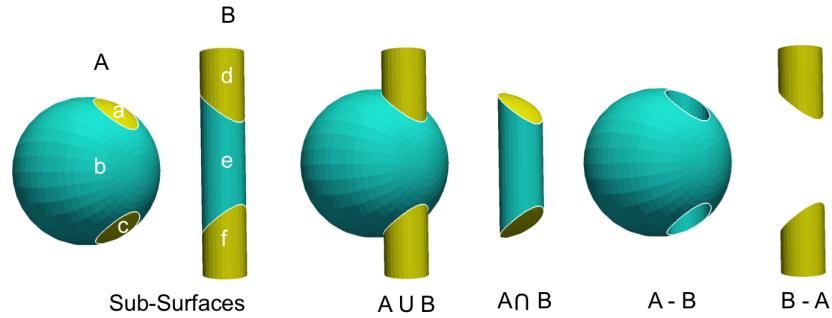


Figure 2.9: A sphere and cylinder intersection. The sphere and cylinder each have three sub-surfaces, a, b, c and d, e, f . The sub-surfaces are colored by their Boolean orientation, yellow is CW and cyan is CCW. $A \cup B = b + d + f$, $A \cap B = a + c + e$, $A - B = b + e$, and $B - A = a + c + d + f$

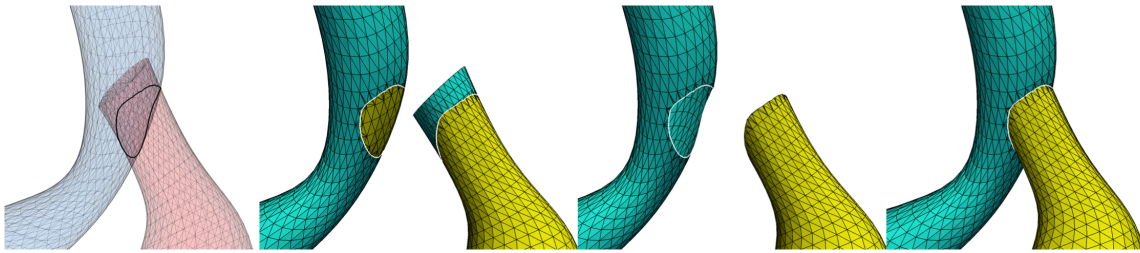


Figure 2.10: The union of two arteries give the combined vascular model. First, the intersection lines are computed. Each surface is re-triangulated and sub-surface orientations are determined. The sub-surfaces corresponding to a union operation are selected and unified to give the output polygonal surfaces.

the normals is the exterior and the opposite direction is the interior. As shown in Fig. 2.7, the Boolean operation on open surfaces can lead to the case of an open intersection loop. This is one of the special cases considered in the Boolean procedure. The other special case arises when surfaces have similar sizes and they intersect at an identical surface point creating soft closed loops.

Open loop intersections. In this scenario there are additional steps required to ensure the correct Boolean is output. The intersection and re-triangulation processes remain the same regardless of intersection type. At the beginning of the Boolean step, the loop intersection types (hard closed, soft closed, open) are characterized. In the case of one open loop, the open loop(s) are sent to the back of the priority queue for sub-surface definition. In this way, any closed or soft closed loops are processed first. The correct sub-surfaces are filled with correct orientations from the closed loops. Subsequently, open loops are processed and any remaining sub-surfaces are given an orientation in the same manner. Alternatively, in cases

where there are *only* open intersection loops, the loops are processed sequentially; however, one surface will contain a sub-surface with no cells. This means two things: (1) the surface's other sub-surface contains all the cells, and (2) this is the sub-surface that needs to be used for the union. In this manner the union is specified to be composed of the larger portion of the sub-surfaces. Fig. 2.11 demonstrates the Boolean outputs in the case of open loop intersections. As shown, when open loop intersections occur, the Boolean is between the actual surfaces, and not the enclosed volume.

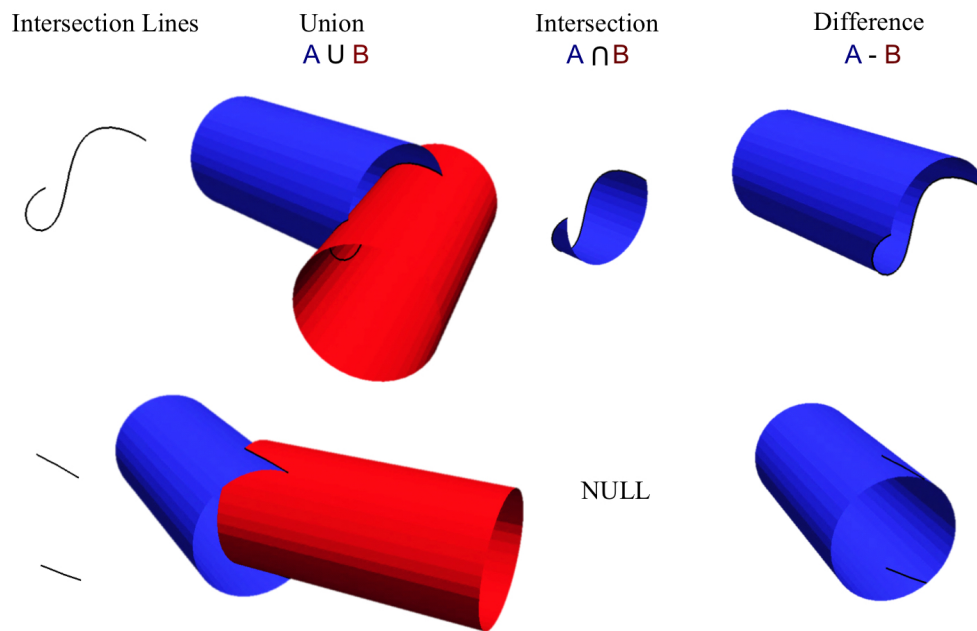


Figure 2.11: Booleans that result in open intersection loops. The top Boolean results in a union that has all of surface B and an intersection comprised of only surface A . The bottom Boolean shows the case with two open intersection loops. The union retains all of both input surfaces and the intersection is a NULL surface.

Soft closed intersection loops. This scenario is less common, but when it occurs there are special procedures followed to ensure the correct Boolean output is achieved. Like the open loop case, the intersection and re-triangulation process are not affected. The specialized procedures come into play during the determination of intersection loop types. A soft closed intersection point (intersection point having more than two attached lines) indicates a soft closed loop. When this point is identified, there are multiple loops possible, and the correct intersection loops must be chosen. To do this, each of the attached lines are taken and the loops completed as potential loop candidates. Note that each loop may contain multiple soft closed intersection points and each possible route must be considered in forming potential

loop candidates. Each possible loop is run through as is done during sub-surface determination (cf. Algorithm 5); however, there are two alterations. First, only cells of one orientation are used to fill regions (CCW in our implementation). Second, the number of regions filled is tracked. In the case that the loop candidate returns only one filled region, the correct loop has been found, and the remaining loop candidates do not need to be tested. After loop identification, the remaining steps are carried out in the same manner as hard closed loops. Fig. 2.12 depicts examples in which a Boolean results in soft closed loop intersections.

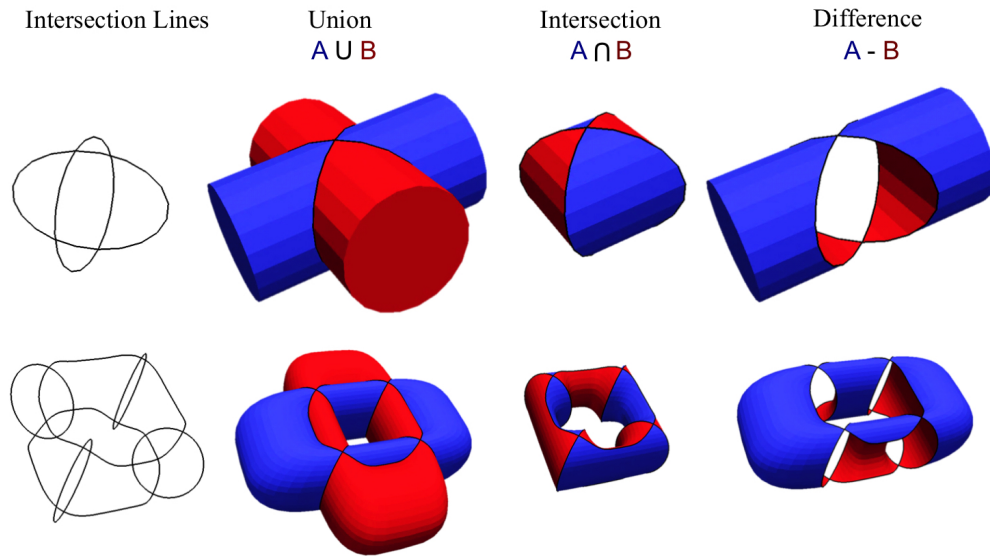


Figure 2.12: Booleans that result in soft closed intersection loops. The top intersection has two soft closed loops while the bottom intersection has four soft closed loops.

2.3.2.6 Performance Improvements

A brief analysis of the run time of Boolean procedures in 3D shows that the implementations described above perform in reasonable time. The worst case run time for a Boolean of polyhedral surfaces is $O(n\lambda^2 + n \log(\Delta))$ time, where n is the number of intersecting cells on surface A and surface B . The density, λ , is defined as the smallest number such that the following holds true: Any ball R intersects λ edges (e) belonging to the surface such that $length(e) \leq diam(R)$. Δ is the spread of surface A and B and is defined as $\Delta = \frac{D}{d}$, where D is the size of the smallest quadrant of the binary space partition (BSP) built on the cells. d is the diameter of the smallest ball that intersects $k + 2N_{SD}\lambda + 1$ edges of the surface. k is the maximum number of cells for BSP tree quadrant and N_{SD} is the number of spatial dimensions (in this case 3). See [106] for additional details on BSP trees and their use on polyhedral surfaces.

Two performance tests are displayed. For the first test (Fig. 2.13), there is only one intersection loop and the triangles on both input surfaces are of low aspect ratio (close to

equilateral). In the second test (Fig. 2.13), there are two intersection loops and the triangles have high aspect ratios. To test the scaling of the algorithm, the cells for each surface were repeatedly subdivided as to increase the number of intersecting cells. In order to compute the worst case runtime (solid curve with *), the following assumptions were made. First, upon each subdivision, Δ is assumed to increase by a factor of 2. Second, upon each subdivision, for test 1 (Fig. 2.13), λ was assumed to increase by a factor of two. For test 2 (Fig. 2.13), because of the high aspect triangles used, λ was assumed to increase by a factor of 4 upon each subdivision. As shown in these figures, both prior and new implementations perform better than the assumed upper limit. However, the new implementation scales significantly better than the Boolean implementation in VTK version 6.2.0 [103] with an increasing number of intersecting edges. This performance improvement is realized for both test cases. In particular, for surfaces containing triangles of low aspect ratios, which is often the case in model construction, the new implementation runs close to two orders of magnitude faster for Booleans when a large number of intersections are present.

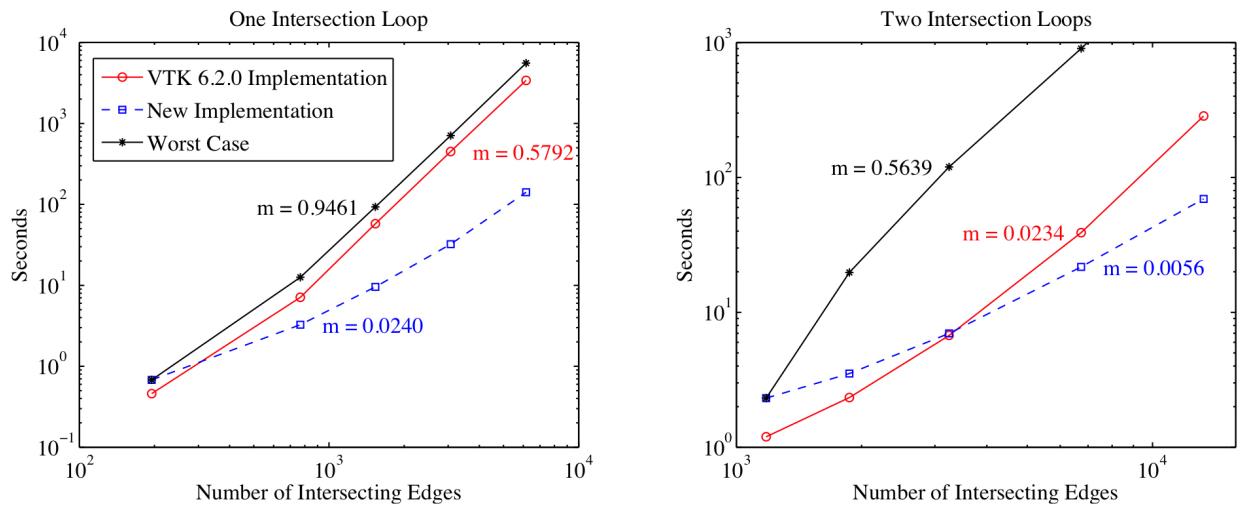


Figure 2.13: Performance plots for Boolean operations with one and two intersection loops. The developed implementation is compared to the VTK version 6.2.0 [103] implementation and the worst case performance for a discrete Boolean in 3D.

2.3.3 Uniting Multiple Vessels

In many anatomic models, the final model for CFD simulation typically is comprised of a large number of vessels. In order to receive the full vascular network, a series of Boolean procedures need to be performed. The vascular network can be as complicated as the pulmonary artery network typically consisting of over 100 vessels. When building a model such as this in SimVascular, users sometimes may not consider the order in which the vessels are constructed. This sometimes leads to the case in which two consecutive branches do not

intersect but are both part of the same final model. For this reason, it is desirable to have this procedure consisting of multiple polydata union operations to be input order independent. This along with passing intersection information from each individual union required a custom method to be designed and implemented.

The first step in creating the entire vascular network requires a pre-processing step involving the construction of an intersection table. **Intersection Table** - The creation of the intersection table starts by calculating the bounding boxes for each of the input surfaces. The intersection of the bounding boxes quickly indicates whether or not the surfaces may intersect. Obviously, if the bounding boxes do not intersect, it is impossible for the surfaces to intersect; however, if they do intersect, it is possible that they intersect and that will be marked in the Intersection Table. The Intersection Table is a symmetric matrix consisting of ones and negative ones. Each surface has a corresponding row and column. Where the surfaces' bounding boxes intersect, a one fills the entry of the table or matrix. If they do not intersect, the entry in the matrix or table is negative one. Obviously, the union should not be computed between an object and itself, so the diagonal of the matrix is populated with negative ones.

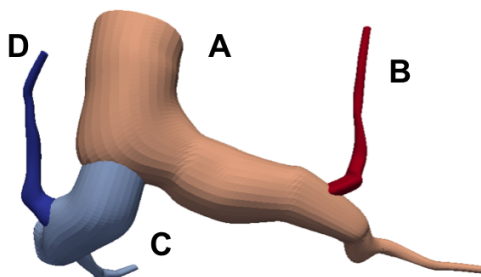


Figure 2.14: Intersection between multiple vessels in a vascular network

	A	B	C	D
A	-1	1	1	-1
B	1	-1	-1	-1
C	1	-1	-1	1
D	-1	-1	1	-1

The procedure then uses this intersection table as a guideline for performing the full Union of the vessels. The first row of the matrix is taken and the Union with each of the possibly intersecting vessels is computed. When this happens, the symmetric location in the matrix is converted to a negative one as well as the entire column (The surface does not need to be processed again). In addition, the row of the newly added vessels is put in a priority queue. This algorithm runs until the queue is empty, indicating all intersecting vessels have been combined. In the case that a vessel does not intersect, it's row is not crossed out and it is not added to the priority queue.

There are also a couple of special cases here that are dealt with according to user input. It is possible that a vessel does not intersect any of the vessels in the network at all. In this case, the user can specify whether to return nothing, everything, or the first group of intersecting vessels. These procedures give a simple and efficient method for providing the Union of a large network of geometries that is independent of order.

2.3.4 Surface Manipulation Operations

A surface output from a Boolean operation, such as the one described in Section 2.3.2, typically has sharp angles where the two input surfaces are united. Also in many practical applications, discrete B-Rep surfaces contain some amount of undesirable roughness since they are often generated from image data where a parametric representation is not available. For either case, surface manipulation operations, which move points or alter point connectivity, are necessary to improve the final surface for its subsequent use.

Before describing these methods and how they are used to improve surface quality, it is important to define surface *quality*. *Quality* can be any sort of metric defined on the surface or surface elements that can be altered through physical manipulation. Typical quality metrics include area, element jacobian, aspect ratio, and maximum or minimum element angles. For the purpose of this work, improving quality involves both achieving better element aspect ratios as well as decreasing the angle between adjacent element normal vectors. Element aspect ratio is simply the ratio of the largest edge to the smallest edge in an element; an aspect ratio of 1.0 is the best possible aspect ratio. Fig. 2.15 compares the aspect ratio of a “poor quality” surface with that of a “high quality” surface, while Fig. 2.16 demonstrates the difference in element normal vectors for a “poor quality” and “high quality” surface.

The most commonly used surface manipulation operations for discrete triangulated surfaces are (1) smoothing, (2) decimation (3) subdivision, and (4) remeshing. Smoothing attempts to lessen the angle difference between adjacent cells in the mesh. These smoothing operations span from basic Laplacian smoothing [107] to more complicated methods using weighted values such as Taubin smoothing [108] (Section 2.3.4.1). As part of this doctoral work, a constrained smoothing method that seeks to retain volume was developed. Decimation is the process of decreasing the number of triangles to represent the surface (Section 2.3.4.2). Decimation is often desirable since it can decrease the number of poor quality elements and decrease the complexity of the model. While decreasing the number of facets, decimation also attempts to introduce the smallest amount of error possible. There are different ways in which to define error and simplify the surface [109]. The methods developed as part of this dissertation use a quadric metric [110]. Subdivision works opposite to decimation, in that it introduces more cells into an existing surface (Section 2.3.4.3). Like most surface operations, subdivision also has multiple implementation techniques [111]. The surface subdivision techniques developed as local operations for this dissertation are butterfly, loop, and linear subdivision. Remeshing typically consists of a combination of local cell changes to systematically improve surface and element quality (Section 2.3.4.4). These local cell changes are usually operations such as edge splits and edge removals.

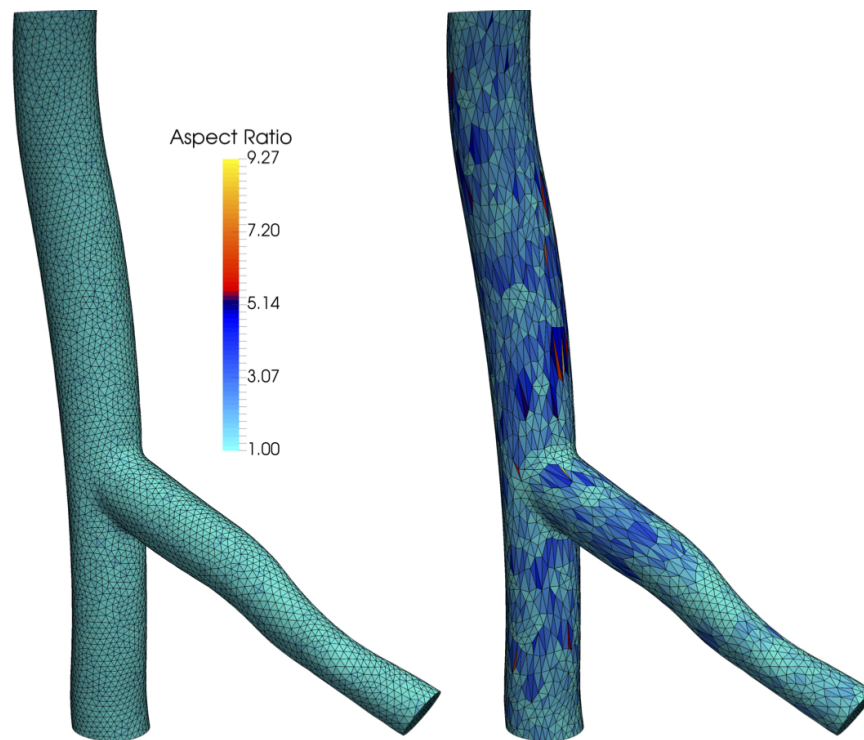


Figure 2.15: (left) The “high” quality simple bifurcation is triangulated with triangles of very low aspect ratios which are all nearly 1.0. (right) The “low” quality model contains triangles of higher aspect ratios.

In most applications, entire geometries do not need to be altered; thus, localized versions of the presented discrete model operations were implemented. Often only a specific, localized portion of a model needs smoothing. Localized methods were created for all mentioned surface manipulation operations by altering existing VTK filters to perform these operations. The exception to this was that an entirely new smoothing method was created, as described in Section 2.3.4.1. To be able to perform localized methods, node and cell selection procedures were created to identify portions of a model for manipulation (Fig. 2.17). The following four methods have been developed as part of a user interface to be able to interactively select regions: (1) A surface can be comprised of multiple faces and certain faces can be selected. (2) Cells can be painted on a surface which designate them as cells to be used. (3) A sphere with a specified radius and location can be placed and cells or nodes encapsulated within are selected. (4) With specialized algorithms, the interface between two faces can be found and then a sphere with a specified radius can be selected around this interface for selection. The above methods were customized to suit this application of developing discrete solid models for use with computational fluid dynamics (CFD).

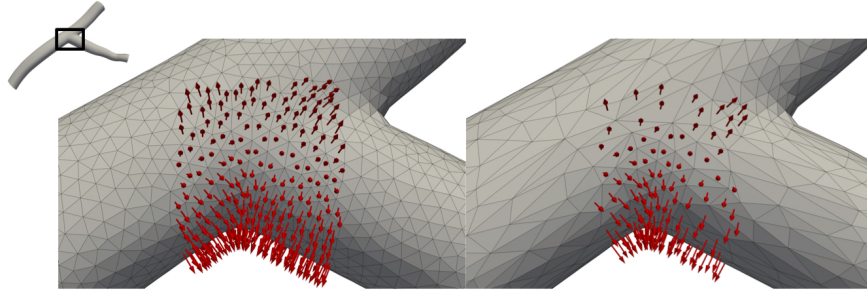


Figure 2.16: (left) The normals at a bifurcation differ less between triangles of a “high” quality surface. (right) The normals have a larger difference between triangles on the “poor” quality surface.

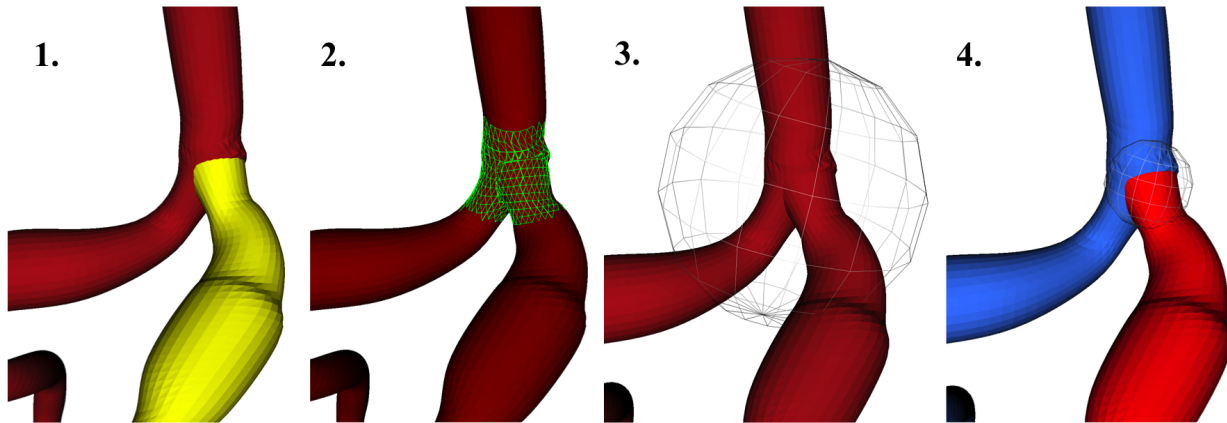


Figure 2.17: Four different cell selection types developed: (1) Any face or combination of faces can be selected. (2) Cells on the surface can be painted for selection. (3) A spherical region can be chosen. (4) The interface between two faces (red and blue) can be found and a sphere region with a specified radius can be selected around the interface.

2.3.4.1 Smoothing

Laplacian smoothing is often used on discrete geometries to decrease roughness; however smoothing can also degrade the accuracy of the representation. Of particular concern is that smoothing the B-Rep of an enclosed volume shrinks a geometry by nature. This occurs because a typical Laplacian smoothing assigns the new location of a point to be an average of the points sharing an edge with the specific point, which for one iteration is

$$\mathbf{x}_i = \frac{1}{n} \sum_{j=0}^n \mathbf{x}_j \quad (2.3)$$

where n is the number of nodes sharing an edge with node i . The Laplacian operator is defined as,

$$L(\underline{\mathbf{x}}_i) = \frac{1}{n} \sum_{j=0}^n \underline{\mathbf{x}}_j - \underline{\mathbf{x}}_i \quad (2.4)$$

Since all points are pulled toward their neighbors, as the number of smoothing iterations increases, each point will be pulled toward every other point in the mesh, and the surface will collapse inward. This is particularly a problem where the mesh object represents features of different scales. In such cases, smoothing necessary to decrease roughness in locations of large scale features degrades the representation of small scale features needing to be preserved. For example, in vascular model construction, the shrinkage effect on large vessels may be insignificant over a few iterations, but the effect on small vessels may alter vessel area substantially, which can have dramatic consequences once the model is used for simulation.

A smoothing method that attempts to counteract the global mesh shrinkage is Taubin smoothing [108]. In Taubin smoothing, a Laplacian mesh smoothing step is followed by an “inflation step” step where a Laplacian mesh smoothing step is performed with a negative weight,

$$\underline{\mathbf{x}}_{i+1} \leftarrow \underline{\mathbf{x}}_i + \lambda L(\underline{\mathbf{x}}_i) \quad (2.5)$$

where, λ , is a weight that is applied to the Laplacian operator which is positive for one step and negative for the next step.

A method that follows the same idea of counteracting the mesh shrinkage was developed. This method minimizes the error between the original mesh and the Laplacian smoothed mesh. Methods of this nature have recently been developed in work on computing watershed ridges [112]. The location of a smoothed point on the surface becomes the minimum of two equations in an optimization problem. Specifically, for each iteration, the coordinates of a new point $\underline{\mathbf{x}}_i$ are described by the equations

$$\underline{\mathbf{x}}_i - (\underline{\mathbf{x}}_{\text{original}} + \underline{\mathbf{w}}) = 0 \quad \text{where} \quad \underline{\mathbf{w}} = \|\underline{\mathbf{x}}_i - \underline{\mathbf{x}}_{\text{original}}\| * w_{user} \quad (2.6)$$

$$\underline{\mathbf{x}}_i - \frac{1}{n} \sum_{j=0}^n \underline{\mathbf{x}}_j = 0 \quad (2.7)$$

where w_{user} is a user defined weighting between 0 and 1 used to penalize deviations from the original representation. These equations can be applied at every point to obtain the following matrix equations,

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & 1 & 0 & \ddots & & & & \vdots \\ 0 & 0 & 1 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & 1 & 0 & 0 \\ \vdots & & & & & \ddots & 0 & 1 & 0 \\ 0 & \dots & \dots & \dots & \dots & 0 & 0 & 0 & 1 \end{bmatrix}}_{A_1} \begin{bmatrix} x_0^x \\ x_0^y \\ x_0^z \\ \vdots \\ \vdots \\ x_m^x \\ x_m^y \\ x_m^z \end{bmatrix} = \underbrace{\begin{bmatrix} x_{original}^x + w_0 \\ x_{original}^y + w_0 \\ x_{original}^z + w_0 \\ \vdots \\ \vdots \\ x_{original}^x + w_m \\ x_{original}^y + w_m \\ x_{original}^z + w_m \end{bmatrix}}_{b_1} \tag{2.8}$$

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & -\frac{1}{n} & \dots & \dots & \dots & \dots \\ 0 & 1 & 0 & 0 & -\frac{1}{n} & \dots & \dots & \dots \\ 0 & 0 & 1 & 0 & 0 & -\frac{1}{n} & \dots & \dots \end{bmatrix}}_{A_2} \begin{bmatrix} x_0^x \\ x_0^y \\ x_0^z \\ \vdots \\ \vdots \\ \vdots \\ x_m^x \\ x_m^y \\ x_m^z \end{bmatrix} = \underbrace{\begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}}_{b_2} \tag{2.9}$$

Matrix A_2 is commonly known as the Laplace-Beltrami operator. Since each point's coordinates has 3 components, the total degrees of freedom of this system is $3 * m$, where m is the number of points on the surface mesh. The weighting is only applicable if the smoothing is done for more than one iteration. In many scenarios, it is desirable to have certain points that do not move at all. This can be implemented by setting the weighting for these points to 0, or by leaving these points out of Equation (2.9). More generally, the weighting values can be varied among the points to locally constrain smoothing in a spatially varying manner.

To solve the above matrix equations (2.8) and (2.9), these matrices are concatenated vertically $A = [A_1; A_2]$ and $b = [b_1; b_2]$ and the extended system $Ax = b$ is solved using the conjugate gradient method. Namely, the conjugate system $A^T Ax = A^T b$ is solved, which corresponds to the minimization of $f(x) = \langle Ax - b, Ax - b \rangle$. That is, when the quadratic function $f(x)$ is minimum, the gradient is equal to zero

$$\nabla_x f = 2A^T(Ax - b) = 0 . \tag{2.10}$$

Fig. 2.18 shows the effect of the constrained smoothing on a select portion of a vascular model. When combined with localized decimation operations, the blending between the vessels increases. For a full blending of the branching vessel, these operations are also

combined with decimation to give a smooth, natural transition between the vessels. Fig. 2.19 displays the effect of using constrained and non-constrained smoothing, demonstrating that constrained smoothing is able to create a smooth transition while maintaining otherwise high fidelity with the original volume. Fig. 2.20 displays the percentage of the original volume for the constrained smoothing as a function of the user-defined weighting (penalty) parameter. The weighting maintains, and in some cases, increases the original surface area and volume of the surface. As shown, the larger the number of smooth iterations, the lower the user weighting should be to maintain the original volume.

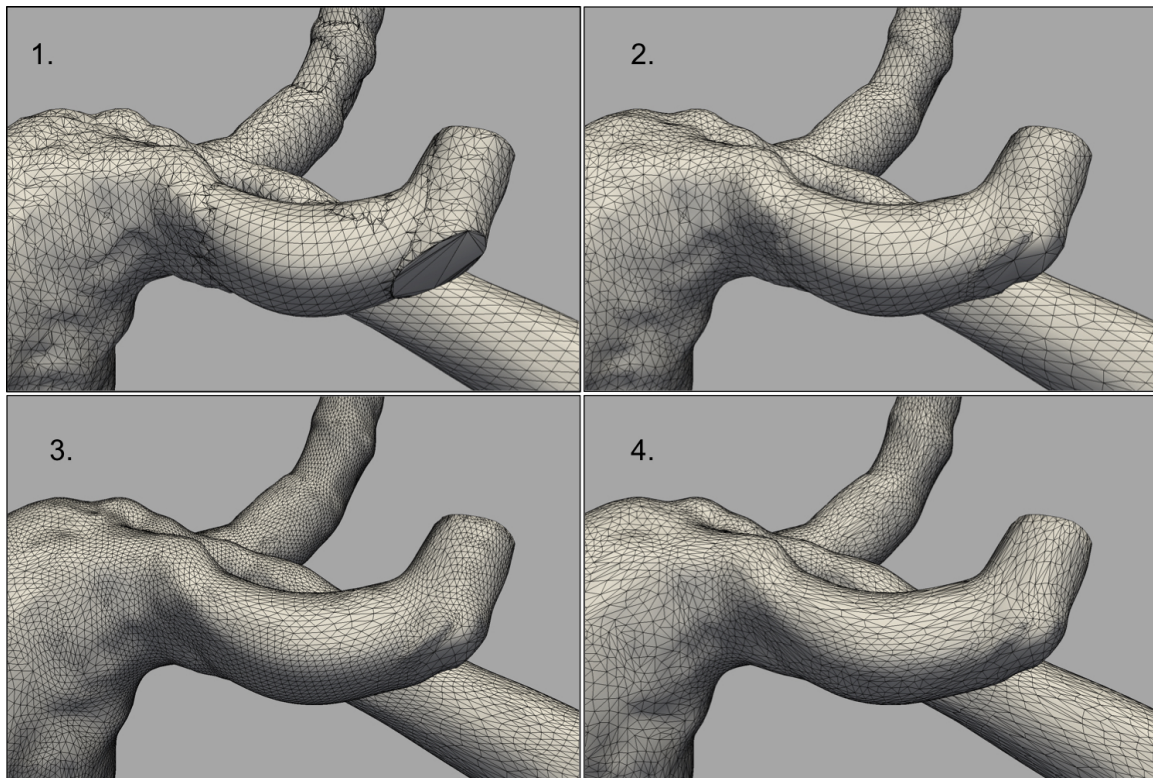


Figure 2.18: The constrained smoothing is used on a model of a cerebral aneurysm. Combined with localized subdivision and decimation, the overall shape is retained while removing high frequency noise and undesired roughness. The four panels show the progression of the smoothing, refinement and decimation process.

2.3.4.2 Decimation

Surface mesh decimation involves removing elements from the discrete surface in a systematic manner in order to reduce mesh complexity. Decimation procedures typically follow four steps in which (1) the surface and point topologies are evaluated, (2) an error is estimated for removal of points or elements, (3) points or elements that provide the minimal amount of

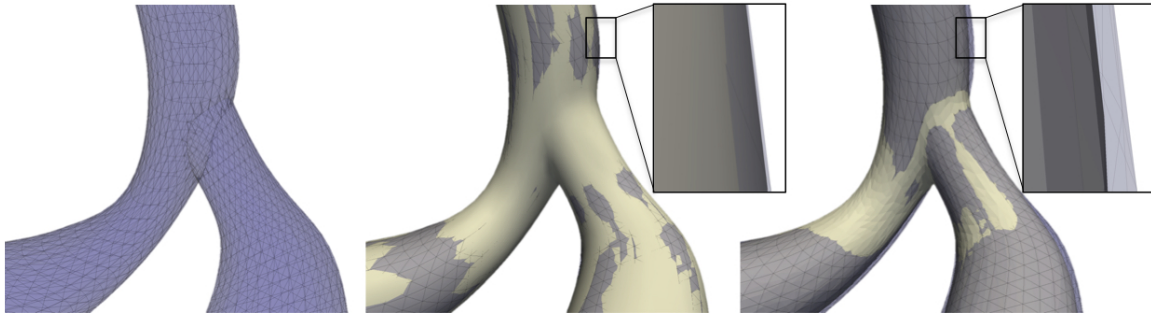


Figure 2.19: The original surface mesh (left) is smoothed using constrained smoothing. The constrained smoothed model (middle) is displayed along with the original mesh demonstrating the ability to create a smooth transition while retaining the original volume. Application of regular Laplacian smoothing (right) is not able to maintain the original volume.

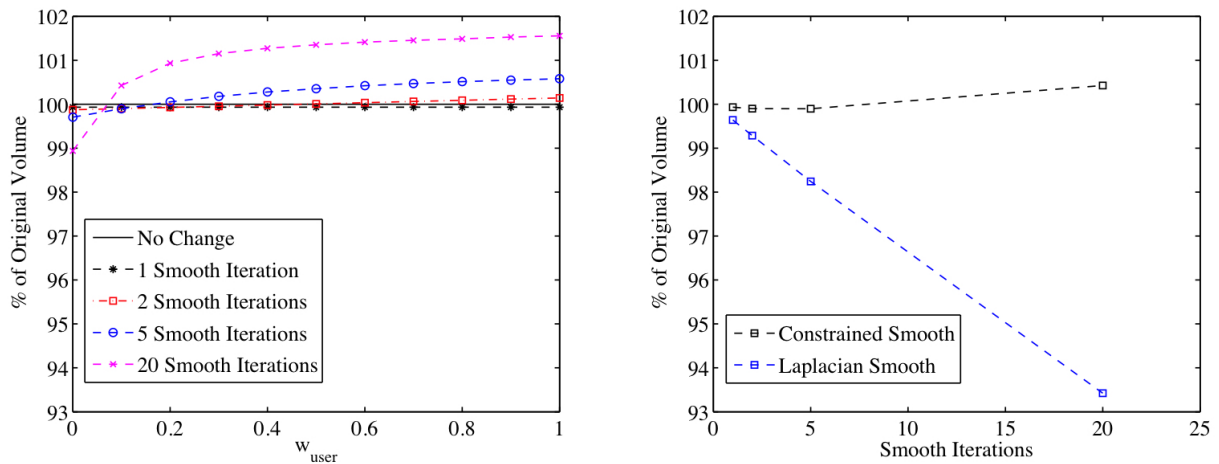


Figure 2.20: Left: The percentage of the original volume when constrained smoothing is performed for 1, 2, 5, and 20 iterations. Each case was attempted for user weightings from 0.1 to 1.0. Right: Constrained smoothing using a weighting of 0.1 compared to the regular Laplacian smoothing.

error are removed, and (4) the mesh is re-triangulated or repaired where points or elements have been removed. Average, median, and quadric error methods are some of the different decimation algorithms. Quadric error algorithms have been proven to provide high quality results, and VTK has implemented this algorithm. A localized version of this algorithm was implemented (Fig. 2.21).

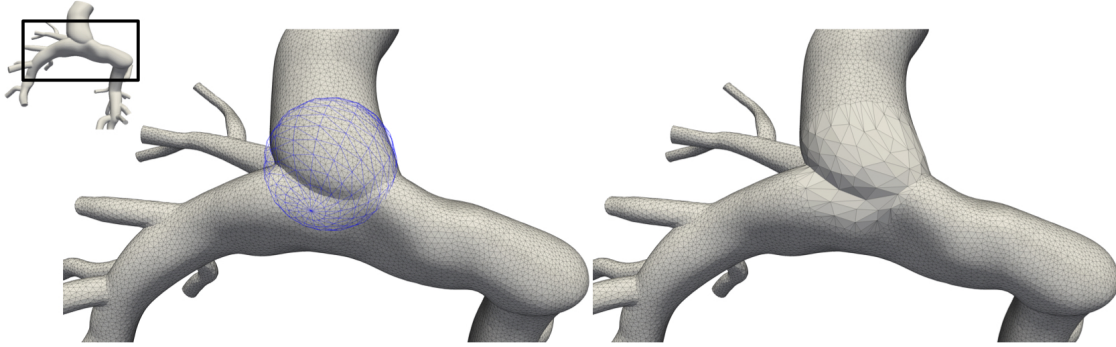


Figure 2.21: Localized decimation allows the remove of triangles in a specific region.

2.3.4.3 Subdivision

Subdivision algorithms introduce complexity to the surface, but in doing so, help to improve the quality of the surface. Linear subdivision is the most simple of these algorithms and a point is added at the midpoint of each edge. This increases the complexity of the surface by a factor of 4 as each triangle is converted into 4 smaller triangles. Although this increases the number of elements on the surface, it does little in the way of improving the smoothness of the surface. More advanced subdivision schemes, such as interpolating and approximating schemes, provide better results. Interpolating subdivision schemes generate new points through higher order interpolation, while approximating subdivision both add new points and then average both existing and new points. The most commonly used interpolating subdivision scheme for triangulated surfaces is butterfly subdivision [113]. Butterfly subdivision uses the scheme demonstrated by Fig. 2.22 and Equation 2.11.

$$n_0 = (p_0 + p_1) + \omega(p_2 + p_3) - \frac{\omega}{2}(p_4 + p_5 + p_6 + p_7) \quad (2.11)$$

The most commonly used approximating subdivision scheme for triangulated surfaces is loop subdivision [113]. Loop subdivision uses the scheme demonstrated in Fig. 2.23 and Equation 2.12.

$$n_0 = \frac{3}{8}(p_0 + p_1) + \frac{1}{8}(p_2 + p_6) \quad (2.12)$$

$$p'_0 = \alpha_n p_0 + \frac{(1 - \alpha_n)}{n} \sum_{j=1}^n p_j \quad (2.13)$$

where $\alpha_n = 3/8 + (3/8 + 1/4 \cos((2\pi)/n))^2$ and n is the number of connected vertices to p_0 .

These algorithms work on the theory that there is an infinitely smooth limit surface and every subdivision iteration brings the surface closer to the ideally smooth mesh. This

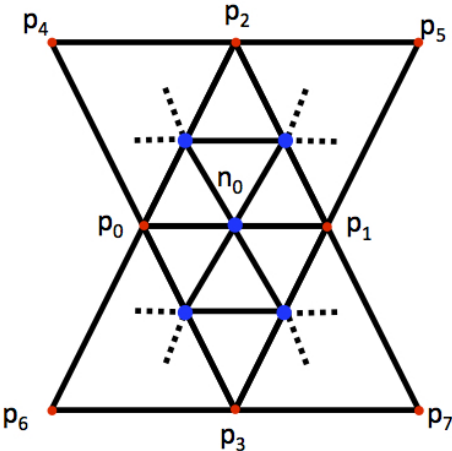


Figure 2.22: The butterfly subdivision is an interpolating subdivision scheme that approaches a G^1 continuous surface.

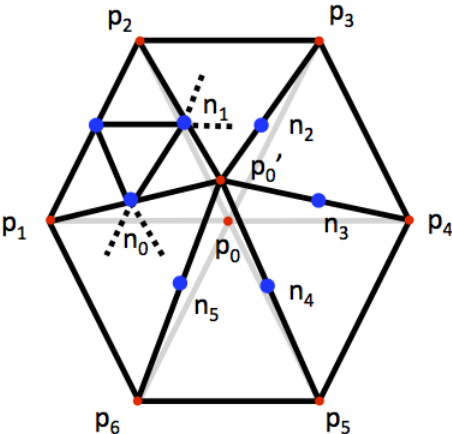


Figure 2.23: The loop subdivision scheme is an approximating subdivision scheme that approaches a G^2 continuous surface.

theory is founded in the work of splines and the schemes are typically derived from different types of splines. There are different schemes for different surface representations. Where loop subdivision is the most commonly used subdivision algorithm for triangulated surfaces, Catmull-Clark [113] is the most commonly used for quadrilateral meshes. In addition, there is a whole category of surfaces referred to as subdivision surfaces in which the object is stored as a minimal piece-wise linear complex with a specific subdivision rule to achieve the infinitely smooth surface. These surfaces are now widely used in the animation industry as it is much more efficient to store large animated scenes with minimal data and apply the subdivision rule to render and visualize the surface. Linear, loop, and butterfly subdivision algorithms

exist in VTK, and localized versions of these algorithms were implemented (Fig. 2.24).

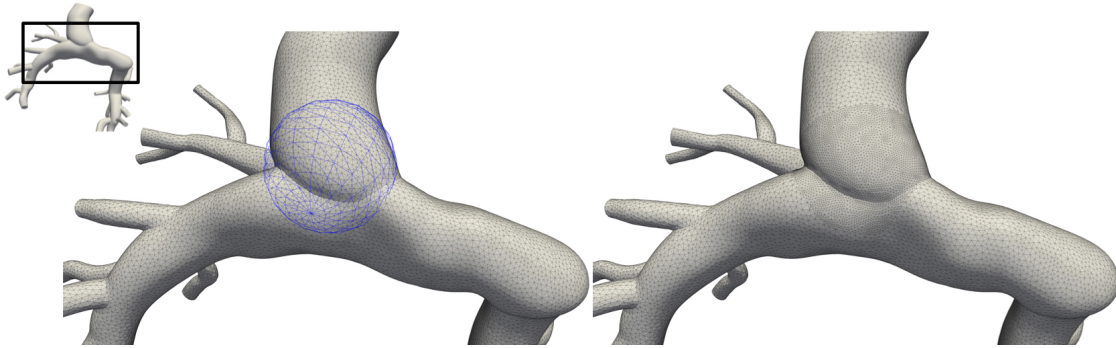


Figure 2.24: Localized subdivision allows surface quality improvement in a localized region.

2.3.4.4 Remeshing

Remeshing algorithms vary widely in implementations, but typically combine a series of edge splits, edge collapses, and edge flips. An edge split involves placing a new point at the midpoint of an edge and splitting a triangle into two new triangles by connecting the new point with the triangle point not on the edge (Fig. 2.25(a)).

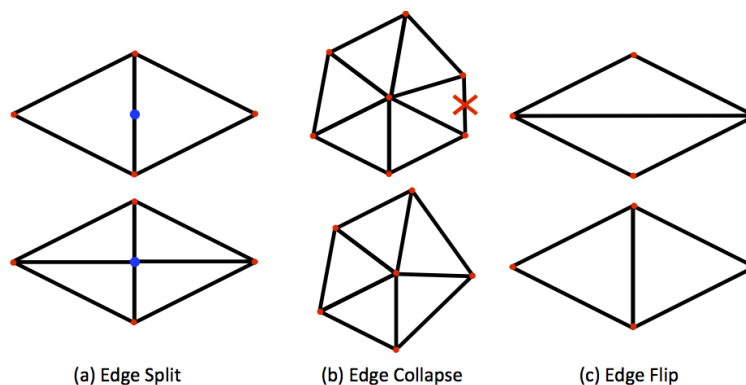


Figure 2.25: Three common edge operations performed when remeshing a triangulated surface involve (a) edge splits, (b) edge collapses, and (c) edge flips.

An edge collapse involves removing an entire edge and in essence removing triangles as well. The two points of the edge are brought together and merged into one point and the triangles containing the edge are squashed and removed (Fig. 2.25(b)). An edge flip involves flipping the edges of two adjacent triangles. The edge between the two triangles is removed, but the points remain in the same location. The four points of the two triangles form a quadrilateral and a new edge is placed between them combining the points on the triangles

that were not involved with the removed edge. An edge flip only requires a connectivity change to the mesh and not a physical point change (Fig. 2.25(c)). These operations are combined to produce higher quality elements and surfaces. Mesh size factors can be prescribed to achieve locally refined surface meshes in which mesh sizes can be adapted to resolve features of different size scales. Fig. 2.26 shows the result of a uniform mesh size remesh while Fig. 2.27 shows the result of a remesh with a localized refined region in the area of a stenosis. Robust tools for open-source remeshing exist, thus a new method was not implemented into SimVascular. Surface remeshing was developed using a combination of VMTK, MMG, and custom code (Chapter 3).

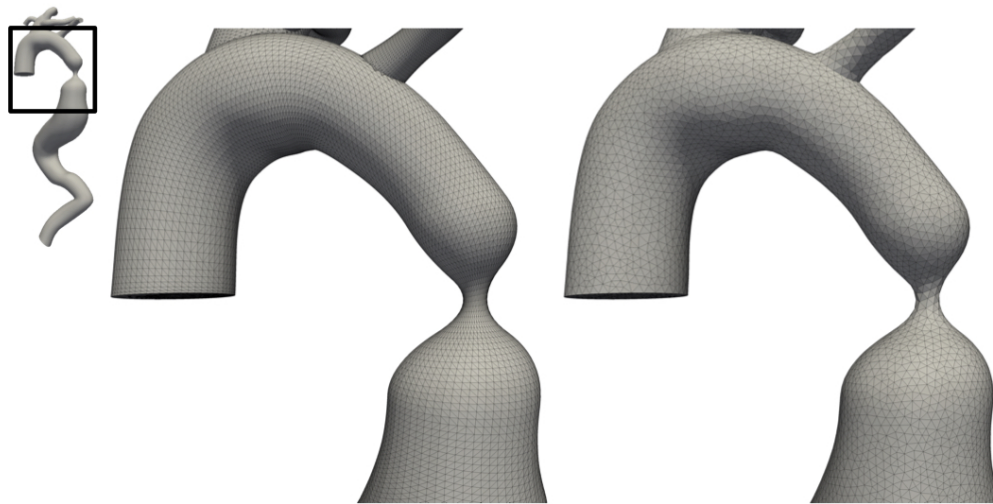


Figure 2.26: Uniform remeshing provides a higher quality triangulation.

2.3.4.5 Combining Surface Manipulation Techniques

In patient-specific vascular modeling, the main challenge comes in attempting to achieve a smooth, realistic lumen or vascular wall. The lofted 2D segmentation technique results in very sharp junctions where vessels have been combined with a union operation. The direct 3D segmentation technique creates a surface that has a voxelized appearance due to the discrete nature of image data. Both of these require special treatment in order to prepare the surface for finite element meshing and analysis. In addition, it is typically important to preserve certain areas of the mesh, especially the artificial truncation boundaries where inflow and outflow boundary conditions will be applied for blood flow simulation. Local surface operations were developed in order to make this possible and allow for mesh operations to be performed on a subset of the mesh. Though there are a variety of operations that can be done and in many different combinations, the following set of operations typical results in a smooth vascular surface with smoothed junctions.

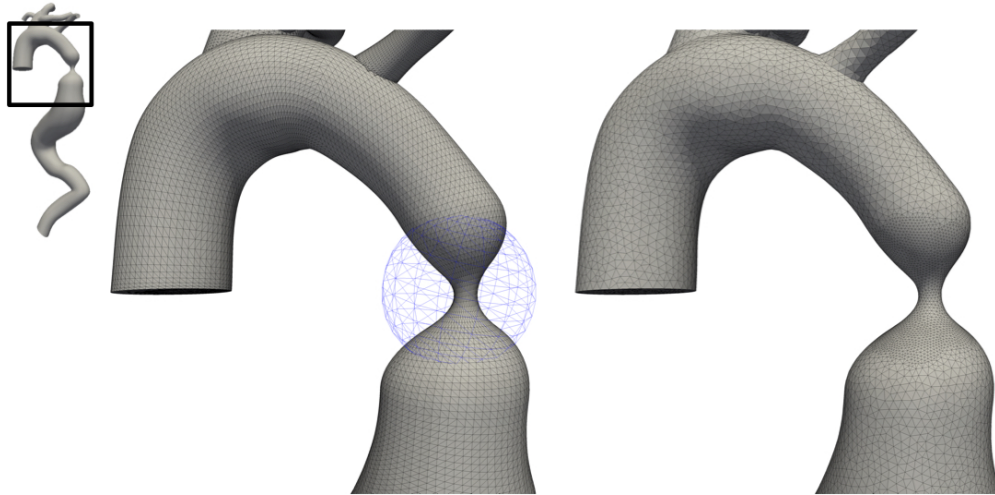


Figure 2.27: Regional refinement provides better resolution in localized areas where it may be necessary to capture small size scale features.

- Global surface remeshing - This removes any poor quality elements on the surface so succeeding operations aren't effected by skewed or thin elements.
- Local constrained surface smoothing - As discussed in section 2.3.4.1, many typical smoothing operations, such as Laplacian smoothing, result in mesh shrinkage. Alternatively, the custom constrained smoothing method is used.
- Local surface decimation - This reduces the number of elements on the surface. In doing this, the complexity of the surface is reduced, allowing the succeeding operations to have a larger effect on the surface.
- Local constrained surface smoothing - Another surface smoothing operation to relocate nodes to a more ideal position.
- Local surface loop subdivision - This adds new nodes on the surface through the loop subdivision scheme. It inherently increases smoothness of the surface by strategic placing of new nodes.
- Global surface remeshing - After these operations to smooth the junction between vessels or smooth out a voxelized surface, a global surface remeshing helps to provide a more uniform surface mesh.

2.4 Parametric Modeling

As described in section 2.2, it is also possible to obtain a parametric model using the lofted 2D segmentation approach. The parametric representation of choice is the non-uniform rational

B-spline. Before describing vascular modeling with NURBS, the underlying definition of NURBS is discussed.

2.4.1 NURBS

The industry standard for CAD is the non-uniform rational B-spline (NURBS) surface. CAD developed out of the automotive industry when Bezier and Faget De Casteljaou independently used Bernstein polynomials to generate curves and surfaces for automobiles [114, 115]. Vesprille expanded on the work of Bezier and De Casteljaou by using the concept of splines to introduce rational B-splines, or what are now known as NURBS [116]. NURBS are spline-based curves and surfaces based on piecewise and parametric polynomial basis functions. A basic foundation for understanding NURBS is presented, starting with the definition of the simplest parametric polynomial function - the power basis curve. From the power basis curve, small steps are taken to build up to the full definition of NURBS.

Power Basis Curves A power basis curve is simply the summation of polynomials in parametric form:

$$C(u) = (x(u), y(u), z(u)) = \sum_{i=0}^n P_i u^i, \quad 0 \leq u \leq 1, \tag{2.14}$$

where u is the parametric variable. The power curve, $C(u)$, depends on $n + 1$ points, P_i , which control the behavior of the curve. Each point has some influence on the curve at each parametric location along u . The P_i are typically called **control points**, as they are not actually on the geometric curve, but physically control the shape.

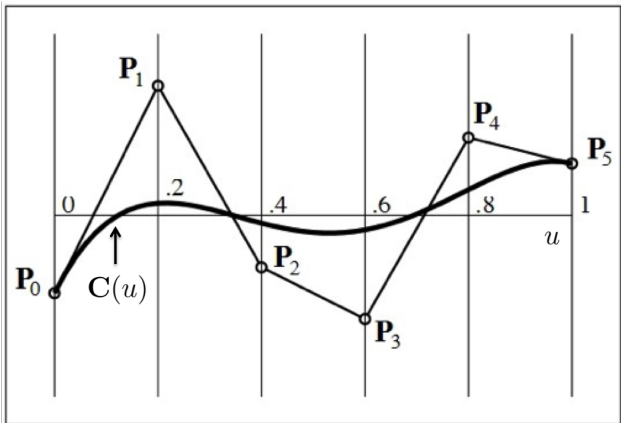


Figure 2.28: A power basis curve. Each control point, P_i , influences the curve. The lines connecting the control points are typically called the control mesh.

B-spline Curves Power basis curves approximate simple curves easily; however, it quickly becomes difficult to approximate complex geometries. In addition, as the number of control points increases, so does the degree of the curve. Increasing the degree of the curve with the addition of new control points is largely unnecessary and can lead to high frequency oscillations when trying to represent a curve of lower inherent degree. To have local or piecewise control of the curve, only a restricted set of control points should effect the shape. This local control is made possible with **B-spline basis functions**. The zero order B-spline basis functions are step functions,

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1}, \\ 0 & \text{else,} \end{cases} \quad (2.15)$$

where again u is the parametric variable and $0 \leq u \leq 1$. Higher order basis functions are a linear combination of the zero degree basis functions (Equation 2.16):

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u), \quad (2.16)$$

where p is the degree of the basis function. This recursion is known as the Cox-de boor recursion [117, 118]. The basis functions in Equation 2.16 require a vector of parametric values u_i on which the basis functions are defined,

$$\mathbf{u} = \{u_1, u_2, \dots, u_{n+p+2}\}. \quad (2.17)$$

Equation 2.17 is a vector of parametric values called the **knot span**. Figure 2.29 shows the zero, first, and second degree basis functions for a simple knot span. Although the knot span does not provide much intuition physically, it is what determines which basis functions locally affect the curve or surface. The knot span separates the parametric space of the B-spline functions into discrete pieces. Thus, when isogeometric analysis (IGA) is discussed in Chapter 4, it is helpful to understand that the knot span is essentially what separates elements.

The B-spline curve then has the following representation:

$$\mathbf{C}(u) = \sum_{i=0}^n \mathbf{P}_i N_{i,p}(u), \quad (2.18)$$

where again \mathbf{P}_i are the $n + 1$ control points of the curve, p is the degree of the curve, and $N_{i,p}(u)$ are the basis functions defined in Equation 2.16.

B-spline Surfaces Because a B-spline curve is a parametric curve, it is trivial to change the dimension of the curve by simply changing the dimension of the control points. However, it can be tough, for example, to control a shape in 3D space with only one parametric variable. Thus, a new parametric variable can be incorporated to remedy this difficulty.

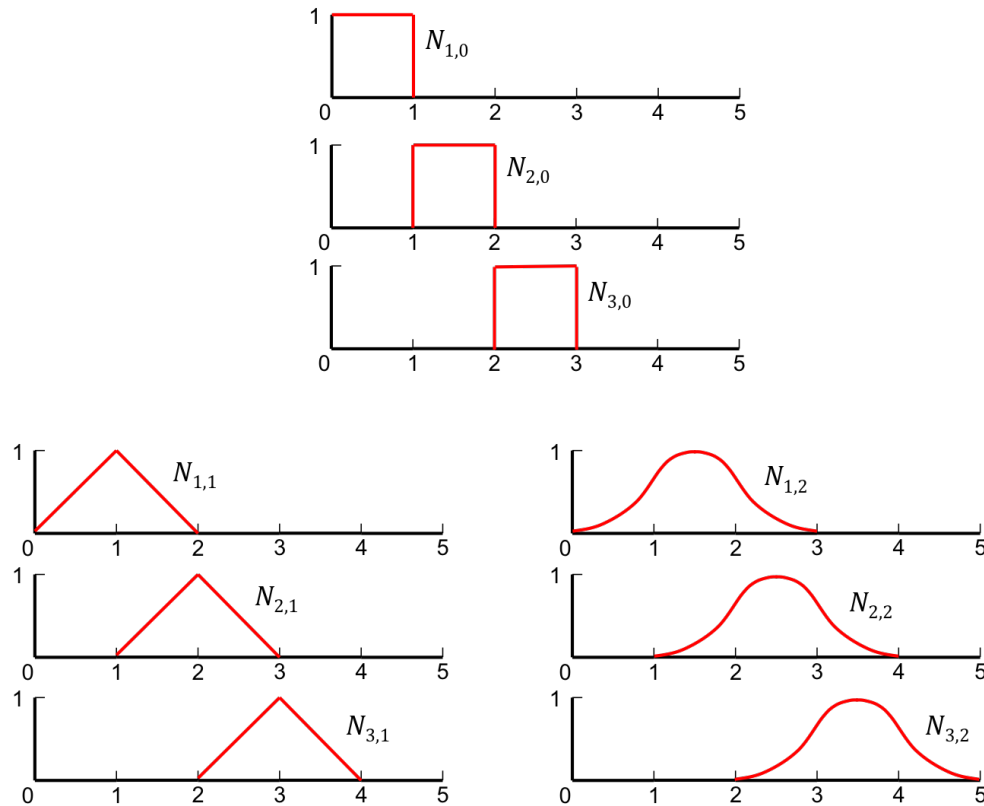


Figure 2.29: Zero (top), first (bottom left), and second (bottom right) degree basis functions for a knot span $\mathbf{u} = [0, 1, 2, 3, \dots]$.

B-spline surfaces are the extension of B-spline curves to two parametric directions (a.k.a bivariate splines):

$$\mathbf{S}(u, v) = \sum_{i=0}^n \sum_{j=0}^m \mathbf{P}_{i,j} N_{i,p}(u) N_{j,q}(v), \tag{2.19}$$

where $m + 1$ is the number of control points in the second parametric direction, v , and q is the degree of the surface in v . The control points, $\mathbf{P}_{i,j}$, are now essentially a 2D structured grid of points.

NURBS Surfaces With B-spline surfaces, it is now easier to represent a wider range of geometries in 3D space; however, it is impossible to represent spheres, ellipses, cones, and other surfaces. Rational B-splines account for this shortcoming by incorporating a weighting factor, $w_{i,j}$, which are a grid of scalar weights corresponding to the control points, $\mathbf{P}_{i,j}$. In addition to this new weighting factor, Equation 5.28 is also normalized by the summation of the entire surface. This normalization ensures that the basis functions satisfy the partition

of unity. The partition of unity simply means that the basis functions for each parametric value sum to 1, and it is important because it guarantees a smooth global curve or surface,

$$\mathbf{S}(u, v) = \frac{\sum_{i=0}^n \sum_{j=0}^m \mathbf{P}_{i,j} N_{i,p}(u) N_{j,q}(v) w_{i,j}}{\sum_{k=0}^n \sum_{l=0}^m \mathbf{P}_{k,l} N_{k,p}(u) N_{l,q}(v) w_{k,l}}. \quad (2.20)$$

If $w_{i,j}$ is equal to 1 at every control point, or even equal to the same scalar, Equation 2.20 reduces to Equation 5.28. Thus, B-splines are just a special case of NURBS. With varying weights, geometric primitives like spheres and cylinders can be formed. In addition, varying weights can also give certain control points more or less influence over the surface.

Equation 2.20 is the industry standard NURBS. Figure 2.30 demonstrates a cylindrical NURBS surface, where the control points clearly outline the structure of the surface.

- NU: (non-uniform), the knot span is capable of being non-uniformly spaced.
- R: (rational), the surface weights can vary with the control points, constructing a rational formulation.
- BS: (B-spline), uses spline basis functions.

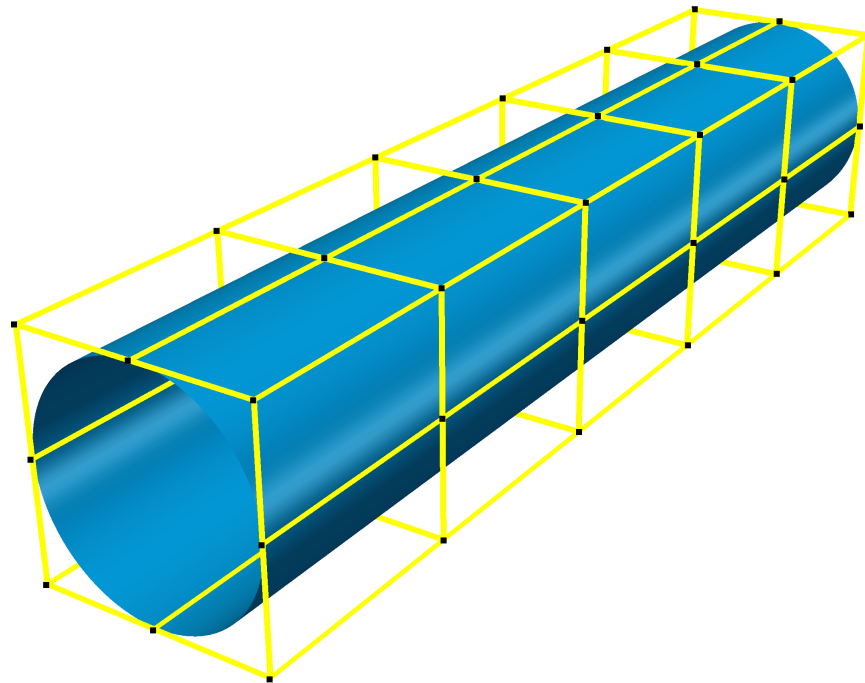


Figure 2.30: A cylinder modeled with NURBS surrounded by the control grid.

NURBS Volumes NURBS can vary in dimension by increasing or decreasing the dimension of the control points; however, it is difficult to represent, for example, a volume with two parametric directions. Therefore, a third parametric dimension is introduced to be able to better define a volume with parametric B-splines. A B-spline volume takes the form:

$$\mathbf{V}(u, v, w) = \sum_{i=0}^n \sum_{j=0}^m \sum_{k=0}^l \mathbf{P}_{i,j,k} N_{i,p}(u) N_{j,q}(v) N_{k,r}(w), \quad (2.21)$$

where $l + 1$ is the number of control points in the parametric direction w , and r is the degree of the volume in w . A NURBS volume could be constructed similar to Equation 2.20, but in three parametric directions.

For clarification, spline surfaces are often called bivariate splines. In addition, spline volumes are called spline solids and trivariate splines.

2.4.2 Lofting 2D Segmentations

The steps of creating paths and 2D segmentations along the vessel of interest are the exact same as that described in Section 2.3.1. The NURBS surface is obtained from the series of oriented 2D segmentations. It is easiest to fit to a NURBS surface with uniform weighting, thus the NURBS is reduced to a B-spline surface (Eq. 5.28). The input to a fitting problem consists of a set of ordered points which lends perfectly to using the series of ordered segmentations. The ordered and aligned segmentation points are input into the surface fitting algorithm as physical points that the surface will pass through, $\mathbf{Q}_{k,l}(u, v)$, where k is the number of segmentations and l are the number of points per segmentation. The data fitting problem is discussed more in detail in Section 2.4.2.1. This method of global interpolation using B-splines provides satisfying qualitative results for vascular modeling. Results can be improved even more by providing derivative information at the end of the vessels. With a general global surface fitting, there are no constraints in the direction of the vessel at the final segmentations. However, the linear system can be expanded and derivative information can be provided to constrain the direction of the vessel near the ends. Useful vector directions can either be the position between the final two segmentation points or the normal at the end of the vessel.

2.4.2.1 Data Fitting

There are both approximate and interpolation methods of data fitting. Interpolation involves ensuring the resultant curve or surface passes directly through the given data points; whereas, approximation involves finding the best fitting curve or surface of the data and it does not need to pass directly through the given data points. For vascular modeling with the lofted 2D segmentation approach, an interpolation approach is used to guarantee that the surface passes through the given segmentations. In the most recent versions of SimVascular, algorithms to update and display a new surface with an added segmentation make it possible

to achieve a surface that matches the image data very well. Using an approximation method doesn't guarantee that the surface lies on the 2D segmentations. Parametric surface fitting is just an extension of parametric curve fitting, so that will be described first. A NURBS curve is defined by Equation 2.18. The input is a set of data points, \mathbf{Q}_k , where $k = 0, \dots, n$, and n is the number of data points to fit. The curve that interpolates in between the input data points is found by satisfying the NURBS curve equation, where the control points, \mathbf{P}_i are $n + 1$ unknowns,

$$\mathbf{Q}_k(u) = \mathbf{C}(\bar{u}_k) = \sum_{i=0}^n N_{i,p}(\bar{u}_k) \mathbf{P}_i. \quad (2.22)$$

To satisfy the remainder of the equation, both parametric values, u_k and a knot span, $U = \{u_1, \dots, u_m\}$, are required. Choosing how to form both u_k and U is important and can drastically change the resultant fit curve. Though they can be formed through equal spacing, a better choice is to base the parametric values off the input data. One method is through chord length,

$$d = \sum_{k=1}^n |\mathbf{Q}_k - \mathbf{Q}_{k-1}|, \quad (2.23)$$

where $\bar{u}_0 = 0$ and $\bar{u}_n = 1$. The parameter values are then created using chord length,

$$\bar{u}_k = \bar{u}_{k-1} + \frac{|\mathbf{Q}_k - \mathbf{Q}_{k-1}|}{d}, k = 1, \dots, n - 1. \quad (2.24)$$

Though this is the most commonly used and usually provides a good approximation of the data, another method is the centripetal method:

$$d = \sum_{k=1}^n \sqrt{|\mathbf{Q}_k - \mathbf{Q}_{k-1}|}, \quad (2.25)$$

where again $\bar{u}_0 = 0$ and $\bar{u}_n = 1$. The parameter values are given using this modified chord length,

$$\bar{u}_k = \bar{u}_{k-1} + \frac{\sqrt{|\mathbf{Q}_k - \mathbf{Q}_{k-1}|}}{d}, k = 1, \dots, n - 1. \quad (2.26)$$

This tends to provide better results for data with sharper curves. The knot spans also need to be chosen. Again, equally spaced knot spans can be used, but tend to provide a worse result. Averaging of the knot span is what is most commonly used and gives quality results,

$$u_{j+p} = \frac{1}{p} \sum_{i=j}^{j+p+1} \bar{u}_i, \quad (2.27)$$

where the first and last $p + 1$ points are repeated to create a clamped knot span. In this way, the knot span similarly represents the parametric values and the given data. With the parameter values and the knot span, the basis function can be evaluated to create a banded

$(n + 1) \times (n + 1)$ system. When using chord length parameter values and an average knot span, the system is positive and has a semibandwidth that is less than the degree of the curve. The same methods are carried out for a surface, but in multiple parameter directions. As mentioned previously, it can be necessary to constrain the derivatives at the end of the curve or surface. This is done by expanding the linear system by one or two equations and modifying the parameter values and the knot span. Applying end derivatives to a curve will increase the input data by two points,

$$\mathbf{C}(u) = \sum_{i=0}^{n+2} N_{i,p}(u) \mathbf{P}_i. \quad (2.28)$$

The parameter values are calculated using Equation 2.24 and the knot span using Equation 2.27. The two additional equations to add to the linear system involve derivatives at the ends, \mathbf{D}_0 and \mathbf{D}_n ,

$$-\mathbf{P}_0 + \mathbf{P}_1 = \frac{u_{p+1}}{p} \mathbf{D}_0, \quad (2.29)$$

$$-\mathbf{P}_{n+1} + \mathbf{P}_{n+2} = \frac{1 - u_{m-p-1}}{p} \mathbf{D}_n. \quad (2.30)$$

This increases the system to $(n + 3) \times (n + 3)$ for a curve which is still a banded linear system.

2.4.3 Boolean

Again, for a vessel network, individual vessels must be united together with a union operation. The Boolean operation for NURBS is not described in detail here, and the reader is referred to [119]. However, in a similar manner to section 2.3.3, all vessels are included to create the entire vascular network.

2.4.4 Surface Manipulation Operations (NURBS)

A NURBS surface is modified very differently than a triangulated surface. It is possible to move control points and effectively move the surface in this manner, but this requires an interface that allows a user to interactively move control points. To smooth in between vessel junctions, a blend or fillet operation is typically used. The reader is referred to [84] for more information on fillets for NURBS surfaces. There are also a number of other operations that can be applied to a NURBS object at a more fundamental level. These operations are useful for manipulation of a parametric curve, surface, or volume, but are also necessary for improving the finite element space when used for IGA. For these different fundamental B-spline and NURBS operations, they are presented using curves of uniform weighting. A parametric direction is added for each increase in dimension; thus, for a volume, two parametric directions would be added. The following sub-sections describe various NURBS operations implemented in the code developed as part of this dissertation.

2.4.4.1 Change Degree

Increasing and decreasing the degree of a curve or surface can be useful for many applications. In order to combine adjacent curves or surfaces, it may be necessary to have them represented with the same degree. Specific to IGA, increasing the degree of a curve or surface can alter the convergence and accuracy of a solution and is similar, but not identical to, p -refinement in FEA. It is always possible to increase the degree of a curve or surface; however, it may not be possible to decrease the degree. In addition, there is error introduced in decreasing the degree of a parametric object, and this error can be calculated. The reader is referred to [84] for a more in depth discussion on degree elevation and reduction. Fig. 2.31 demonstrates degree elevation of a curve. Note that the curve stays identical geometrically and parametrically in this process.

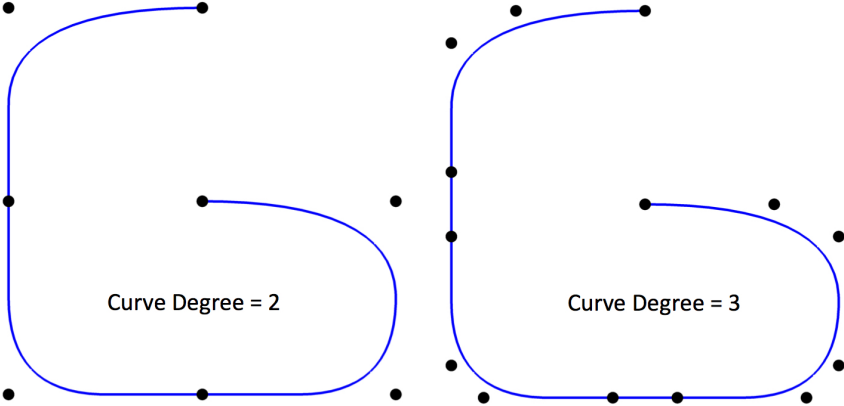


Figure 2.31: (Left) A B-spline curve of degree 2 with 8 control points defining it’s shape. (Right) the same B-spline curve is elevated to degree 3. Despite the increase in degree, the curves are geometrically and parametrically identical.

2.4.4.2 Knot Insertion/Removal

Knot insertion and removal involves increasing or decreasing the size of the knot span in order to change the vector space of the curve or surface. Like with degree elevation, knot insertion does not alter the curve or surface geometrically or parametrically. However, it may not always be possible to remove a knot and removing a knot can alter the parametric object geometrically with some known error. This is very similar, but not identical to, h -refinement in FEA. Knot insertion is useful for taking derivatives of curves or surfaces and is necessary for adding new control points. The reader is referred to [84] for a more in depth discussion on knot insertion. Fig. 2.32 displays the altered control mesh for a curve of increased knot span.

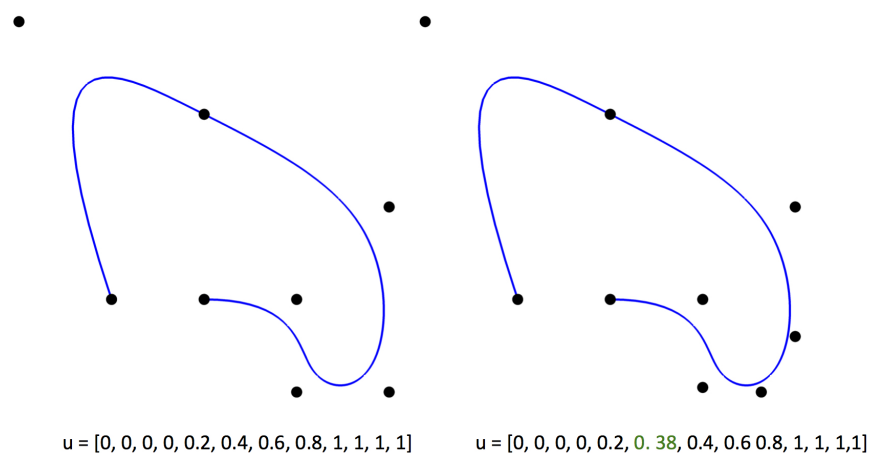


Figure 2.32: ((Left) A B-spline curve of degree 3 and with the knot span u is displayed. (Right) A knot with value 0.38 is inserted into the knot span. The B-spline stays geometrically identical to the curve before knot insertion.

2.4.4.3 Bezier Extraction

Another operation that is useful is Bezier extraction, and this is a step that is necessary for evaluation in IGA. In Bezier extraction, a parametric curve or surface is reduced to a complete combination of Bezier curves or surfaces. A Bezier curve is the special case of B-spline curve where $p = n - 1$. For example, if a B-spline curve has a degree of 3 and there are 8 points along the curve, the B-spline curve would be extracted into two separate Bezier curves of 4 points each. The reader is referred to [84] for a more in depth discussion on Bezier extraction. Fig. 2.33 shows the Bezier curves making up a larger B-spline curve.

2.5 Discussion

Modeling is typically the most time consuming step in performing a study involving a patient-specific blood flow simulation. Furthermore, the model can be one of the largest sources of error in simulation. Thus, it is important to have accurate and effective modeling tools in a framework that enables construction of a model from medical image data. The tools described here are a subset of the tools within SimVascular that have been improved upon or developed as part of this dissertation.

The algorithms in this chapter are implemented in varying capacities. Though some of the procedures are implemented and usable directly from the GUI of SimVascular, others are typically not used in the SimVascular pipeline and thus remain as implementations solely within vtkSV.

The remeshing algorithms are built directly into SimVascular using the libraries of VMTK and MMG. It is also important to note that there are a variety of other softwares that provide

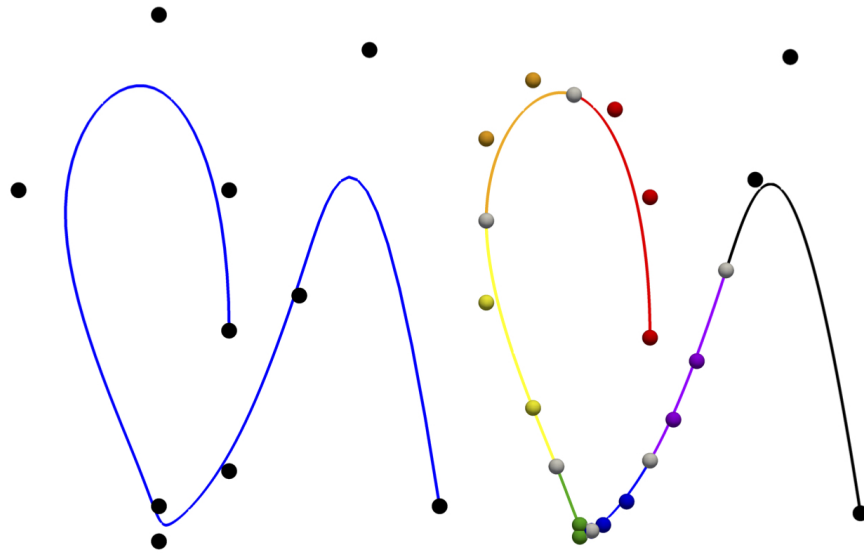


Figure 2.33: (Left) A B-spline curve of degree 3 is displayed. (Right) Each Bezier segment is extracted and displayed with a different color. The white points are shared control points between two adjacent Bezier curves.

surface manipulation algorithms for discrete surfaces. MeshLab (<http://www.meshlab.net>), OpenFlipper (<https://www.openflipper.org>), and MeshMixer (<http://www.meshmixer.com>) all provide binaries with a suite of tools for discrete surface manipulation. Alternatively, open-source tools for parametric modeling and modification are sparse. OpenCASCADE (<https://www.opencascade.com>) is the most advanced parametric solid modeling kernel, while FreeCAD (<https://www.freecadweb.org>) provides a user interface on top of OpenCASCADE complete with python scripting.

Chapter 3

Vascular Meshing for Finite Element Analysis

3.1 Introduction

In the early stages, meshing began with the formation of structured grids in 1 and 2 dimensional studies. As numerical methods expanded and extended beyond 2 dimensions, meshing procedures also expanded to provide discretized domains in 3D [120]. Likewise, to adapt to more complicated domains, meshing expanded from structured grids to fully unstructured grids. The progression of meshing included new and different techniques including advancing front methods and delaunay tetrahedralization methods [121][122]. Meshing also adapted to include a multitude of options to provide a better discretization for numerical solutions. For example, mesh refinement and coarsening allows for the possibility to add more or take out points in a specific region where a more or less refined solution is desired [123]. In addition, custom meshing techniques developed around specific applications. For example, boundary layer meshing is used heavily in CFD because it provides smaller, thinner elements on the boundary where the gradient of the velocity normal to the surface is changing most drastically [124]. In addition, mesh adaption based on *a-posteriori* estimates is common and can provide a mesh that is ideal in size for the simulation [36].

The ideal meshing procedure is robust, computationally efficient, and provides a desired element quality/shape. Most meshers that meet these requirements are commercial and expensive. In the past decade, a variety of open-source tools have become available, and although all desirable meshing options are not available in one package, a combination of tools provides a strong and efficient finite element mesher. This chapter describes the variety of meshing options developed for vascular models using a combination of open-source tools and custom code. The open-source tools include TetGen (<http://www.tetgen.org>), the Vascular Modeling Tool Kit (VMTK; <http://www.vmtk.org>), and Mesh Modification and Generation (MMG; <https://www.mmgtools.org>).

3.2 Uniform Isotropic Meshing

Uniform meshing prescribes the same mesh size everywhere within a domain. A quality mesher attempts to achieve the same specified size for all elements of the mesh. Typically, the size specified is either given in terms of desired element volume or element edge size. As it is almost always impossible to obtain this exact size measure for all elements in irregular domains, this is often taken to be the maximum size allowed. Thus, no element within the mesh will have a volume or edge size larger than the one specified. In more rigorous meshing applications, a minimum and maximum area constraint can bound the element size. This measure drives the number of elements and nodes in a given mesh. **Isotropic** refers to the way in which this size measure is specified. Isotropic means the same mesh size will be applied the same in all directions. With certain meshers, anisotropic meshing is possible where a different mesh size is specified for each coordinate direction. This gives a mesh in which elements have a specified direction and is often used in CFD to align elements with the flow. Figure 3.1 demonstrates a uniform isotropic mesh from the open-source meshing module in SimVascular.

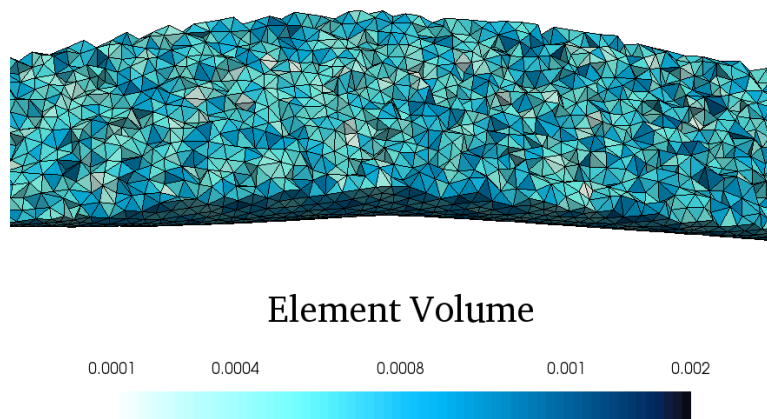


Figure 3.1: Uniform Isotropic Mesh generated from SimVascular 2.0: All elements have nearly the same volume. An element edge size of 0.2 cm was prescribed giving element volumes ranging from 0.0001 cm^3 to 0.001 cm^3

3.3 Boundary Layer Meshing

Boundary layer meshing applies a special layer of elements on the boundary of the mesh. The elements on the boundary are thin and aligned along the boundary direction. This provides a more refined and ideal mesh size on the wall where, in fluid simulations, the velocity at the wall is changing drastically. An example of a boundary layer mesh is displayed in Figure 3.2.

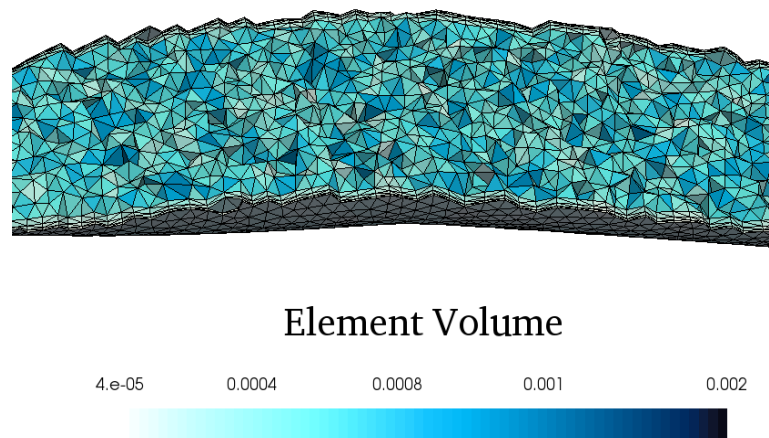


Figure 3.2: Boundary Layer Mesh generated from SimVascular 2.0: Elements on the exterior have a much smaller volume than the interior elements. An element edge size of 0.2 cm with three boundary layers was prescribed. The radial edge size of the initial boundary layer is 0.5 cm , and the continuing layers decrease at 60% of the previous layer.

3.4 Spherical Mesh Refinement

Spherical Mesh Refinement applies a different mesh size in a specific region. This region can be specified in a variety of ways, but the one implemented applies either a smaller or larger mesh edge size in a specified spherical region. Fig. 3.3 demonstrates a smaller mesh edge size within a spherical region at a vessel bifurcation.

3.5 Local Mesh Size Application

Local Mesh Size Application is useful when vessel geometries have a wide range of length scales. For example, anatomic models for simulating coronary blood flow can include the aortic arch, which can be an order of magnitude larger in diameter than the modeled coronary arteries. This allows for the entire domain to still be meshed in a reasonable computational time. This also allows the capability to specify a mesh size function at all nodes within the mesh. VMTK has developed a robust algorithm for finding centerline paths of a surface based on seed and target points [82]. Using these centerlines, it is possible to find the distance from the centerline path. In order to mesh a vessel of smaller radius, it will obviously need a smaller mesh size. Therefore, a mesh sizing function is set based on this radius size and the mesh edge size specified by the user. The radius values on each point of the surface are normalized by the minimum vessel radius size and then multiplied by the mesh edge size given. An aorta with renal branches is shown with radius-based meshing in Figure 3.4.

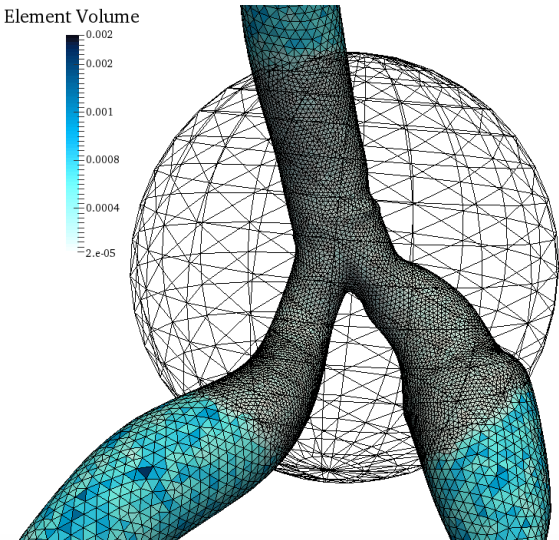


Figure 3.3: Mesh with spherical refinement produced by SimVascular 2.0. A uniform mesh edge size of 0.2 cm was prescribed, and a mesh edge size of 0.1 cm was prescribed within the spherical region.

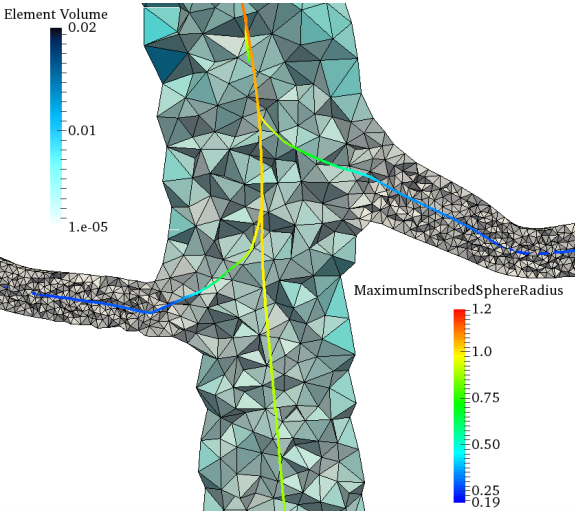


Figure 3.4: Radius based mesh produced by SimVascular 2.0. Centerline paths of the vessels are computed and then distances on the mesh from the centerline are computed to distribute the mesh size based on the radius at each particular point.

3.6 Multi-Domain Meshing

Multi-domain meshing is useful in a variety of applications. Specific to blood flow simulations, it can be desirable to capture the mechanics within the vessel wall and couple this to the blood flow mechanics through a fluid structure interaction simulation. In this case, a separate mesh is required for the vessel wall. However, it is often necessary that the interface between the domains share mesh nodes to aid in solution transfer. As is seen in Fig. 3.5, the multi-domain meshing implemented creates a mesh in which the interface between the two domains is consistent. It is also possible to prescribe different mesh sizes for the different regions of the multi-domain mesh.

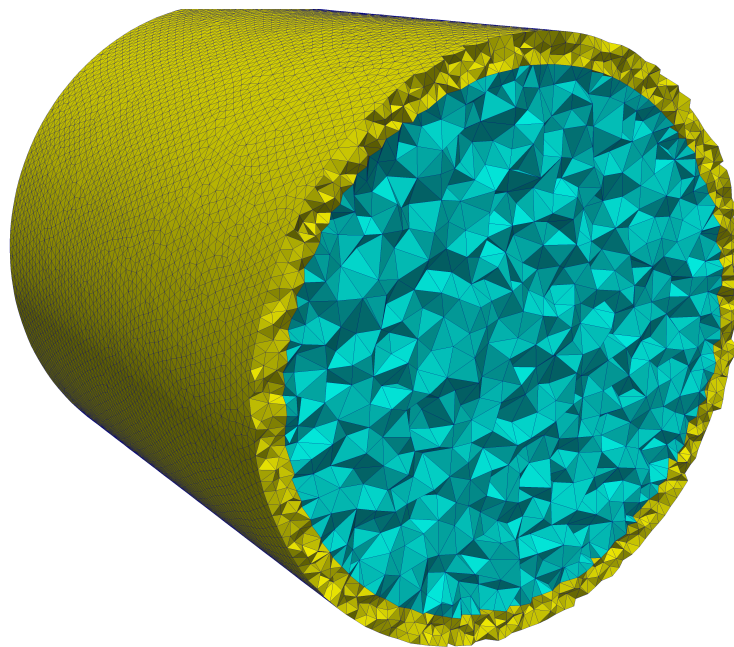


Figure 3.5: Multi-Domain Mesh generated from SimVascular 2.0: A multi-domain mesh is created by meshing an internal cylinder to a mesh size of 0.1 cm , while an outer wall region is meshed to 0.05 cm . Nodes are consistent at the interface between the two domains.

3.7 Meshing with Holes

Meshing with holes or voids in the mesh is similar to multi-domain meshing in some cases; however, one or more of the domains are gaps or voids in the mesh rather than another mesh region. This can be useful in blood flow, for example, to simulate the effect of some obstruction in the blood flow, say for instance an intra-arterial catheter or some other medical device. In Fig. 3.6, a cylinder is meshed with a spherical void on the interior.

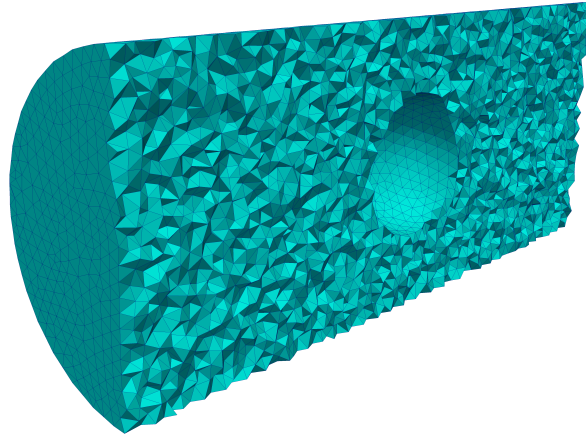


Figure 3.6: A mesh with a hole generated in SimVascular 2.0: A cylindrical mesh with a void demonstrates the capability of generating a volumetric mesh with one or multiple void regions.

3.8 Isotropic Mesh Adaption

Mesh adaption uses *a posteriori* estimates to redefine a mesh that is more appropriate for the solution. According to Sahni et. al [36], mesh adaption is a feedback process with three key steps. With the addition of a fourth alternate step, the following describes the process of mesh adaption.

1. Estimating error of the solution. [125].
2. Transforming the error information into a mesh applicable factor for all mesh nodes.
3. Recreating a mesh based on the local mesh size factors [126].
4. Interpolating the solution back onto the newly created mesh.

In the mesh adaption implemented (Figs. 3.7 and 3.8), the speed of the solution was used to compute an error estimate. From this field metric, the Hessian, the second directional derivative, is computed [127]. For a scalar value, such as velocity magnitude s , the Hessian, H , is a symmetric 9×9 Tensor. H can be decomposed as in Equation (3.1) to give the Eigen Vector Matrix, R , and the diagonal eigenvalue matrix, Λ . The strategy then uses this matrix, Λ , which gives three independent eigenvalues, λ_k due to the symmetric nature of the Hessian, where the eigenvalues provide information about the error in the flow field. The higher magnitude eigenvalues indicate a larger error in the solution, and the mesh size in this direction should be refined. Likewise, the lower magnitude eigenvalues indicate that the mesh in that direction can actually be coarsened. In this way, the Hessian is computed for all the nodes of the mesh. The only caveat comes when computing the Hessian on the boundary

nodes where the solution is zero due to the no-slip boundary condition. This means in terms of error estimation that the flow is changing infinitely and would suggest that the mesh be refined infinitely at the wall. Therefore, the boundary nodes are treated specially and the information from a node just interior to the boundary is interpolated to each boundary node.

$$H = R\Lambda R^T \quad (3.1)$$

The gradient of speed is given by the partial derivative in each coordinate direction (3.2), and the Hessian is given by taking the gradient again (3.3).

$$\nabla s = \frac{\partial s}{\partial x} \mathbf{i} + \frac{\partial s}{\partial y} \mathbf{j} + \frac{\partial s}{\partial z} \mathbf{k} \quad (3.2)$$

$$H(s)_{ij}(x) = \frac{\partial^2 s}{\partial x_i \partial x_j} \quad (3.3)$$

After the eigenvalues for the Hessian are obtained, the transformation of this information into a mesh size metric is necessary. The applied size metric is a modification of the eigenvalues. First, bounds to the mesh size are supplied because it is possible to have points where the error is nearly zero and the suggestion to create an unnecessarily large mesh size is unwanted. These are defined as h_{min} and h_{max} in Equation (3.4). Then, the mesh size metric, $\tilde{\lambda}_k$, is a function of the eigenvalues, λ_k , and these limits. Epsilon, ϵ , is a user defined tolerance on the error.

$$\tilde{\lambda}_k = \min \left(\max \left(\epsilon^{-1} |\lambda_k|, \frac{1}{h_{max}^2} \right), \frac{1}{h_{min}^2} \right) \quad (k = 1, 2, 3) \quad (3.4)$$

$$\tilde{\Lambda} = \text{diag}(\tilde{\lambda}_k) \quad (3.5)$$

$$M = R\tilde{\Lambda}R^T \quad (3.6)$$

This gives a mesh metric, M , in the principal directions corresponding to the eigenvalues. In certain mesh cases, an anisotropic mesh size can be applied to the mesh to give this mesh size alteration in each direction. In the implemented case, the meshing module used does not provide anisotropic mesh adaption options, so the error estimate is done on a nodal basis and prescribed isotropically. Based on these new mesh metrics, a re-meshed domain is computed and the solution is interpolated onto the new mesh by just taking the solution from the nearest node.

3.9 Comparison

To demonstrate the difference in meshing algorithms and softwares, a brief comparison between MeshSim and TetGen is displayed. MeshSim is the robust commercial mesher inside

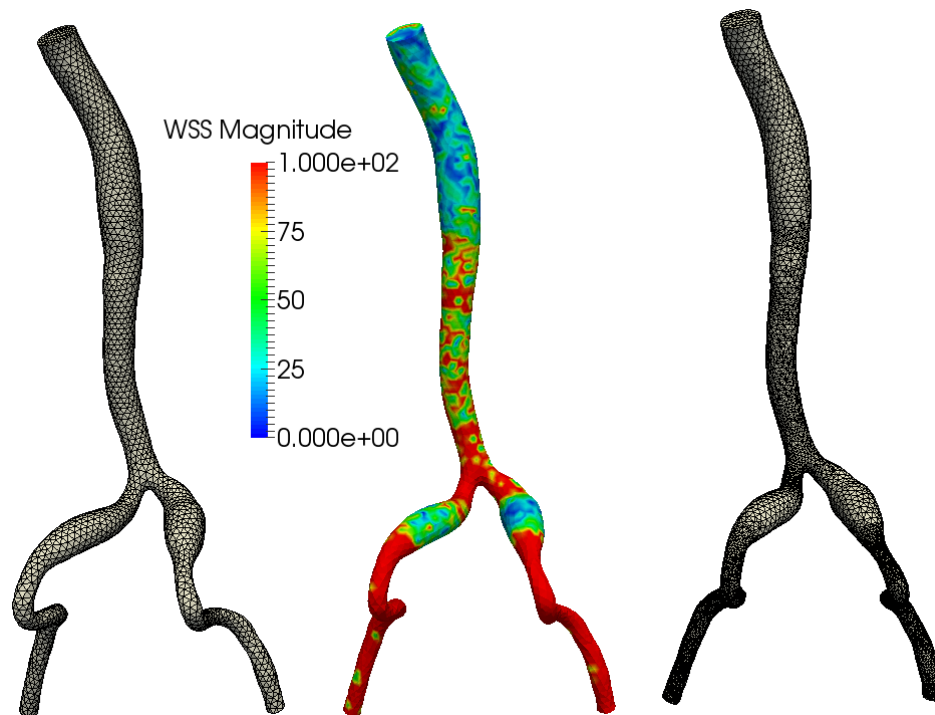


Figure 3.7: On the left, a very coarse initial mesh on which the incompressible Navier-Stokes equations are solved. In the middle, the WSS magnitude displayed giving some interpretation on what the solution is like within the domain. On the right, the adapted mesh based on *a posteriori* estimates.

of SimVascular. In Figure 3.9, a section of the mesh is extracted to show the difference of the two meshing techniques when similar mesh options and mesh sizes are applied. A boundary layer mesh was generated with three layers and a mesh edge size of 0.2 cm. Select meshing statistics are given in Table 3.1. As shown in the table, a similar number of elements with comparable mesh quality were generated using both meshing techniques.

Steady blood flow simulations were run using both meshes to compare the impact of the meshing techniques on simulation results of interest. A physiologically reasonable steady volumetric flow rate was applied to each mesh. Figure 3.10 displays the wall shear stress magnitude after reaching steady state. The instantaneous wall shear stress was visualized and compared favorably as shown. After running the steady simulation, each mesh was adapted to the solution using techniques described previously; this information is displayed in Table 3.1. Differences in mesh elements and nodes are likely due to three factors:

1. MeshSim does not adapt within the boundary layer.
2. TetGen only refines and does not coarsen elements.

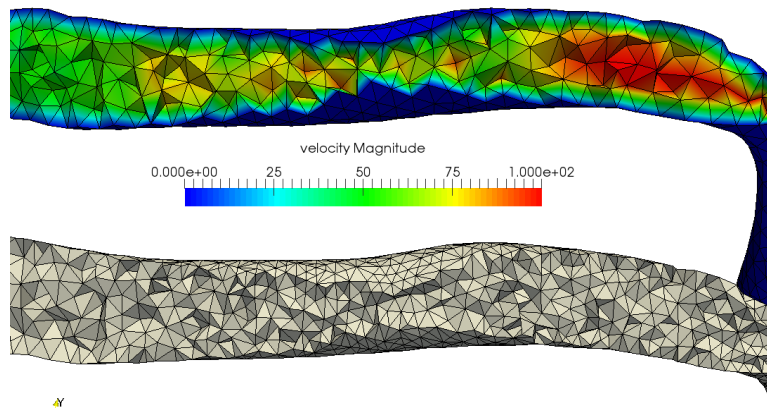


Figure 3.8: On top, the original mesh prior to adaption with the velocity field visible. On bottom, the adapted mesh is displayed. Where the solution is changing rapidly, it is easy to see that the adaptor has refined in this region. An initial mesh size of 0.45 cm was applied. The h_{max} and h_{min} bounds for adaption were then prescribed to be 0.45 cm and 0.05 cm correspondingly.

Table 3.1: Comparison of MeshSim (A.) and TetGen (B.)

Mesh Metric	MeshSim	TetGen
Number of Elements	436,048	456,309
Number of Nodes	82,235	81,887
Min/Max Volume (cm^3)	$6.84e^{-6} / 2.57e^{-3}$	$2.26e^{-5} / 1.59e^{-3}$
Min/Max Aspect Ratio	1.028 / 12.74	1.00 / 7.45
Min/Max Radius Ratio	1.00 / 21.29	1.00 / 8.37
Number of Elements after Adaption	453,142	481,472
Number of Nodes after Adaption	83,220	85,880
Min/Max Volume after Adaption (cm^3)	$3.21e^{-6} / 1.74e^{-2}$	$6.09e^{-6} / 1.59e^{-3}$

3. TetGen and MeshSim differ in adaption technique. MeshSim applies an anisotropic mesh function (different based on direction); whereas, TetGen applies an isotropic mesh function (same parameter for each direction).

3.10 Discussion

Meshing is the critical link between a computer model and performing numerical analysis. Despite the obvious need for robust tools, many of the software meshing options are com-

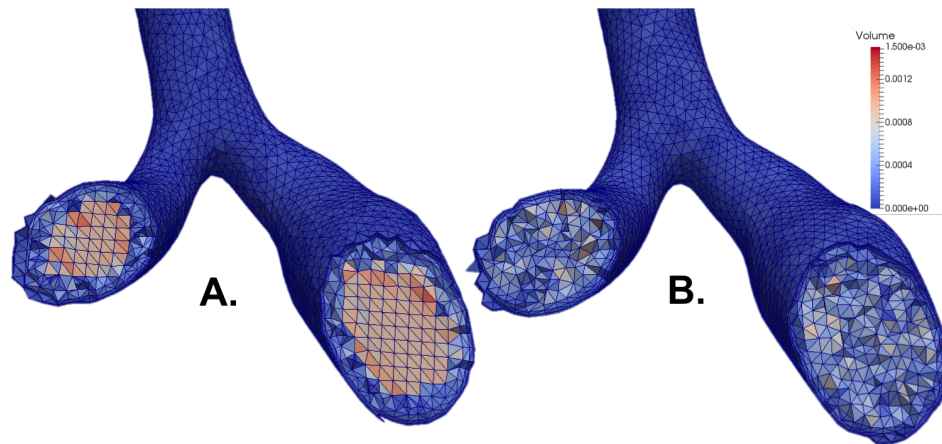


Figure 3.9: Comparison of a MeshSim mesh (A.) and a TetGen mesh (B.) on the same geometry. A mesh edge size of 0.2 cm was applied with three boundary layers. The initial boundary layer size in the radial direction was 0.1 cm with a decreasing ratio of 0.6 for each continuing layer.

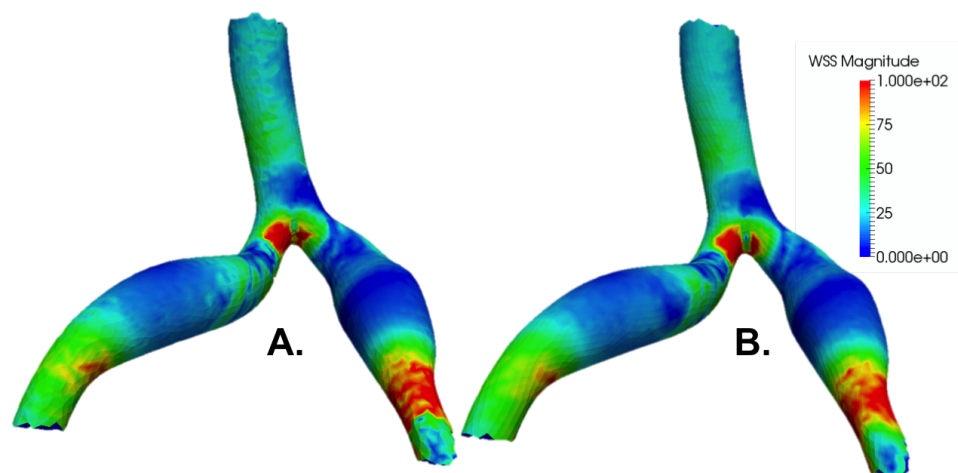


Figure 3.10: Comparison of WSS values from a steady flow simulation on the MeshSim mesh (A.) and TetGen mesh (B.).

mercial. Thus, the tools here were developed to provide a viable open-source alternative for meshing of vascular anatomies. Although the primary application of these tools is for patient-specific vascular modeling, the tools can be used and expanded to other applications.

The tools are implemented with VMTK, MMG, and custom techniques for surface remeshing and TetGen for volumetric meshing. An alternate open-source meshing software that is commonly used but was not included in this work is gmsh (<http://gmsh.info>).

Chapter 4

Vascular Modeling for Isogeometric Analysis

4.1 Introduction

Finite element analysis (FEA) is a computational method used to solve a differential equation, where that differential equation typically represents a physical problem such as solid deformation or fluid flow. FEA began as a technique to solve simple structural problems, but it is now being used to analyze problems of increasing complexity. The increasing complexity in FEA problems has led to much more time being spent in designing the geometry of interest and creating the finite element mesh. In addition, it is common to iterate between a computational model and the corresponding finite element mesh to achieve desirable numerical results, further amplifying the time spent in modeling and meshing. However, with the relatively new framework of isogeometric analysis (IGA) [128], it is possible to iterate quicker between a geometric model and numerical analysis. This is made possible by using the basis functions already used in a computer aided design (CAD) model definition. A new mesh does not need to be created for small alterations to the geometry or when higher refinement is desired. Despite the obvious attractiveness of IGA, it is still a major challenge to obtain a model definition that is said to be *analysis suitable* for IGA. This is due to the stiff restrictions required of a finite element space, and although many CAD models are defined using basis functions that are valid for numerical analysis, the entire geometric model definition does meet the requirements of a finite element space.

In *design*, the industry standard geometric representation is the non-uniform rational B-spline (NURBS) surface. Thus, NURBS surfaces were the geometric representation initially used and validated for IGA. NURBS, however, have inherent issues for use in *analysis*. Namely, it is impossible to represent complex geometries with a single NURBS patch, and NURBS do not support local refinement. Thus, almost every complex parametric model designed in a CAD framework contains multiple NURBS patches, and often times those patches are actually *trimmed NURBS*. For example, when two CAD objects are united

through a Boolean operation, a trimmed NURBS is created to allow the objects to meet at the object boundaries. Trimmed NURBS contain parametric curves that cut through the parameterization, destroy the finite element space, and actually produce small gaps in between NURBS patches. In addition, it is very rare for parameterizations of NURBS patches to match at patch boundaries. Depending on the implementation of IGA, this can be a major issue. Recent studies have developed techniques to utilize trimmed NURBS and NURBS with mismatching parameterizations at patch boundaries in IGA [84]. These studies still require quite special cases and the algorithms still need further development to be a viable option. Alternatively, many studies have begun to investigate other geometric representations than NURBS for IGA.

This chapter will discuss some of the newer geometric representations and assess their viability in an IGA framework. First, a brief background of FEA is given with special emphasis on how IGA is incorporated in an analysis framework in section 4.2. Then, some of the more recently proposed geometric representations for both surfaces and volumes are discussed in Section 4.3. Finally, conclusions about some of the geometric representations and the future of IGA is discussed in Section 4.4.

4.2 Isogeometric Analysis

FEA began as a method to solve complex elasticity and structural problems in civil and aeronautical engineering, and its origins can be attributed to the work of Hrennikoff and Courant [129, 130]. They proposed a method to approximate a solution to a differential equation without analytically solving the equation. Clough then coined the term finite elements in 1960 [131].

In FEA, there are two main steps: (1) developing a variational form of the differential equation, and then (2) discretizing or breaking up the equation and domain into smaller spatial dimensions.

There are several classes of variational forms. These include collocation, least squares, meshless, and Galerkin type methods. Galerkin type methods are both the most popular variational methods and what was used originally for IGA; thus, a Galerkin formulation will be presented. In the Galerkin method, the weighted average of a differential equation is constructed, where the weighting functions to approximate the solution locally are intelligently chosen. These weighting functions are typically polynomials and are where NURBS fit into the framework. For a more detailed description of finite element methods, variational forms, and discretization methods, see [132].

Discretizing the domain refers to breaking up the geometry into smaller elements onto which the variational form can be prescribed (Chapter 3). As discussed previously, with FEA, a finite element mesh is created from the domain. This finite element mesh is a discretization choice. In IGA, there is a smarter way to discretize the domain based on how the domain is already defined. We demonstrate this below. For a more complete understanding of the integration of CAD geometries in IGA, see [133].

To help demonstrate FEA, a simple example is presented. This demonstration is not intended to be a rigorous explanation of FEA, but rather to give a simple and understandable background to show how NURBS fit in. The example is a simple second order ordinary differential equation, where a solution for u is desired:

$$-u''(x) = f(x) \text{ in } \Omega, \quad (4.1)$$

$$u(0) = 0 \text{ on } \Gamma_D, \quad (4.2)$$

$$u'(1) = g \text{ on } \Gamma_N, \quad (4.3)$$

where $\Omega \in \mathbb{R}$ is the domain of the problem, and in this case, $\Omega = x \in (0, 1)$. Γ is the boundary of the domain. The total boundary can be broken up into two parts ($\partial\Omega = \Gamma = \Gamma_D \cup \Gamma_N$) for boundary condition application, and the boundary parts do not overlap ($\Gamma_D \cap \Gamma_N = \emptyset$). On Γ_D , a Dirichlet, or exact boundary condition, $u(0) = 0$, is applied. On Γ_N , a Neumann, or flux type, boundary condition, $u'(1) = g$, is applied, where $g \in \mathbb{R}$.

To start the Galerkin formulation, the weighted average is formed:

$$\int_0^1 -\hat{u}''(x)w(x)dx = \int_0^1 f(x)w(x)dx, \quad (4.4)$$

where \hat{u} is the approximate solution, and $w(x)$ are the weighting functions. The variational form is created through integration by parts. In this process, a derivative of the solution is transferred to the weighting function, and a boundary term is formed. Integration by parts results in the variational form on which it is easy to apply Neumann type boundary conditions:

$$\int_0^1 \hat{u}'(x)w'(x)dx - [\hat{u}'(x)w(x)]_0^1 = \int_0^1 f(x)w(x)dx, \quad (4.5)$$

$$\int_0^1 \hat{u}'(x)w'(x)dx - \hat{u}'(1)w(1) + \hat{u}'(0)w(0) = \int_0^1 f(x)w(x)dx. \quad (4.6)$$

Part of the restrictions placed on the weighting functions make $w(0) = 0$ (Equation 4.9). The boundary condition, $u'(1) = g$, can also be applied. Equation 4.7 is the full variational or weak formulation for the example problem:

$$\int_0^1 \hat{u}'(x)w'(x)dx = \int_0^1 f(x)w(x)dx + g \cdot w(1). \quad (4.7)$$

This formulation currently exists on the domain from 0 to 1. The domain can be split up into smaller pieces of non-overlapping elements such that:

$$\mathbf{E}_h = \{K_1, K_2, \dots\}, \quad (4.8)$$

where $\cup_{K_i \in \mathbf{E}_h} = \Omega$. In addition, the function space of the weighting functions on the elements should also be restricted:

$$W_h = \{w \in C^0([0, 1]) : w|_K \in \mathbb{P}_p(K) \forall K \in \mathbf{E}_h, w(0) = w(1) = 0\}. \quad (4.9)$$

$\mathbb{P}_p(K)$ is the space of polynomials on K of at most degree p . Equation 4.9 says that the weighting functions must be at least C_0 continuous across elements on the entire domain and the weighting functions are of degree p . The weighting functions can now be specified by element-wise basis functions:

$$\phi_i = \phi_i(x_j) \in W_h, \text{ for } i, j = 1, \dots, n, \quad (4.10)$$

where n is the number of elements. The approximate solution can also be discretized with the basis functions:

$$\hat{u}(x) = u_h(x) = \sum_{i=1}^n u_i \phi_i(x), \quad (4.11)$$

and $u_h(x_j) = u_j$ for $j = 1, \dots, n$. Equation 4.7 can now be written as:

$$\int_0^1 u_h'(x) w'(x) dx = \int_0^1 f(x) w(x) dx + g \cdot w(1), \quad \forall w \in W_h. \quad (4.12)$$

Equation 4.12 holds for $w = \phi_i, i = 1, \dots, n$. Substituting in the basis functions for u_h and w gives:

$$\int_0^1 \left(\sum_{j=1}^n u_j \phi_j'(x) \right) \phi_i'(x) dx = \int_0^1 f_i(x) \phi_i(x) dx + g \cdot \phi_i(1), \text{ for } i = 1, \dots, n. \quad (4.13)$$

Equation 4.13 is the Galerkin formulation discretized to an element space where the solution and the weighting functions are approximated with the element-wise, polynomial basis functions $\phi(x)$.

The polynomial basis functions are how NURBS fit in. The basis functions that were talked about with respect to B-spline and NURBS curves and surfaces in Section 2.4.3 are used as the polynomial basis functions. So, $\phi_i(x) = N_{i,p}(u)$ where u is a function of x for the B-spline basis functions. In addition, the basis functions given in Equation 2.16 and defined along the knot span in Equation 2.17 contain key features that make them viable for IGA:

1. Partition of unity: $\sum_{i=0}^n N_{i,p}(u) = 1, \forall u$.
2. Nonnegative pointwise over the entire domain: $N_{i,p}(u) \geq 0, \forall u$.
3. Linear independence: $\sum_{i=0}^n \alpha_i N_{i,p}(u) = 0$ where $\alpha_k = 0, k = 1, 2, \dots, n$.
4. Local or compact support: $\{u \mid N_{i,p} \geq 0\} \subset [u_i, u_{i+p+2}]$.

5. Controllable continuity: C_{p-m} continuity between across a knot value. Where m is the multiplicity of the knot, or the number of times the knot value is repeated ($u_i = u_{i+1} = \dots = u_{i+m-1}$).

Numbers 1-4 guarantee that the finite element matrix is well-conditioned and sparse, and number 5 provides controllable continuity. A higher degree of continuity will provide a smoother basis which can provide better accuracy in finite element simulations.

4.3 Analysis Suitable Representations

NURBS are the industry standard for modeling, and they are easily extended for use in IGA. However, NURBS have the following shortcomings in IGA:

1. Complex geometries cannot be represented as a single NURBS surface. Therefore, complex geometries must be constructed with multiple connected NURBS surface patches, which makes the definition much more complicated. In addition, material properties, boundary conditions, and other attributes must be applied for each new surface patch.
2. It is very difficult to avoid gaps and overlaps at intersections of surfaces. When this difficulty is combined with the fact that most complex geometries cannot be constructed with one NURBS surface, many problems arise. Gaps and intersections destroy the continuity between surface patches, which leads to errors in analysis.
3. It is not possible to locally refine the surface. In a large class of realistic problems, the geometry is deforming rapidly and needs to be adapted locally. Without local refinement, these adaption problems are essentially impossible.

Despite these shortcomings in analysis, NURBS are not going anywhere. The CAD industry has poured millions of dollars into NURBS, and they work very well for design. Therefore, it is important that if a new geometric representation is to be established, it should be in some form compatible with NURBS. Recent studies have attempted to come up with new geometric representations that overcome the drawbacks of NURBS for IGA, but still are representable as NURBS. First, various recent *surface* geometric representations are discussed. Next, the more complicated *volume* geometric representations are discussed. In many cases, there is an overlap in surface and volume geometric representations and many studies evaluate both, but we will do our best to separate the two subjects, often times referring to the same study in both cases. These are active areas of research, and particularly with volumes, a satisfiable geometric representation for design and analysis is yet to be found.

Surface Representations A geometric representation gaining a lot of attention in IGA is T-splines. The T-spline was proposed by Tom Sedberg in 2003, and it is both a generalization and improvement of NURBS [134]. Where NURBS require a grid like structure of

control points, T-splines allow a less grid-like structure with T-junctions, hence the name. These T-junctions not only allow T-splines to capture complex geometries in a single patch, but they also allow local refinement. Bazilevs et al. al [135] implemented T-splines in IGA to study simple structural mechanics. In this study, a few drawbacks to T-splines arise. Namely, T-splines do not necessarily satisfy the partition of unity, they lack efficient refinement algorithms, and they have poor treatment of singular points.

Another geometric representation that is growing in popularity is the subdivision surface [136, 137]. A subdivision surface is a triangle or quadrilateral surface mesh with a rule on how the surface mesh cells are subdivided iteratively to reach what is called the limit surface. The most common rule is by Catmull-Clark subdivision proposed by Catmull in 1978 [138]. Like T-splines, subdivision surfaces are ideal for gap-free or water-tight surfaces. In addition, there is no restriction on the topology of the original or control surface mesh. These surfaces are very popular in visualization and are used in all Pixar animations because of the minuscule amount of data needed to store a surface. Subdivision surfaces have previously been used to study the structure mechanics of thin shells [139]. Despite being so popular in the visualization community, subdivision surfaces have failed to take hold in the CAD community due to the lack of inherent compatibility with NURBS. However, subdivision surfaces are useful for representing free form and naturally-shaped objects, and thus warrant future investigations.

In another representation, Schillinger et al. recently extended the work of Forsey and Bartels [140] on hierarchal B-splines to NURBS and T-splines [141]. Hierarchical B-splines allow local refinement by computing what is called an overlay of the surface. The overlay breaks down the basis functions in one knot span into more basis functions and control points. This overlay is then linearly combined with the original B-spline surface definition. Careful analysis in the way the overlay is constructed helps the hierarchical B-spline definition maintain important properties such as the partition of unity. Schillinger et al. used hierarchical B-splines with immersed boundary and finite cell methods to solve structural mechanics equations. They praise the reduction in complexity due to T-splines; however, they mention that there will most likely be situations in which a traditional finite element mesh is preferable. Drawbacks to these representations include the fact that they are not outright representable as a NURBS surface. Conversion to the standard CAD representation causes a loss of the locally refined portion.

Another spline representation was proposed by Deng et al. in polynomial splines over hierarchal T-meshes or PHT splines [142]. PHT splines are essentially a generalization of T-splines. PHT splines differ from T-splines in that they contain piecewise basis functions of polynomials that extend over a mesh that allows T-junctions. PHT splines satisfy the partition of unity, and they are very easily refined locally. Wang et al. [143] recently presented a framework to solve a simple class of adaptive problems using the nice local refinement properties of PHT splines. These surfaces still require much investigatory work; for example, in the realms of efficient boundary condition application and error estimation.

An even newer representation that focusses on overcoming refinement issues is the LR (locally refined) B-spline [144]. LR B-splines are similar to T-splines; however, they differ in

refinement technique. Whereas T-splines use a control point based refinement technique, LR B-splines use a spline space refinement technique. Spline space refinement allows knot spans to be broken into more polynomial segments while introducing fewer new control points (refines while reducing added complexity). LR B-splines were extended for use in IGA by Johannesen et al. [145]. Better local refinement provides a better representation for the large class of realistic problems that require adaption.

Figure 4.3 displays the merging of design and analysis and the recent development of geometric representations and their use in IGA. Surfaces not on this timeline that could be investigated for use in IGA also include Gordon patches, Gregory patches, S-patches, A-patches, and Coon patches. Lastly, the most recent geometric representation is the U-spline. It is claimed that the U-spline solves the problem of local refinement and is usable in an unstructured manner; however, it is an extremely new technology and the details have not been published, so it is yet to be seen on whether the new geometric representation takes hold.

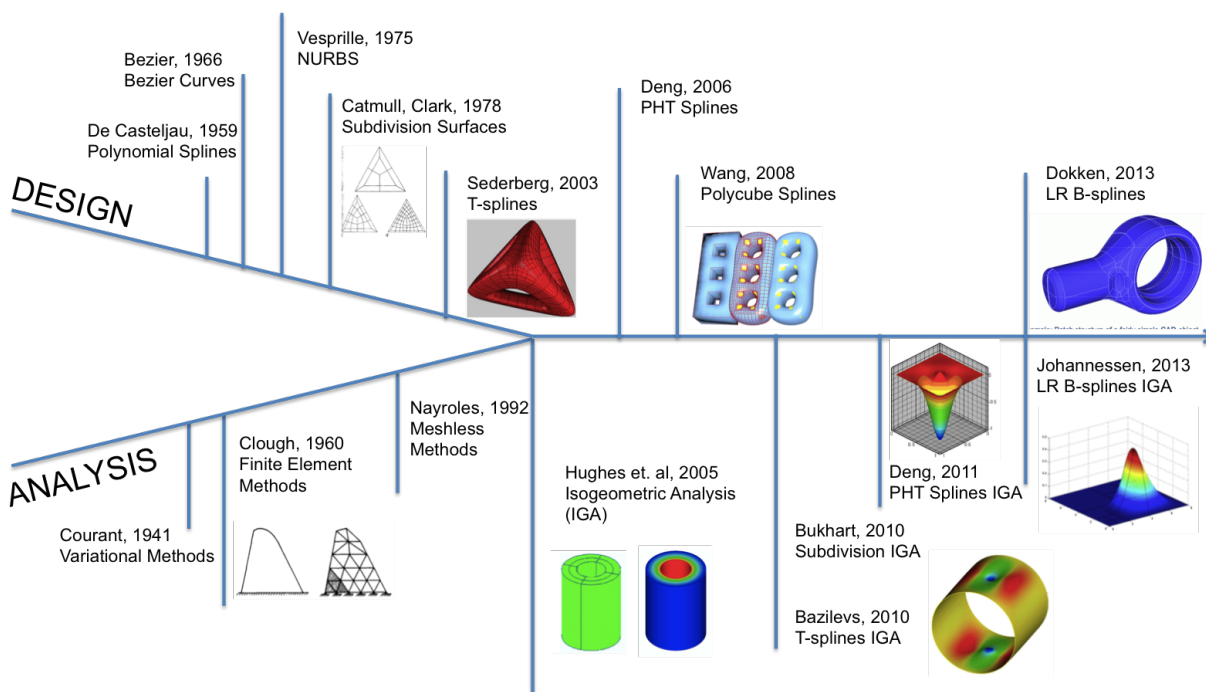


Figure 4.1: Design and Analysis developed independently and merged with the invention of IGA in 2005. Recent geometric representations and analysis on these representations are pushing forward the field.

Volume Representations For surface and shell analysis, a surface representation is obviously sufficient for IGA; however, a wide class of analysis problems require a full volumetric

domain (i.e. fluids). Developing a method and/or representation for a spline volume is a difficult task. Not much work has been done in this area as this has only recently come about with the concept of IGA. For design and visualization research, surface representations have been the key aspect of interest, and thus, little work has investigated spline volumes or trivariate splines. Whereas computational topology of surfaces are more or less understood, volumes are still an open book [146]. Recent work has attempted to make headway in understanding spline volume representations.

As a start, NURBS have been extended to volume representations in a few studies [135, 147]. NURBS volume construction is not straight forward, and a wide range of methods have been used. For example, Zhang et al. used sweeping methods for the construction of patient-specific vascular models [148].

T-splines, although used increasingly for surface representations, have yet to be used widely in volume representations. Escobar et al. provide methods to first generate a volumetric finite element mesh and then optimize this to obtain a T-spline volumetric representation [149]. Zhang et al. propose methods to convert a triangulated surface to a T-spline solid by initially mapping the surface to a cube [150]. The nature of this mapping restricts this method to only be useful for genus 0 surfaces or surfaces with no holes. Further investigations could look into directly converting a T-spline surface into a T-spline volume while maintaining the exact T-spline surface topology.

There are a few studies investigating subdivision volumes [151, 152]; however, it has been extended to IGA in a single study [153]. This study only investigates Catmull-Clark subdivisions and is restricted to only subdivision volumes of hexahedral elements. Subdivision volumes provide an interesting avenue for research because of the flexibility in the control mesh. New subdivision schemes can also be easily implemented and tested.

Because PHT splines are based on a 2D manifold T-mesh, they have not yet been extended to the volumetric case. Future work would include constructing a 3D T-mesh for the base structure of the polynomial splines.

In a single study, polycube splines have been extended to volumes with generalized polycubes [154]. Generalized polycubes are advantageous because they are able to represent complex geometries on a rigidly defined structure. This rigidly defined structure also makes these representations easily editable. However, decomposition into generalized polycubes brings extra singularities that can be difficult to handle. Future investigations can work to define an optimal generalized polycube representation that minimizes the number of singularities.

Although LR B-splines are easily extended to higher dimensions, they have yet to be extended to the volumetric domain for IGA. This should be a focus of future studies, as they have demonstrated promising attributes in surface problems.

Another area of research involves attempting to convert existing finite element meshes to spline volumes [155, 156, 157]. This is important work as there are large number of studies that already have a computed finite element mesh. These studies also help to shed light on the decomposition of a volume into a spline representation, which is very helpful in understanding how a spline surface might be extended to a spline volume.

Table 4.1 compares a few of the representations discussed in terms of their advantages and disadvantages to be used for IGA.

	NURBS	T-splines	Subdivision	PHT splines	LR B-splines
Local Refinement	No	Yes	Yes	Yes	Yes
Refinement Strategy	N/A	Control Point Addition	Arbitrary Structure	Spline Space Division	Spline Space Division
Guaranteed Partition of Unity	Yes	No	N/A	Yes	No
Linear Independence	Yes	Yes	N/A	Yes	Yes
Max Continuity	Controllable	Controllable	C_1 and C_2	C_1	Controllable
Single Watertight Patch ¹	No	Yes	Yes	Yes	No
NURBS compatible	N/A	Yes	No	Yes	Yes
Extended to Volume	Yes	Yes	Yes	No	No
Volume used in IGA	Yes	Yes	Yes	No	No

Table 4.1: Comparison of geometric representations that have been or have the potential to be used in IGA. Each representation has at least one drawback, which is highlighted in red and bolded.

Applications IGA is currently being used with varying geometric representations in many different applications.

- **Contact problems:** In contact problems, it is very important to have a desired level of continuity across surface boundaries. NURBS and T-splines already are an improvement over finite element meshes for contact problems because they can obtain a higher level of continuity across elements. It is important that new geometric representations satisfy continuity requirements for high levels of accuracy in contact problems. Current work in IGA for contact problems include [158, 159, 160].
- **Optimization problems:** In design optimization, a geometry that best satisfies a solution is desired. With IGA, the geometry can change, and the basis functions can be used immediately to run a new simulation. It is important that a slight change in geometry will not alter the representation too drastically. For efficient geometric representations, the basis used in the simulation should actually stay the same while only the control shape changes. Current work in optimization includes [161, 162, 163].
- **Fluid-structure interaction (FSI):** For FSI, it is important to have a high level of continuity between the fluid and solid boundary. In addition, it helps if the boundary between the two domains is smooth. With NURBS and newer geometric representations, this smooth boundary is much easier to capture than with a finite element mesh. Future geometric representations should ensure smoothness across all boundaries. FSI with IGA has been investigated in a couple studies [147, 164].
- **Large deformation:** Studies on large deformation with IGA show the true power of adaption [165, 166, 167]. With large deformations, it is very important the the geometric representation has the ability to refine and coarsen well both globally and locally.

4.4 Discussion

IGA relies on a concrete geometric representation that is ideal for both *design* and *analysis*. The CAD standard geometric representation NURBS is currently the most used geometric representation in IGA, but has some drawbacks. Some of the recent developments in surface and volume geometric representations show promising use in IGA, but further research is required. Certain representations, such as T-splines, are ideal for modeling complex geometries, but fail to extend well to volumes or do well in local refinement. There are certainly a number of potential investigations into geometric representations that are possible.

- Improved refinement strategies for T-splines: T-splines are extremely powerful because they can represent any geometry with one surface. Better refinement strategies will make T-splines more useful in IGA.

¹The representation can model a complex geometry as a single water-tight surface patch.

- Extension of T-splines to volumes: Very little work has been done to extend T-splines to volumes. This is due to the wide range of possible T-spline volumes that can represent the same object in combination with the extremely complex T-spline definition. A set of rules defining how a T-spline surface should be extended to a volume would improve the field greatly.
- Combination of LR B-splines on the polycube spline parametric framework: As indicated in Table 4.1, LR B-splines have many strengths; however, they have a fairly significant weakness in the fact that they cannot represent complex closed geometries. A combination of LR B-splines and polycubes would provide a better framework for representing complex geometries while still maintaining the nice properties of LR B-splines.
- Extension of LR B-splines to volumes: Although it is relatively trivial to extend LR B-splines to volumes, this has not yet been implemented. This could be valuable work and would allow a larger number of problems to be tackled.
- Conversion from subdivision surfaces to T-splines: In certain cases, subdivision surfaces can, and have been, converted to NURBS surfaces. Unfortunately, the cases in which subdivision surfaces can be converted to a NURBS surface are very limited because of the rigid structure of NURBS. Algorithms indicating whether a subdivision surface could be converted to a T-spline surface would advance research in this area greatly.

Chapter 5

Vascular Meshing for Isogeometric Analysis

Parts of this chapter follow work from [168] and are in collaboration with *Nathan M. Wilson* and *Shawn C. Shadden*. The methods described here have been extended to NURBS volumes and a manuscript is in preparation for journal submission.

5.1 Introduction

Patient-specific image-based modeling is commonly used in academic research to investigate physiological conditions where it is difficult to accurately or easily measure a physical quantity. Such modeling involves constructing a *geometric model* that is used as the computational domain for numerical simulations. For example, in image-based blood flow modeling [169, 16], computed tomography, magnetic resonance, or ultrasound image data is used to define a vascular region, which is volumetrically meshed and blood flow is simulated through the meshed domain. Other applications include modeling biomechanics of the airways [170], cerebrospinal fluid [171], bones [172], and joints [173] among others.

Medical image segmentation and model construction is a diverse and well-researched topic [174, 175]. In this context, image segmentation is defined as the process of extracting a manifold surface that encloses a region of interest from a 3D image. Image segmentation will result in a *boundary representation* surrounding the region of interest. The resulting boundary representation is typically a triangulated surface mesh, which is a *discrete* representation of the geometry. Although discrete representations have nice properties, it can be desirable to have a more design friendly representation. For example, a *parametric* representation, such as the non-uniform rational basis spline (NURBS) surface, which is the computer-aided design standard to represent a geometric model. A NURBS parameterization of a geometry is attractive due to its precision, flexibility and mathematical properties, and NURBS models are easier to combine (e.g., virtually fit a medical device), or systematically manipulate

(e.g., computationally optimize a surgical procedure).

When using an *analytic* or *parametric* computational model in design, it can be advantageous to ensure that the representation is *analysis suitable*. For a representation to be analysis suitable, the piecewise polynomial splines defining the model everywhere need to satisfy the requirements of a valid finite element basis. This allows the model to be directly supported in an isogeometric analysis framework. However, one of the main challenges of IGA is generating an analysis suitable parameterization of a complex domain [176]. This is a particular challenge for vascular geometries, which are geometrically complex due to bifurcations, drastic changes in feature size, curvature, tortuosity, etc.

As a method to generate analysis suitable representations, [155] introduced a technique that starts with a volumetric mesh and solves volumetric discrete harmonic equations to form a B-spline on the resultant parameterization. However it is generally necessary and desirable to form an analysis suitable volume from a boundary representation. In this regard, Aigner et al. [177] proposed a method to form simple analysis suitable genus-zero geometries by utilizing volume sweeping. [178] proposed methods to construct T-spline volumes from an input genus-zero surface. These methods work well for simpler geometries, but appear to break down for more complex geometries such as those encountered in image-based modeling. To handle vascular models, Zhang et al. [148] introduced a template-based approach to form analysis suitable volumes. While this method can effectively handle bifurcations, it appears to generally reduce the geometric detail of image-segmented geometries. For more complex models, this can be significant to modeling blood flow since simulation results are highly dependent on subtleties of the vascular morphology.

To generate a NURBS parameterization for more complex geometries, [179] generalized the concept of a polycube structure to decompose a geometry into a series of cubes and form volumetric splines from the generalized polycube structure. Akhras et al. [180] used a similar approach to generate analysis suitable NURBS surfaces from a variety of complex input surfaces by using a polycube decomposition. Using shape diameter functions and direction fields respectively, both methods obtain a pants decomposition of an arbitrary geometry. However, the pants decomposition is not unique, and depending on the technique used to obtain the decomposition and the topology of the geometry, it can result in warped patches. Updegrave et al. [168] provided a pipeline to create a NURBS surface of arbitrary patient-specific geometries utilizing the geometry's centerlines and an automatic axis-aligned polycube generation method. Despite this progress, the axis-aligned polycube provided a rigid parameterization domain and the final representation was a NURBS surface rather than a NURBS volume.

Thus, this chapter introduces methods to construct NURBS volumes of complex geometries starting from a triangulated surface mesh and utilizing a centerline structure. This work provides the following contributions:

- A procedure to extract centerlines from a triangulated surface using a cell-thinning method on the Voronoi diagram.

- A method to decompose an arbitrary vascular model into a graph simplification based on its centerlines.
- A novel and automated method to form a non axis-aligned polycube structure from the graph simplification.
- A new approach to decompose tubular structures using a centroidal voronoi tessellation based on the work of Hu et al. [181].
- A framework to convert a vascular geometry into an analysis suitable volumetric NURBS using polycubes.
- An implementation that links against widely-available, open-source tools such as the Visualization Tool Kit (VTK) [182] and the Vascular Modeling Tool Kit (VMTK) [82].

The main difficulty in converting an arbitrary discrete geometry into an analysis suitable parametric format is defining a globally valid parameter space. The CAD standard for modeling is non-uniform rational B-splines (NURBS). Converting to NURBS is particularly challenging due to the extremely rigid parameter space that is essentially a 2D structured grid for surfaces and a 3D structured grid for volumes. To convert to a NURBS surface, the discrete geometry needs to be decomposed into topological rectangles in which edges of the rectangles match to other edges, corners match to other corners, and there are no gaps in between rectangles. To convert to a NURBS volume, the volume encompassed by the discrete geometry needs to be decomposed into topological cuboids in which faces of the cuboids match to other faces, edges match to other edges, corners match to other corners, and again there are no gaps in between cuboids. This is exceedingly difficult for natural, free-form surfaces like those of vascular geometries as it is difficult to determine how and where to decompose the geometry into topological cuboids. This decomposition is simplified through the formation of a polycube structure. As part of this work, a novel method for automatically constructing a non-axis aligned polycube structure from a geometry's centerlines is outlined and described in detail. Creating parameterizations of single vessels is fairly straightforward, therefore the focus of this work is branched, bifurcating geometries. The input is a genus-zero triangulated surface model, $S = (\{T_i\}_{i=0}^{n_t-1}; \{v_j\}_{j=0}^{n_v-1})$, where n_t is the number of surface triangles T , and n_v is the number of surface vertices v . The following steps will be demonstrated using a model of the aortic arch with multiple branching vessels (Fig. 5.1(a)): (1) centerlines of the model are extracted to provide insight into the branching structure of the geometry, (2) radius information, classifier ids, and a local coordinate system are defined on the centerlines to make them meaningful and usable, (3) a *simplification graph* is created from the centerlines, (4) a non axis-aligned *polycube structure* is created from the *simplification graph*, (5) triangles on the input model are labelled according to the closest centerline, which defines *surface groups*, (6) each *surface group* is decomposed into cuboid *surface patches* using a centroidal voronoi tessellation (CVT), (7) each *surface patch* is mapped conformally to its corresponding cuboid patch on the *polycube structure*, (8) a structured grid defined on each group of the *polycube structure* is mapped back to the

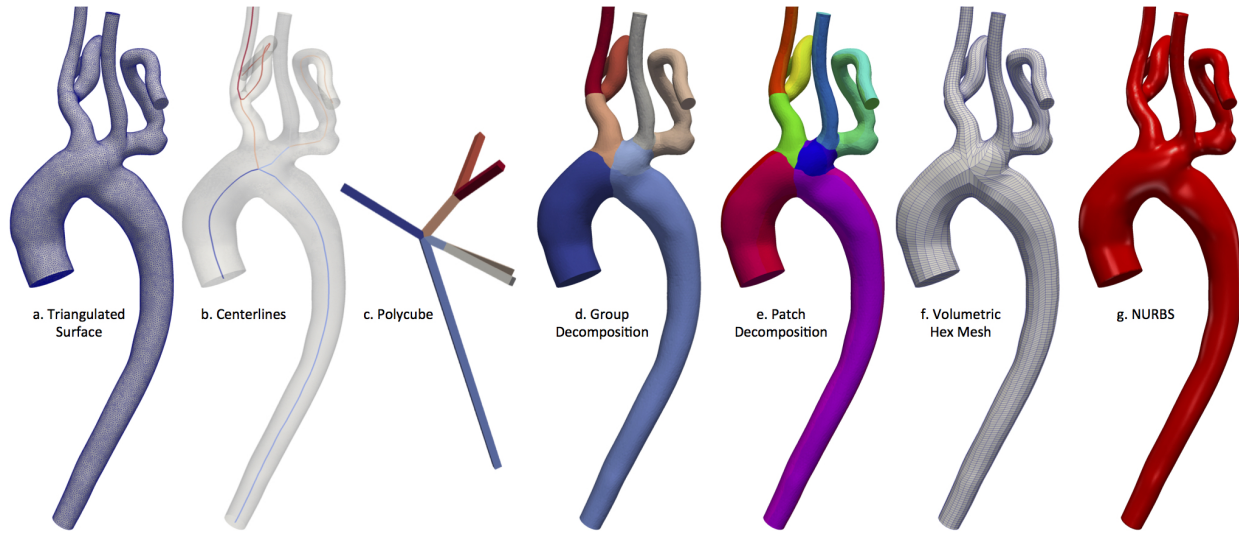


Figure 5.1: The steps in converting a discrete triangulated surface (a.) into an analysis suitable NURBS volume (b.).

input geometry, and (9) the structured grids are used as scaffolds to define a *globally-valid* volumetric non-uniform rational B-spline (NURBS) parameterization.

5.2 Centerline Extraction



Purpose:

The centerlines provide a simplified 1D representation of the geometry which aids in determining the branching structure of the geometry and where to place cuboid boundaries.

Methods:

Centerline, medial axis, or skeleton extraction refers to the well-known problem of extracting a one-dimensional representation of a geometry. It has garnered much research attention due to its applicability in a wide range of applications including computer graphics, mesh morphing, and solid modeling. There are a variety of methods to extract centerlines including distance field based methods [183, 184], potential field based methods [185], and cell-thinning methods [186]. Sobiecki et al. compare many of the different skeleton extraction methods for voxel shapes [187]. Though there are many useful techniques, cell-thinning methods have many benefits: (1) source and target seeds are not explicitly required by the user, (2) they are useful for non genus zero geometries, (3) they can produce medial axes of multiple dimensions (i.e. one-dimensional medial lines and two-dimensional medial curves), and (4) the extracted results do not heavily depend on the resolution of the input surface, thus, for surfaces of high resolution, computation time can be greatly reduced by coarsening the surface. The drawback to cell-thinning methods is that the resulting centerlines can be quite rough and they do not necessarily fulfill the strict definition of a *centerline* in which

Table 5.1: Inputs and Outputs for Centerline Extraction

Input	Definition	Data
Genus-zero triangulated surface model 	$S = (\{T_i\}_{i=0}^{n_t-1}; \{v_j\}_{j=0}^{n_v-1})$ $n_t = \text{number of surface triangles}$ $n_v = \text{number of surface vertices}$	N/A
Output	Definition	Data
Raw Centerlines 	$C = (\{L_i\}_{i=0}^{n_l-1}; \{p_j\}_{j=0}^{n_p-1})$ $n_l = \text{number of centerlines}$ $n_p = \text{number of centerline points}$	N/A

the minimal distance between centerline points and the surrounding geometry should be maximal. However, the cell-thinning method can reliably provide locations where segments of the geometry terminate and bifurcate. Thus, the cell-thinning method can be used to find termination and bifurcation points and then a minimization problem utilizing the radius values on the Voronoi diagram can be used to form true centerlines in between termination and bifurcation points. The cell-thinning method utilized in this work operates directly on a Voronoi diagram. The Voronoi diagram is the dual of the Delaunay triangulation in 2D and the Delaunay tetrahedralization in 3D. The reader is referred to a standard computational geometry textbook for the technical details of computing the Delaunay tetrahedralization and Voronoi diagram of a point set [188]. The visualization toolkit (VTK) is used to generate both the Delaunay tetrahedralization and its corresponding Voronoi diagram. Cell-thinning algorithms reduce a cell complex or a connected set of explicit geometric elements (i.e. edges and faces) to multi-dimensional medial axes. Because the Voronoi is the dual of the Delaunay tetrahedralization, the Voronoi diagram contains faces (dual of Delaunay edges), edges (dual of Delaunay triangles), and vertices (dual of Delaunay tetrahedrons). The Voronoi diagram is triangulated, so the faces consist of three connected edges and each edge consists of two connected vertices, making the Voronoi diagram a connected set of geometric elements and a valid cell complex. A straight-forward cell-thinning procedure [186] produces a medial axis of the input surface (Fig. 5.2).

During the cell-thinning procedure, edges and points on the outside of the Voronoi diagram are removed iteratively until no more edges or points can be removed. In this process,

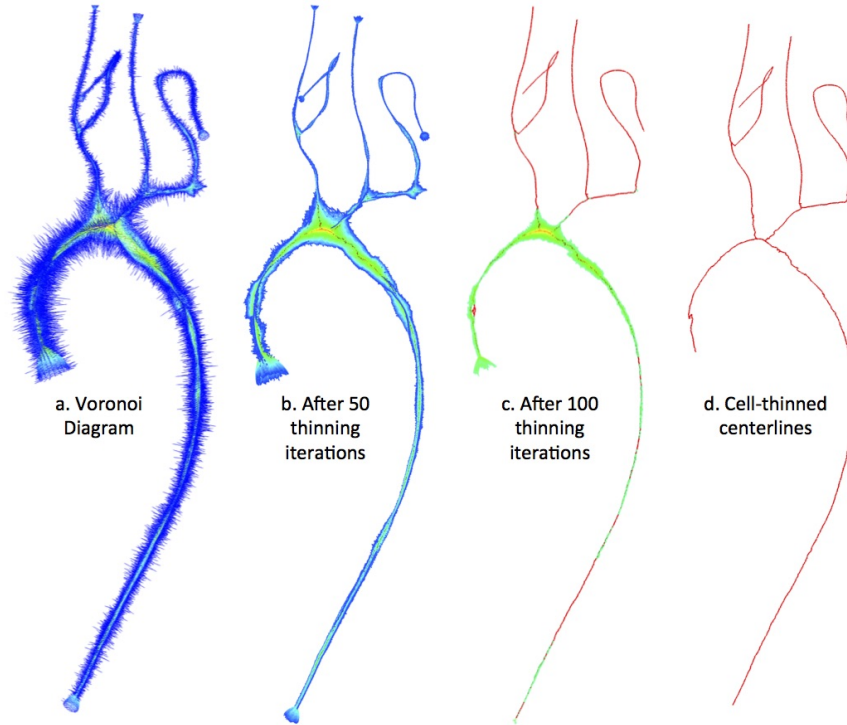


Figure 5.2: The edges of the Voronoi diagram (a.) are used as the initialization for a cell-thinning algorithm that produces a medial axis (d.)

the iteration at which an edge is exposed on the boundary, I , and the iteration at which an edge is removed, R , is retained. An absolute and relative measure of the medial persistence of each edge can be defined as

$$M_{abs} = R - I \quad (5.1)$$

$$M_{rel} = 1 - \frac{I}{R} \quad (5.2)$$

Thresholding with a high value of M_{abs} and a value close to 1 for M_{rel} provides the set of medial edges that tend to lie near the center of the Voronoi diagram. This produces a connected set of edges that lie near the interior of the Voronoi diagram; however, due to the nature of the cell-thinning procedure, the output may contain small spurious bifurcations. The small spurious bifurcations are removed when the number of points in the bifurcation is less than a specified threshold. The termination points are then found and one is defined as the source and all other as targets (Fig. 5.3). The source point is typically chosen to be the termination point with the largest radius value. For arterial modeling, the termination point of largest radius value typically corresponds to the inlet and is a valid assumption. Even if the termination point of largest radius value does not correspond to the inlet, it still is reasonable to use this point as the initialization point of the centerline tree.

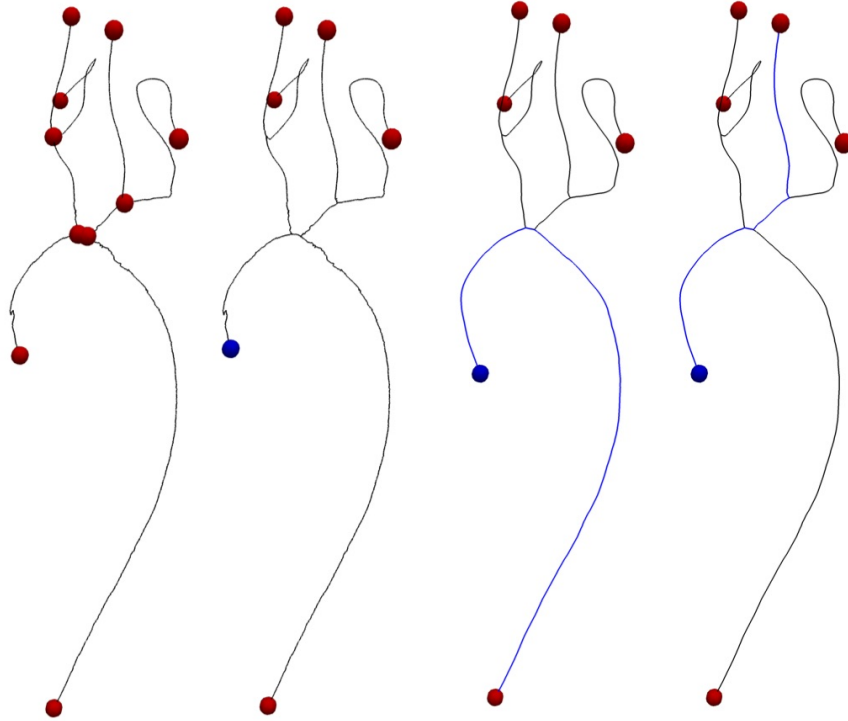


Figure 5.3: All termination and bifurcation points are located on the cell-thinned centerline. One point is chosen as the source point (blue) and the targets (red) are processed one by one to find the complete centerline paths for each target.


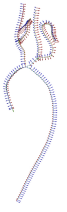
Starting at the chosen source point, the connected set of points and edges is processed recursively to find the set of bifurcation points leading to each target point. In between each consecutive set of points, a smooth and true centerline is found by solving a minimization problem on the Voronoi diagram,

$$E(C(p)) = \int_{p_0}^{p_1} G(C(p)) dp \quad (5.3)$$

where $C(p)$ is the set of points minimizing the energy $E(C(p))$ between two points on the Voronoi diagram, p_0 and p_1 , and $G(\cdot)$ is some cost function. In this case, the exact center of the geometry is desired (i.e. the location of maximum radius value); therefore, the cost function is the inverse of the radius function, $G = \frac{1}{R(x)}$, where $R(x)$ is the radius value at every point, x , on the Voronoi diagram. In this manner, the path of steepest descent between two points will lie along the locations of maximum radius values. This minimization problem can be rewritten into a minimal cost path problem using the Eikonal equation,

$$|\nabla\tau(x)| = \frac{1}{R(x)} \quad (5.4)$$

Table 5.2: Inputs and Outputs for Centerline Processing

Input	Definition	Data
Raw Centerlines 	$C = (\{L_i\}_{i=0}^{n_l-1}; \{p_j\}_{j=0}^{n_p-1})$ $n_l = \text{number of centerlines}$ $n_p = \text{number of centerline points}$	N/A
Output	Definition	Data
Merged centerlines with data 	$M = (\{L_i\}_{i=0}^{n_m-1}; \{p_j\}_{j=0}^{n_p-1})$ $n_m = \text{number of merged centerlines}$ $n_p = \text{number of centerline points}$	$\text{Radius } (p)$ $\text{LocalCoordinateSystem } (p)$ $\text{GroupId } (L)$

where $\tau(x)$ is the time it takes to travel on the Voronoi diagram from some initialization point. This method is implemented in the vascular modeling toolkit (VMTK) using the Fast Marching Method [189] and more details on the centerline extraction method can be found here [190]. The minimal cost path is found between each set of two consecutive points to complete the entire centerline tree (Fig. 5.3). For example, for a path that has two bifurcations between the source point and the target point, the minimal cost path problem is computed three times: (1) source point to bifurcation point 1, (2) bifurcation point 1 to bifurcation point 2, and (3) bifurcation point 2 to the target point. Once the Eikonal equation has been solved between all consecutive sets of points, the centerline pieces are stitched together to form a structured, ordered, and smooth raw centerline structure in which there is a line for each target point connecting it to the source point.

5.3 Centerline Processing

Purpose:

The raw centerlines can only be used minimally without additional information defined on the centerlines. Radius information and local coordinate systems provide useful information for geometric algorithms performed later.

Methods:

The raw centerline structure provides a nice visual of the medial axis of the geometry,

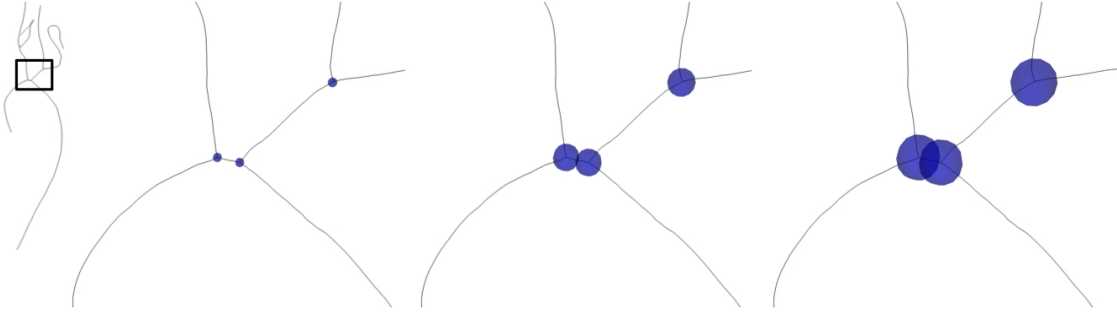


Figure 5.4: Identification of bifurcation regions (blue spheres). Close bifurcations are merged if bifurcation regions overlap.

but additional information is needed to use the centerlines for useful geometric processing algorithms. Therefore, three additional steps are taken to get this information: (1) the centerlines are processed into a format similar to VMTK with attached cell and point data describing radius, branch, and bifurcation information, (2) the centerlines are merged to reduce centerline complexity and provide one centerline per branch, and (3) a local coordinate system is defined on the merged centerlines.

The data attached to the centerlines is very similar to what is provided in VMTK (<http://www.vmtk.org/tutorials/BranchSplitting.html>), making it possible to use other centerlines processing algorithms in VMTK. First, radius information is attached to all centerline points. Because the centerline points were taken directly from the Vornoi diagram, it is easy to pass the information onto the centerline points. Next, the centerlines are merged by defining bifurcation regions on the centerlines. A bifurcation region encompasses the points within a specified distance from each bifurcation point. The distance is specified by the user and can either be an absolute distance or a percentage of the radius value at the bifurcation point. At each bifurcation point, the centerline is then split up into three tracts: (1) Points before the bifurcation region, (2) points within the bifurcation region, and (3) points after the bifurcation region. Every tract is given a unique *GroupId* as well as information indicating whether or not it is a bifurcation region. If bifurcation regions overlap, the bifurcations are merged into a trifurcation (Fig. 5.4). Thus, the user has control over whether close bifurcations are merged into trifurcations, quadfurcations, etc. A larger user-specified merged distance will result in more branches being merged together. Duplicate lines are deleted, bifurcation tracts are removed, and the average bifurcation point is attached to all touching centerlines at each bifurcation, trifurcation, etc. The *GroupId* defining the tracts is carried over to define a unique id for each centerline branch. The result is a set of merged centerlines, $M = \{L_i\}_{i=0}^{n_l-1}$, where n_l corresponds to the number of branches in the geometry.

The final centerline processing step involves defining a local coordinate system. At each bifurcation point, j , a coordinate frame, $\{\hat{e}_{b_{j_0}}, \hat{e}_{b_{j_1}}, \hat{e}_{b_{j_2}}\}$, is computed based on the parent and child centerline directions. The parent direction vector, \hat{l}_{parent} , is the normalized difference

between the last two vertices of the branch. For each child, k , the child direction vector, \hat{l}_{child_k} , is the normalized difference between the first two vertices. To determine how much the child centerlines diverge from the parent centerline, the angle, θ_k , between the parent and each child centerline k is computed, where

$$\theta_k = \text{atan2} \left(\frac{\|\hat{l}_{child_k} \times \hat{l}_{parent}\|}{\hat{l}_{child_k} \cdot \hat{l}_{parent}} \right). \quad (5.5)$$

The child centerline that diverges least (i.e. largest angle) from the parent branch is defined as the *aligning branch*, while the child centerline that diverges most (i.e. smallest angle) from the parent branch is defined as the *diverging branch*. The bifurcation coordinate frame is then defined using these vectors. The first vector of the bifurcation orthonormal basis is the parent centerline, $\hat{e}_0 = \hat{l}_{parent}$. The remainder of the orthonormal basis is computed using the diverging centerline,

$$\begin{aligned} \hat{e}_2 &= \hat{e}_0 \times \hat{l}_{diverging} \\ \hat{e}_1 &= \hat{e}_2 \times \hat{e}_0 \end{aligned} \quad (5.6)$$

This defines a coordinate system at a every bifurcation point, but the entire centerline tree needs to be populated with a connected and smooth coordinate frame. To do this, a *parallel transport frame* is used, which is an orthonormal coordinate system computed by traversing along the line or curve of interest and rotating the coordinate system to mimic the change in direction of the curve [191]. Starting at the initial bifurcation point, the centerlines are traversed in each direction until either a termination point or another bifurcation point is reached (Fig. 5.5). At each point, i , along the centerline, the *parallel transport frame* is defined as $\{\vec{t}_i, \vec{n}_i, \vec{b}_i\}$, where \vec{t}_i is the difference between two consecutive centerline points, $\mathbf{p}_{i+1} - \mathbf{p}_i$, and \vec{n}_i and \vec{b}_i are the two normal components completing the orthonormal frame. At each successive point, a vector \vec{v}_i is computed by rotating \vec{n}_i around $\vec{u}_i = \vec{t}_i \times \vec{t}_{i+1}$ by the angle $\alpha_i = \text{acos}(\vec{t}_i \cdot \vec{t}_{i+1})$. After the orthonormal frame is rotated, \vec{v}_i is projected onto \vec{t}_{i+1} and the normal components are updated to the the new frame.

$$\begin{aligned} \vec{n}_{i+1} &= \vec{v}_i - [\vec{t}_{i+1} \cdot \vec{v}_i] \vec{t}_{i+1} \\ \vec{b}_{i+1} &= \vec{t}_{i+1} \times \vec{n}_{i+1} \end{aligned} \quad (5.7)$$

When another bifurcation point, j , is reached, the bifurcation coordinate system already defined needs to match the parallel transport frame. The parallel transport frame is compared to the bifurcation coordinate system, $\{\hat{e}_{b_{j_0}}, \hat{e}_{b_{j_1}}, \hat{e}_{b_{j_2}}\}$. Specifically, the normal component of the parallel transport frame, \vec{n} , is compared to the normal components of the bifurcation coordinate frame, $\hat{e}_{b_{j_1}}$ and $\hat{e}_{b_{j_2}}$ (Fig. 5.6). The normal component of the bifurcation coordinate frame that it aligns with most is used as the matching direction. In order to get the

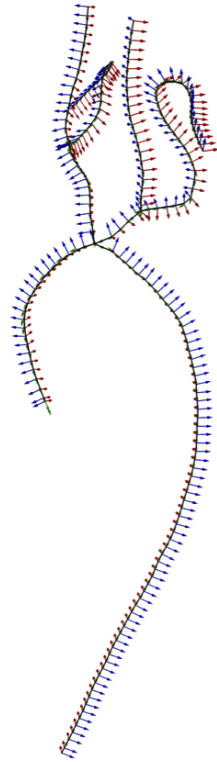


Figure 5.5: The local coordinate system is defined along the centerlines using a parallel transport frame and comparing to the local coordinate system defined at each bifurcation point.

traversed parallel transport frame to match, the angle between the parallel transport frame and the matching direction is computed, $\beta = \text{atan2}\left(\frac{\|\hat{n} \times \hat{e}_{matching}\|}{\hat{n} \cdot \hat{e}_{matching}}\right)$. The angle β is then divided by the number of points along the centerline and this angle contribution factor is added to the normal vector at each point during another traversal of the centerline. After adjusting the parallel transport frame, the local coordinate system will transition smoothly at the bifurcation (Fig. 5.7). This is done for each centerline until all termination points have been reached and there is a local coordinate system defined on each centerline point.

5.4 Graph Simplification

Purpose:

The geometry needs to be abstracted into a polycube structure that recognizably represents the input geometry. The simplification graph reduces the centerline structure to a set of connected nodes in a tree-like structure that acts as a skeleton for the polycube. The

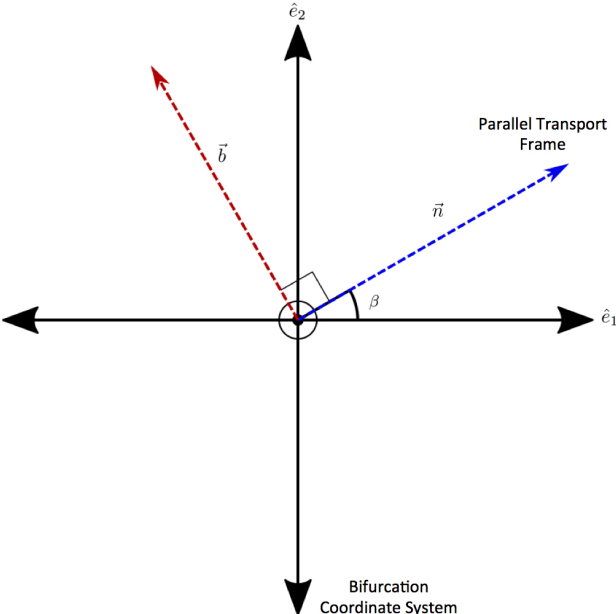


Figure 5.6: The comparison between the parallel transport frame and a local bifurcation coordinate system. The parallel transport will be update by angle β so that the coordinate frames match at the bifurcation point.

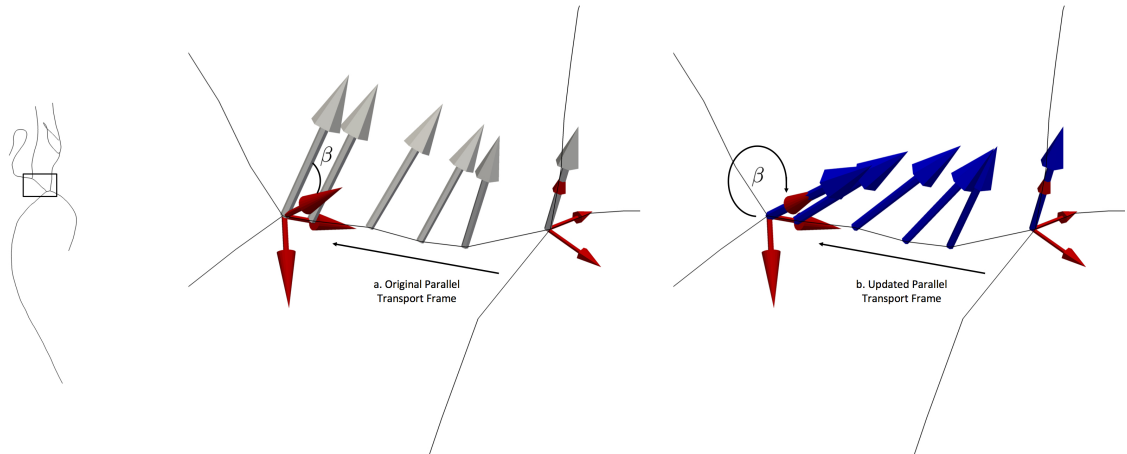




Figure 5.7: (a.) The parallel transport frame is compared to the local bifurcation coordinate system. (b.) The transport frame is adjusted by the angle β to match with the bifurcation coordinate system.

Table 5.3: Inputs and Outputs for Graph Simplification

Input	Definition	Data
Merged centerlines with data 	$M = (\{L_i\}_{i=0}^{n_m-1}; \{p_j\}_{j=0}^{n_p-1})$ $n_m = \text{number of merged centerlines}$ $n_p = \text{number of centerline points}$	$Radius (p)$ $LocalCoordinateSystem (p)$ $GroupId (L)$
Output	Definition	Data
Simplification Graph 	$G = \{g_i\}_{i=0}^{n_g-1}$ $n_g = \text{number of graph nodes}$	$Parent (g)$ $Children (g)$ $GroupId (g)$ $DivergingChild (g)$ $AligningChild (g)$ $EndPoint (g)$ $Direction (g)$

simplification graph also eases subsequent operations by providing a quick lookup of parent, child, and sibling nodes as well as a variety of model attributes.

Methods:

Each node of the graph corresponds to a branch in the geometry and contains a parent reference and two to three child references. The initial node or branch has no parent node (the root), and termination nodes have no children. Data is contained on the node indicating the corresponding *GroupId*, the diverging and aligning children, and start and end points for the physical representation of the graph. The final piece of information contained on the graph indicates the direction of the branch node. The direction refers to the direction in which the cuboid for the branch will be built when the polycube is constructed. The cuboids will be built at a specified angle from the the parent cuboid in one of four directions: (0) *RIGHT*, (1) *BACK*, (2) *LEFT*, and (3) *FRONT*. Computing each branch direction follows two steps in which the direction is first defined locally and then updated to a global direction. The same computation carried out in section 5.3 is used to find the diverging child at each bifurcation. The diverging branch is given the default direction of *RIGHT*. The other branches are then compared to the diverging branch and given a corresponding direction based on the angle

made with the diverging branch,

$$Direction = \begin{cases} RIGHT, & \text{if } \Theta > 7\pi/4 \text{ or } \Theta \leq \pi/4 \\ BACK, & \text{if } \pi/4 < \Theta \leq 3\pi/4 \\ LEFT, & \text{if } 3\pi/4 < \Theta \leq 5\pi/4 \\ FRONT, & \text{if } 5\pi/4 < \Theta \leq 7\pi/4 \end{cases} \quad (5.8)$$

where Θ is the angle between a child branch vector \hat{l}_{child} and the diverging child branch vector $\hat{l}_{diverging}$.

These directions are sound locally, but need to be updated corresponding to the updates made to the local coordinate system in section 5.3. Again, the same computation carried out in section 5.3 is used to find the update to each bifurcation coordinate system, $\{\hat{e}_{b_0}, \hat{e}_{b_1}, \hat{e}_{b_2}\}$, based on the comparison to the parallel transport frame, $\{\vec{t}, \vec{n}, \vec{b}\}$. Depending on how the coordinate frames compare, an update to the direction may be required.

$$Update = \begin{cases} 0, & \text{if } |\vec{n} \cdot \hat{e}_{b_{j_1}}| \geq |\vec{n} \cdot \hat{e}_{b_{j_2}}| \text{ and } \vec{n} \cdot \hat{e}_{b_{j_1}} \geq 0 \\ 1, & \text{if } |\vec{n} \cdot \hat{e}_{b_{j_1}}| < |\vec{n} \cdot \hat{e}_{b_{j_2}}| \text{ and } \vec{n} \cdot \hat{e}_{b_{j_1}} < 0 \\ 2, & \text{if } |\vec{n} \cdot \hat{e}_{b_{j_1}}| \geq |\vec{n} \cdot \hat{e}_{b_{j_2}}| \text{ and } \vec{n} \cdot \hat{e}_{b_{j_1}} < 0 \\ 3, & \text{if } |\vec{n} \cdot \hat{e}_{b_{j_1}}| < |\vec{n} \cdot \hat{e}_{b_{j_2}}| \text{ and } \vec{n} \cdot \hat{e}_{b_{j_1}} \geq 0 \end{cases} \quad (5.9)$$

The update number is added to the old direction to produce the new direction,

$$Direction_{new} = mod(Direction + Update, 4) \quad (5.10)$$

The physical end points are computed using this direction, the angle between parent and child branches, and the length of the centerline. This centerline graph now contains all the information to build a non-axis aligned polycube structure.

5.5 Polycube Generation




Purpose:

In order to define a globally valid parameterization, a polycube structure that is similar to the input geometry needs to be constructed.

Methods:

A novel polycube structure is introduced in which the polycubes are not axis-aligned in all three global coordinate directions. Instead, the child cuboids branch off parent cuboids at the same angle at which the child centerlines branch from the parent centerline. Freedom in one direction reduces the distortion in the final parameterization, increases the flexibility of the polycube structure, and provides a branching-type polycube that is ideal for vascular geometries. Each cuboid attaches to the rest of the polycube structure at either one or two locations. Termination branches attach to the polycube structure at just one location,

Table 5.4: Inputs and Outputs for Polycube Generation

Input	Definition	Data
<p>Simplification Graph</p> 	$G = \{g_i\}_{i=0}^{n_g-1}$ <p>$n_g =$ number of graph nodes</p>	<p><i>Parent</i> (g)</p> <p><i>Children</i> (g)</p> <p><i>GroupId</i> (g)</p> <p><i>DivergingChild</i> (g)</p> <p><i>AligningChild</i> (g)</p> <p><i>EndPoints</i> (g)</p> <p><i>Direction</i> (g)</p>
Output	Definition	Data
<p>Surface Polycube</p> 	$P_s = \{c_i\}_{i=0}^{n_c-1}$ <p>$n_c =$ number of cuboid faces (i.e. number of patches)</p>	<p><i>GroupId</i> (c)</p> <p><i>PatchId</i> (c)</p>
<p>Volumetric Polycube</p> 	$P_v = \{s_i\}_{i=0}^{n_s-1}$ <p>$n_s =$ number of structured grid volumes (i.e. number of branches)</p>	<p><i>GroupId</i> (s)</p>

while connecting branches attach to the polycube at two locations (*top* and *bottom*). At these attachment locations (bifurcations or trifurcations), information about the connecting branches is needed. Based on the number and directions of the connecting branches, a certain type of cuboid will be built. Bifurcations, in-plane trifurcations, and out-of-plane trifurcations are all handled as part of this work (Fig: 5.9).

There are total of 11 different possible *top* cuboid types and 7 different possible *bottom* cuboid types (Fig. 5.8). Using the end points of each graph node, a cuboid is built around the graph node line (i.e. a 2D square is built at each end and the corners are connected). At the *top* and *bottom* of the cuboid, points are added and shifted where needed to form the correct end type. Each cuboid is built as a series of polygonal faces. On each face, the corresponding *GroupId* is defined as well as the *PatchId* that will be used for that cuboid face. All polygonal faces are added to an unstructured grid data structure, and this defines

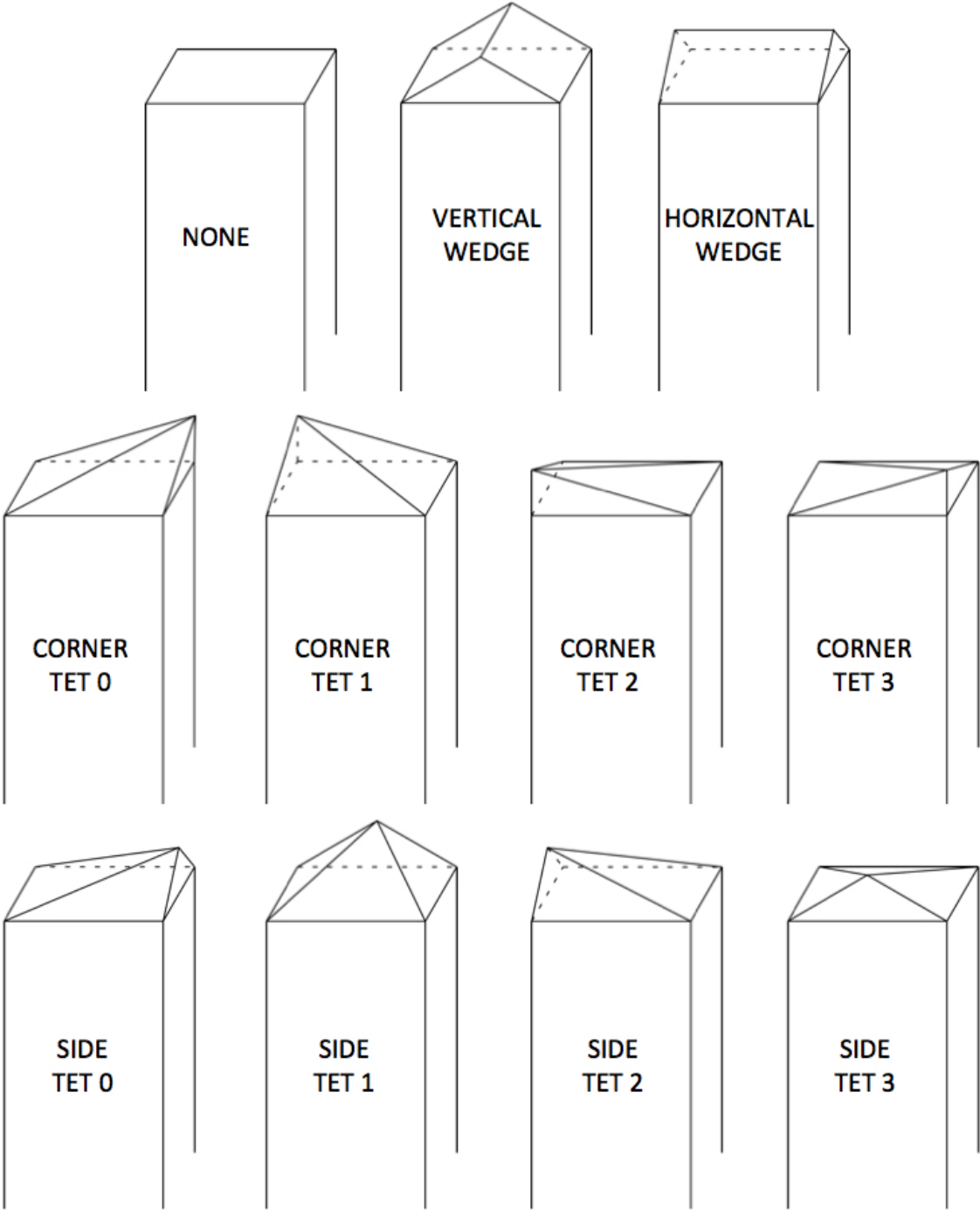


Figure 5.8: The different cuboid types. The wedge end types are used primarily for bifurcations while the corner and side tetrahedrons are used primarily for trifurcations.

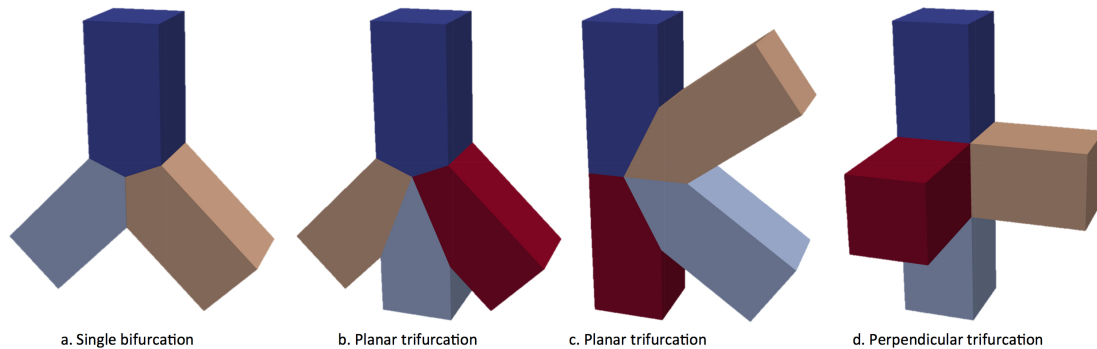


Figure 5.9: Both bifurcations and trifurcations are handled as part of this work. Bifurcations are straight forward and are all variations of polycube (a.). The trifurcations can contain a third branch that is either in-plane with the other two child branches (b., c.) or a third branch that is out-of-plane with the other two child branches (d.).

the surface polycube.

Now, the volumetric polycube of desired resolution is constructed on top of the surface polycube. The height, width, and the number of divisions along the height and width is specified by the user. The length varies from branch to branch and is calculated from the length of the graph node line. For each cuboid, a structured grid is formed by looping through the height, width, and length and adding points to match the specified resolution. Each structured grid cuboid is concatenated together into one full polycube structure.

5.6 Graph Simplification



Purpose:

The geometry needs to be abstracted into a polycube structure that recognizably represents the input geometry. The simplification graph reduces the centerline structure to a set of connected nodes in a tree-like structure that acts as a skeleton for the polycube. The simplification graph also eases subsequent operations by providing a quick lookup of parent, child, and sibling nodes as well as a variety of model attributes.

Methods:

Each node of the graph corresponds to a branch in the geometry and contains a parent reference and two to three child references. The initial node or branch has no parent node (the root), and termination nodes have no children. Data is contained on the node indicating the corresponding *GroupId*, the diverging and aligning children, and start and end points for the physical representation of the graph. The final piece of information contained on the graph

Table 5.5: Inputs and Outputs for Graph Simplification

Input	Definition	Data
Merged centerlines with data 	$M = (\{L_i\}_{i=0}^{n_m-1}; \{p_j\}_{j=0}^{n_p-1})$ $n_m = \text{number of merged centerlines}$ $n_p = \text{number of centerline points}$	$\text{Radius } (p)$ $\text{LocalCoordinateSystem } (p)$ $\text{GroupId } (L)$
Output	Definition	Data
Simplification Graph 	$G = \{g_i\}_{i=0}^{n_g-1}$ $n_g = \text{number of graph nodes}$	$\text{Parent } (g)$ $\text{Children } (g)$ $\text{GroupId } (g)$ $\text{DivergingChild } (g)$ $\text{AligningChild } (g)$ $\text{EndPoints } (g)$ $\text{Direction } (g)$

indicates the direction of the branch node. The direction refers to the direction in which the cuboid for the branch will be built when the polycube is constructed. The cuboids will be built at a specified angle from the the parent cuboid in one of four directions: (0) *RIGHT*, (1) *BACK*, (2) *LEFT*, and (3) *FRONT*. Computing each branch direction follows two steps in which the direction is first defined locally and then updated to a global direction. The same computation carried out in section 5.3 is used to find the diverging child at each bifurcation. The diverging branch is given the default direction of *RIGHT*. The other branches are then compared to the diverging branch and given a corresponding direction based on the angle made with the diverging branch,

$$\text{Direction} = \begin{cases} \text{RIGHT}, & \text{if } \Theta > 7\pi/4 \text{ or } \Theta \leq \pi/4 \\ \text{BACK}, & \text{if } \pi/4 < \Theta \leq 3\pi/4 \\ \text{LEFT}, & \text{if } 3\pi/4 < \Theta \leq 5\pi/4 \\ \text{FRONT}, & \text{if } 5\pi/4 < \Theta \leq 7\pi/4 \end{cases} \quad (5.11)$$

where Θ is the angle between a child branch vector \hat{l}_{child} and the diverging child branch vector $\hat{l}_{diverging}$.

These directions are sound locally, but need to be updated corresponding to the updates made to the local coordinate system in section 5.3. Again, the same computation carried out in section 5.3 is used to find the update to each bifurcation coordinate system, $\{\hat{e}_{b_0}, \hat{e}_{b_1}, \hat{e}_{b_2}\}$, based on the comparison to the parallel transport frame, $\{\vec{t}, \vec{n}, \vec{b}\}$. Depending on how the

coordinate frames compare, an update to the direction may be required.

$$Update = \begin{cases} 0, & \text{if } |\vec{n} \cdot \hat{e}_{b_{j_1}}| \geq |\vec{n} \cdot \hat{e}_{b_{j_2}}| \quad \text{and} \quad \vec{n} \cdot \hat{e}_{b_{j_1}} \geq 0 \\ 1, & \text{if } |\vec{n} \cdot \hat{e}_{b_{j_1}}| < |\vec{n} \cdot \hat{e}_{b_{j_2}}| \quad \text{and} \quad \vec{n} \cdot \hat{e}_{b_{j_1}} < 0 \\ 2, & \text{if } |\vec{n} \cdot \hat{e}_{b_{j_1}}| \geq |\vec{n} \cdot \hat{e}_{b_{j_2}}| \quad \text{and} \quad \vec{n} \cdot \hat{e}_{b_{j_1}} < 0 \\ 3, & \text{if } |\vec{n} \cdot \hat{e}_{b_{j_1}}| < |\vec{n} \cdot \hat{e}_{b_{j_2}}| \quad \text{and} \quad \vec{n} \cdot \hat{e}_{b_{j_1}} \geq 0 \end{cases} \quad (5.12)$$

The update number is added to the old direction to produce the new direction,

$$Direction_{new} = \text{mod}(Direction + Update, 4) \quad (5.13)$$

The physical end points are computed using this direction, the angle between parent and child branches, and the length of the centerline. This centerline graph now contains all the information to build a non-axis aligned polycube structure.

5.7 Surface Group Decomposition

Purpose: The geometry needs to be split into groups that match the centerlines and the polycube structure.

Methods:

In the group decomposition, each triangle, T , on the surface is assigned a *GroupId* corresponding to the *GroupId* of the *closest* centerline. When computing the distance to each centerline point, the radius is taken into account for a modified distance function,

$$dist(\mathbf{x}_c(T_i), \mathbf{p}_j) = \sqrt{\|\mathbf{x}_c(T_i) - \mathbf{p}_j\|^2 - r_j^2} \quad (5.14)$$

where $\mathbf{x}_c(T_i)$ is the centroid of i^{th} triangle, \mathbf{p}_j is the j^{th} centerline point, and r_j is the radius value at p_j . The centerline point with the minimum value of the modified distance function is found for every triangle on the surface. After a *GroupId* has been assigned to every triangle on the surface, the surface group decomposition is compared to the polycube. Each group is checked for consistency with the matching polycube group. The following rules apply to each surface group: (1) the group should contain only one connected region, (2) the edges should touch the same groups as the matching polycube group edges, and (3) the corner points should touch the same groups as the matching polycube corner points. In most cases, the surface group decomposition already matches the group topology of the polycube. However, in cases of many close bifurcations and trifurcations, the group may differ from the polycube topology, and it must be adjusted to be able to form a globally valid parameterization.

5.8 Surface Patch Decomposition

Purpose:

The surface of each group needs to be clustered into patches that match the sides of the corresponding group cuboid in the polycube structure.

Table 5.6: Inputs and Outputs for Surface Group Decomposition







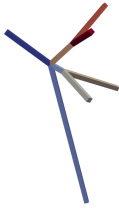

Input	Definition	Data
<p>Grouped genus-zero triangulated surface model</p> 	$S_g = (\{T_i\}_{i=0}^{n_t-1}; \{v_j\}_{j=0}^{n_v-1})$ <p>n_t = number of surface triangles n_v = number of surface vertices</p>	<p><i>GroupId</i> (T)</p>
<p>Merged centerlines with data</p> 	$M = (\{L_i\}_{i=0}^{n_m-1}; \{p_j\}_{j=0}^{n_p-1})$ <p>n_m = number of merged centerlines n_p = number of centerline points</p>	<p><i>Radius</i> (p) <i>LocalCoordinateSystem</i> (p) <i>GroupId</i> (L)</p>
<p>Surface Polycube</p> 	$P_s = \{c_i\}_{i=0}^{n_c-1}$ <p>n_c = number of cuboid faces (i.e. number of patches)</p>	<p><i>GroupId</i> (c) <i>PatchId</i> (c)</p>
Output	Definition	Data
<p>Grouped genus-zero triangulated surface model</p> 	$S_g = (\{T_i\}_{i=0}^{n_t-1}; \{v_j\}_{j=0}^{n_v-1})$ <p>n_t = number of surface triangles n_v = number of surface vertices</p>	<p><i>GroupId</i> (T)</p>

Table 5.7: Inputs and Outputs for Surface Patch Decomposition

Input	Definition	Data
Grouped genus-zero triangulated surface model 	$S_g = (\{T_i\}_{i=0}^{n_t-1}; \{v_j\}_{j=0}^{n_v-1})$ $n_t = \text{number of surface triangles}$ $n_v = \text{number of surface vertices}$	$GroupId(T)$
Merged centerlines with data 	$M = (\{L_i\}_{i=0}^{n_m-1}; \{p_j\}_{j=0}^{n_p-1})$ $n_m = \text{number of merged centerlines}$ $n_p = \text{number of centerline points}$	$Radius(p)$ $LocalCoordinateSystem(p)$ $GroupId(L)$
Surface Polycube 	$P_s = \{c_i\}_{i=0}^{n_c-1}$ $n_c = \text{number of cuboid faces}$ (i.e. number of patches)	$GroupId(c)$ $PatchId(c)$
Output	Definition	Data
Patched genus-zero triangulated surface model 	$S_p = (\{T_i\}_{i=0}^{n_t-1}; \{v_j\}_{j=0}^{n_v-1})$ $n_t = \text{number of surface triangles}$ $n_v = \text{number of surface vertices}$	$GroupId(T)$ $PatchId(T)$

Methods:

For decomposition into patches, a modified version of the skeleton-based harmonic boundary enhanced centroidal Voronoi tessellation (HBECVT) [192] scheme with k-means clustering is utilized. A k-means clustering algorithm partitions data into Voronoi regions, $V = \{V_i\}_{i=0}^{k-1}$, based on a set of clusters or generators, $C = \{c_i\}_{i=0}^{k-1}$, in which each piece of data belongs to the cluster with the closest mean. In the HBECVT, every triangle, T , is assigned to one of six clusters, $k = 6$, (corresponding to the six sides of a cube), where the six clusters are the positive and negative vectors of the global coordinate system, $\{\hat{e}_0, \hat{e}_1, \hat{e}_2, -\hat{e}_0, -\hat{e}_1, -\hat{e}_2\}$. To assign triangles to one of the six global coordinate clusters, the set of normals, $\mathbf{X} = \{\hat{n}(T_j)\}_{j=0}^{n_t-1}$, are compared to the coordinate directions, where $\hat{n}(T_j)$ is the cell normal of the j^{th} triangle. The Voronoi region corresponding to each cluster, c_i , is then defined as

$$V_i = \{\hat{n}(T_j) \in \mathbf{X} : \text{dist}(\hat{n}(T_j), c_i) \leq \text{dist}(\hat{n}(T_j), c_r), \text{ for } r = 1, \dots, k\}, \quad (5.15)$$

where the distance function is a modified distance function that contains an additional component, $\tilde{n}_i(T_j)$.

$$\text{dist}(\hat{n}(T_j), c_i) = \sqrt{\|\hat{n}(T_j) - c_i\|^2 + \eta \tilde{n}_i(T_j)}. \quad (5.16)$$

The $\tilde{n}_i(T_j)$ weights the distance based on the neighborhood of T_j and η is positive weighting factor controlling the influence of $\tilde{n}_i(T_j)$. Specifically, $\tilde{n}_i(T_j)$ indicates the number of elements within a neighborhood $N_\omega(T_j)$ that do not belong to the same cluster as T_j . The size of the neighborhood, $N_\omega(T_j)$, is controlled by ω , where ω designates the number of rings around T_j . A neighborhood of $\omega = 1$ includes all triangles touching a vertex of T_j . All succeeding neighborhoods, $\omega = k$, for $k > 1$, include all triangles touching a vertex of the triangles in the neighborhood $\omega = k - 1$ as well as all triangles in neighborhoods $\omega < k$. This contribution allows the cluster of T_j to be influenced by its neighborhood. If there are many triangles in $N_\omega(T_j)$ that have been assigned to a different cluster than T_j , then $\tilde{n}_i(T_j)$ is large and the distance function returns a larger value (i.e. the likelihood that it actually belongs to that cluster is lower). Alternatively, if $\tilde{n}_i(T_j)$ is small, then the likelihood that it belongs to that cluster is high. The energy of the HBECVT can be written and defined in the normal space,

$$E^H(C; U) = \sum_{j=0}^{n-1} \left[k / \left(\sum_{i=0}^{k-1} \|\hat{n}(T_j) - c_i\|^2 + \eta \tilde{n}_i(T_j) \right) \right] \quad (5.17)$$

where U is any tessellation of the data. Once the energy is minimized, the clustering $V = U$. The HBECVT provides a good decomposition of objects in which the sides of the object align well with the global coordinate space; however, vascular geometries have a natural winding structure that does not lend well to alignment in the global coordinate space. Thus, Hu et al. introduced a skeleton-based HBECVT to decompose tubular structures with an identifiable centerline or skeleton [181]. In section 5.3, a local coordinate system was defined at every point on the centerline. To allow the clustering to follow the curvature of

the centerlines, the surface normals are transformed to the closest centerline point's local coordinate system. Any unit vector, \hat{v} , in the global coordinate system, $\{\hat{e}_0, \hat{e}_1, \hat{e}_2\}$, can be transferred to the local coordinate system, $\{\hat{e}'_0, \hat{e}'_1, \hat{e}'_2\}$ through a transformation, $\hat{v}' = Q\hat{v}$, where $Q_{ij} = \cos(\hat{e}_i, \hat{e}'_j) = \hat{e}_i \cdot \hat{e}'_j$. Using matrix Q , every surface normal is converted to the local coordinate system to define a new normal, $\hat{n}'(T_j) = Q\hat{n}(T_j)$. The new normals are used for the clustering algorithm, in which the skeleton-based HBECVT energy is

$$E^S(C; U) = \sum_{j=0}^{n-1} \left[k / \left(\sum_{i=0}^{k-1} \|\hat{n}'(T_j) - c_i\|^2 + \eta \tilde{n}_i(T_j) \right) \right] \quad (5.18)$$

This produces a clustering that follows the parallel transport frame defined on the centerlines rather than the global surface normals. In situations where there are significant undulations in the surface, the clustering may be influenced too much by the cell's normals, and an unusable clustering can occur (Fig. 5.10). Thus, a positionally influenced, skeleton-based HBECVT is introduced as part of this work, which provides better results when tubular surfaces have significant undulations. At each triangle, T , the vector between the triangle centroid and the closest centerline point can be defined as $\vec{d}(T_j)$. The normalized position vector, \hat{d} is combined with the normal vector, \hat{n} , to provide a new vector

$$r(T_j) = \gamma \hat{n}(T_j) + (1 - \gamma) \hat{d}(T_j) \quad (5.19)$$

where γ is a positive factor between 0 and 1 that controls the influence of the triangle normal and position vector. The clustering energy now depends on the locally-transformed, normalized combination vector, \hat{r}' ,

$$E^P(C; U) = \sum_{j=0}^{n-1} \left[k / \left(\sum_{i=0}^{k-1} \|\hat{r}'(T_j) - c_i\|^2 + \eta \tilde{n}_i(T_j) \right) \right] \quad (5.20)$$

A high value of γ will put more influence on the normals of the triangles, while a low value of γ will put more influence on the position relative to the centerline (Fig. 5.10).

The clustering algorithm is performed on each group decomposing it into a set of patches that matches the group's polycube patches. Special care is taken to make sure that the patches will match at boundaries in between groups. Prior to the patch clustering, the boundaries in between groups, *ridge lines*, are traversed and *slice points* are defined to indicate the locations where group patches should meet (Fig. 5.11).

Based on these slice points, the surface normals at group boundaries are modified to force the patches to meet at these locations. The normals at the boundaries are set to one of the six clustering directions that matches the direction of that patch. After the patch clustering, the group boundaries are checked to verify that the patches interface correctly at each boundary. If they were not met correctly, then slight modifications can be made to the patch to correct for the inconsistency.

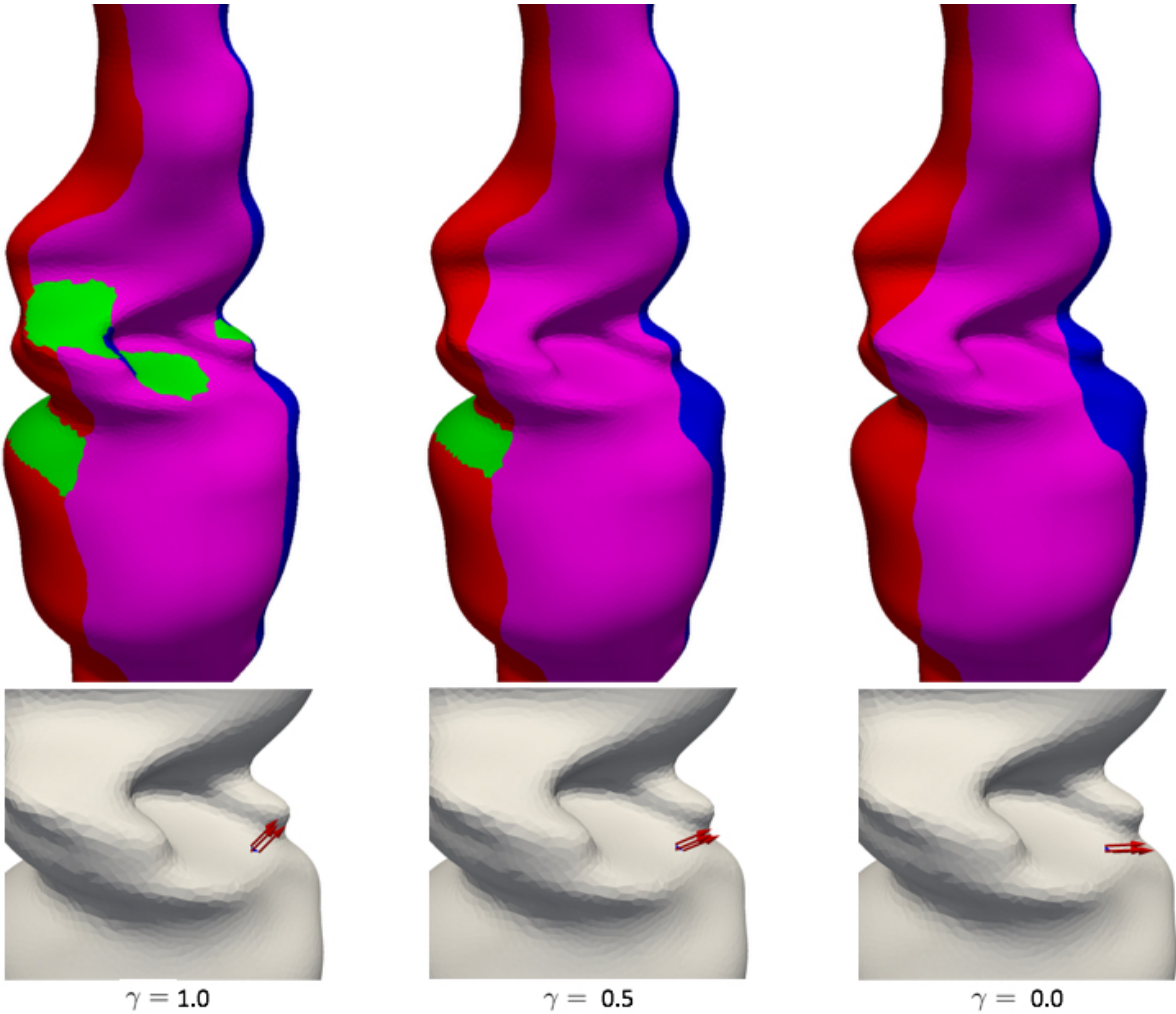


Figure 5.10: The clustering can be weighted more towards the normal of the cell or more towards the position relative to the centerline with factor γ .

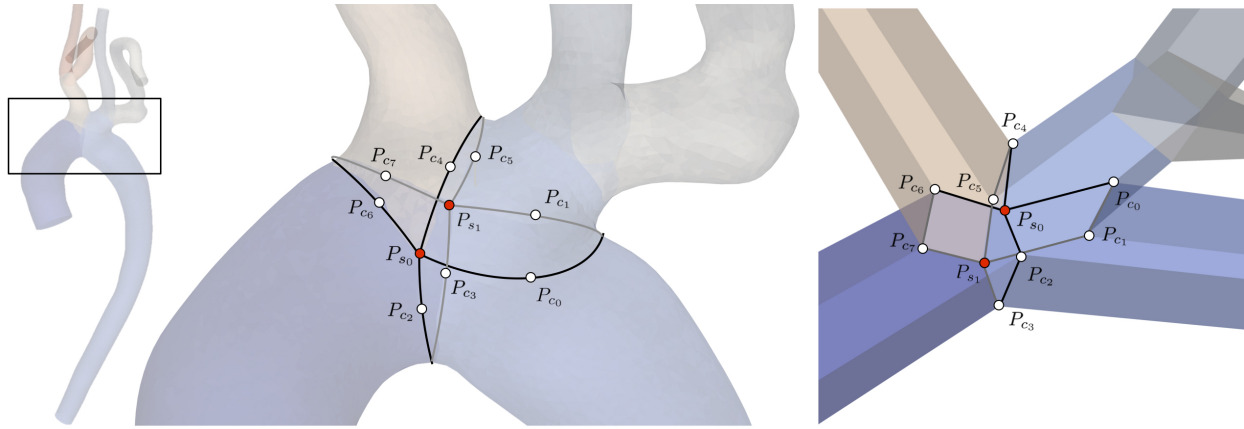


Figure 5.11: The surface groups are compared to the polycube. Lines in between surface groups are defined as *ridge lines*. The ridge lines are traversed to define matching *slice points*, P_{s_i} , and *corner points*, P_{c_i} , on the surface.

5.9 Conformal Mapping

Purpose:

The points of the surface need to be mapped to the structured polycube space in order to define a parameterization.

Methods:

With the triangulated surface mesh decomposed into patches, and the patch boundaries well defined, mapping of each patch can be performed. Each patch is conformally mapped to the planar base domain using a discrete conformal parameterization (DCP) [193]. A conformal mapping preserves angle magnitude and direction, which helps ensure that the final parameterization has minimal distortion. The conformal mapping is computed by minimizing the Dirichlet energy function




$$E(f) = \sum_{v_i, v_j \in S} w_{v_i, v_j} \|f(v_i) - f(v_j)\|^2, \quad (5.21)$$

where v_i and v_j are the set of vertices for all oriented edges on the surface. The weight, w_{v_i, v_j} , determines the energy to be minimized. With w_{v_i, v_j} equal to one at all vertices, the Tutte energy is minimized. The harmonic energy is minimized if the weights are equal to the harmonic edge weight

$$w_{v_i, v_j} = \frac{1}{2}(\cot(\alpha_{ij}) + \cot(\beta_{ij})), \quad (5.22)$$

where α_{ij} and β_{ij} are the angles opposite the edge in the two connected triangles as in Fig. 5.12. The energy is quadratic, and thus the derivation yields a linear system to be

Table 5.8: Inputs and Outputs for Conformal Mapping

Input	Definition	Data
Patched genus-zero triangulated surface model 	$S_p = (\{T_i\}_{i=0}^{n_t-1}; \{v_j\}_{j=0}^{n_v-1})$ n_t = number of surface triangles n_v = number of surface vertices	$GroupId(T)$ $PatchId(T)$
Volumetric Polycube 	$P_v = \{s_i\}_{i=0}^{n_s-1}$ n_s = number of structured grid volumes (i.e. number of branches)	$GroupId(s)$
Output	Definition	Data
Conformally mapped triangulated polycube surface model 	$S_c = (\{T_i\}_{i=0}^{n_t-1}; \{v_j\}_{j=0}^{n_v-1})$ n_t = number of surface triangles n_v = number of surface vertices	$GroupId(T)$ $PatchId(T)$

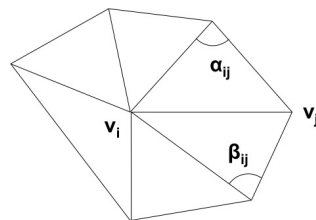


Figure 5.12: Angles α_{ij} and β_{ij} used for the harmonic edge weights.

solved

$$\Delta f(v_i) = \sum_{v_j \in \text{neigh}(v_i)} w_{v_i, v_j} (f(v_j) - f(v_i)) . \quad (5.23)$$

The conformal mapping is done in three steps. First, the matching polycube patch face is extracted from the polycube and rotated to the x-y plane and boundary vertices are placed on the plane according to chord length. A fixed border parameterization is used rather than a free border parameterization in order for the parameterizations to match up at patch borders

and form a globally valid parameterization. Second, the weights of the interior vertices are computed and inserted into the linear system

$$\begin{bmatrix} A_{interior} \\ I \end{bmatrix} \begin{bmatrix} \mathbf{v}_{interior} \\ \mathbf{v}_{boundary} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b} \end{bmatrix} \quad (5.24)$$

where

$$A_{interior_{ij}} = \begin{cases} w_{v_i, v_j} & \text{if } j \in n(v_i) \\ \sum_{v_k \in n(v_i)} -w_{v_i, v_k} & \text{if } i = j \\ 0 & \text{otherwise,} \end{cases} \quad (5.25)$$

where $n(v_i)$ refers to the neighborhood of vertices surrounding v_i , the weights, w_{v_i, v_j} , are again calculated according to the energy to be minimized, and the vertices, \mathbf{b} , are the fixed boundary locations on the plane. For an initial step, the edge weights of all the interior vertices are set to one, and the Tutte energy is minimized using conjugate gradients. This provides a better initial condition for the final conformal mapping, which uses the harmonic edge weights defined in Eq. (5.22). Therefore, the Tutte energy is first minimized and the result is used as an initial condition for the minimization of the harmonic energy. This creates a conformal map on the plane that can be used for texture mapping, parameterization, etc.

5.10 Volumetric Parameterization

Purpose:

A volumetric NURBS is defined on 3D structured grid, and that structured grid needs to be defined on the input geometry.

Methods:

The conformal maps are now used to obtain a parameterization for the NURBS surface, which can be extended to a volume parameterization by virtue of the polycube structure. The rectangular grid of vertices at the specified resolution are mapped back to the original triangulated surface mesh from the parameterization. Barycentric coordinates are used to place the rectangular grid of vertices on the original triangulated surface mesh. Then, the interior points are mapped using linear interpolation. The points are moved into the interior of the geometry using a hexahedral mesh laplacian smoothing algorithm. After both surface and interior points are mapped onto and into the original geometry, each group or branch is defined by its own structured grid or hexahedral mesh.

Table 5.9: Inputs and Outputs for Volumetric Parameterization







Input	Definition	Data
<p>Conformally mapped triangulated polycube surface model</p> 	$S_c = (\{T_i\}_{i=0}^{n_t-1}; \{v_j\}_{j=0}^{n_v-1})$ <p> n_t = number of surface triangles n_v = number of surface vertices </p>	<p> <i>GroupId</i> (T) <i>PatchId</i> (T) </p>
<p>Patched genus-zero triangulated surface model</p> 	$S_p = (\{T_i\}_{i=0}^{n_t-1}; \{v_j\}_{j=0}^{n_v-1})$ <p> n_t = number of surface triangles n_v = number of surface vertices </p>	<p> <i>GroupId</i> (T) <i>PatchId</i> (T) </p>
<p>Volumetric Polycube</p> 	$P_v = \{s_i\}_{i=0}^{n_s-1}$ <p> n_s = number of structured grid volumes (i.e. number of branches) </p>	<p><i>GroupId</i> (s)</p>
Output	Definition	Data
<p>Unstructured set of volumetric hex meshes</p> 	$H_c = \{s_i\}_{i=0}^{n_s-1}$ <p> n_s = number of structured grid volumes (i.e. number of branches) </p>	<p><i>GroupId</i> (s)</p>

Table 5.10: Inputs and Outputs for Volumetric NURBS Creation

Input	Definition	Data
Unstructured set of volumetric hex meshes 	$H_c = \{s_i\}_{i=0}^{n_s-1}$ $n_s = \text{number of structured grid volumes}$ (i.e. number of branches)	$GroupId (s)$
Output	Definition	Data
Volumetric NURBS 	$\mathbf{N} = \{\mathbf{V}_i(u, v, w)\}_{i=0}^{n_s-1}$ $n_s = \text{number of NURBS volumes}$ (i.e. number of branches)	$GroupId (\mathbf{V})$

5.11 Volumetric NURBS Creation

Purpose:

The goal is to obtain analysis suitable volumetric NURBS.

Methods:

The final volume is found using NURBS global interpolation techniques with uniform weights, average knot spacing, and chord length parameter spacing [84]. A NURBS volume is an extension of the CAD standard NURBS surface. With uniform weights, a NURBS surface is identical to a B-spline surface,

$$\mathbf{S}(u, v) = \sum_{i=0}^n \sum_{j=0}^m \mathbf{P}_{i,j} N_{i,p}(u) N_{j,q}(v), \quad (5.26)$$

where $\mathbf{S}(u, v)$ is the surface in two parameter directions, u and v , $\mathbf{P}_{i,j}$ are the control points and $N_{i,p}$ and $N_{j,q}$ are the the B-spline basis functions in the u and v parameter directions of p and q degrees respectively. The B-spline basis functions are defined accordingly,

$$N_{i,0} = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise.} \end{cases} \quad (5.27)$$

$$N_{i,p} = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u).$$

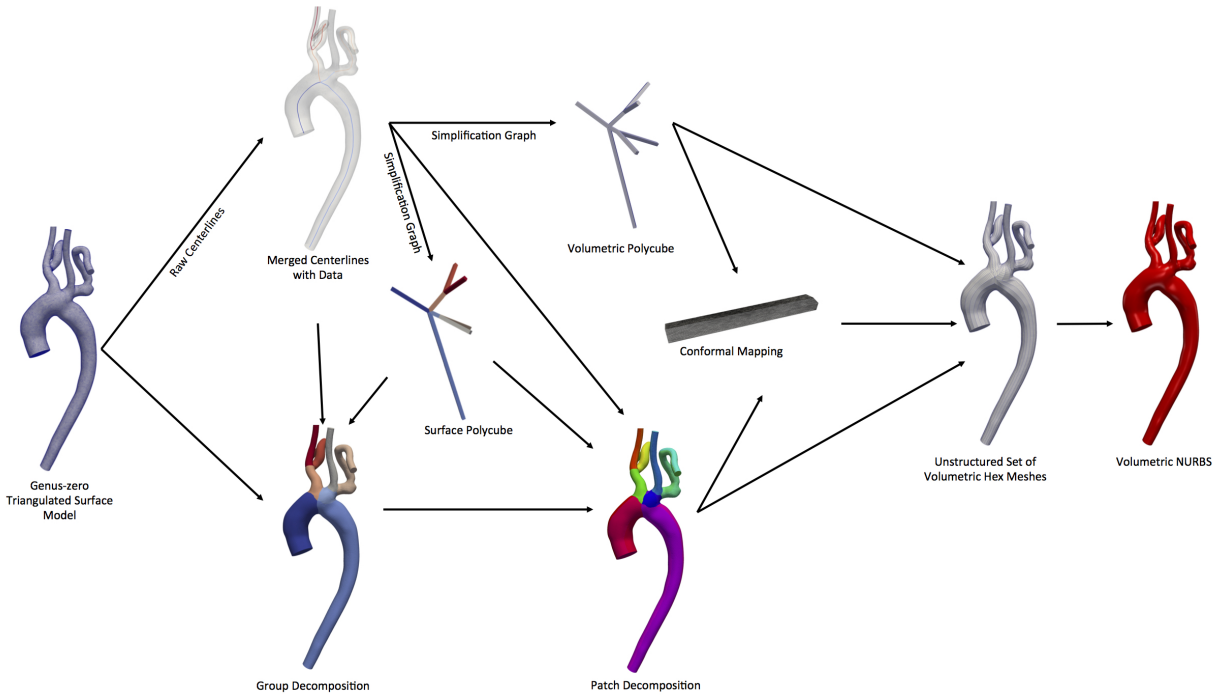


Figure 5.13: The creation and involvement of different objects in the formation of a volumetric NURBS from a genus-zero triangulated surface model.

A NURBS volume with uniform weights then has a very similar definition with an additional parametric direction,

$$\mathbf{V}(u, v, w) = \sum_{i=0}^n \sum_{j=0}^m \sum_{k=0}^o \mathbf{P}_{i,j,k} N_{i,p}(u) N_{j,q}(v) N_{k,r}(w) \quad (5.28)$$

where $\mathbf{P}_{i,j,k}$ is now the structured grid of control points in three parametric directions, u , v , and w , and $N_{k,r}$ are the new B-spline basis functions in the w parameter direction of degree r .

5.12 Results

The methods developed succeed in converting arbitrary complex triangulated surfaces of vasculature to volumetric NURBS that are suitable for IGA. In addition, the framework produces a volumetric, hexahedral mesh that is usable in FEA. Fig. 5.14 displays the final NURBS parameterization (right) of the example patient-specific triangulated surface (left).

Qualitatively, the results are satisfying and the resulting representation matches the original surface very well. Average distance, Hausdorff distance [194, 195] and the Dice coefficient [196] are used to quantitatively evaluate the accuracy of the final NURBS geometry.

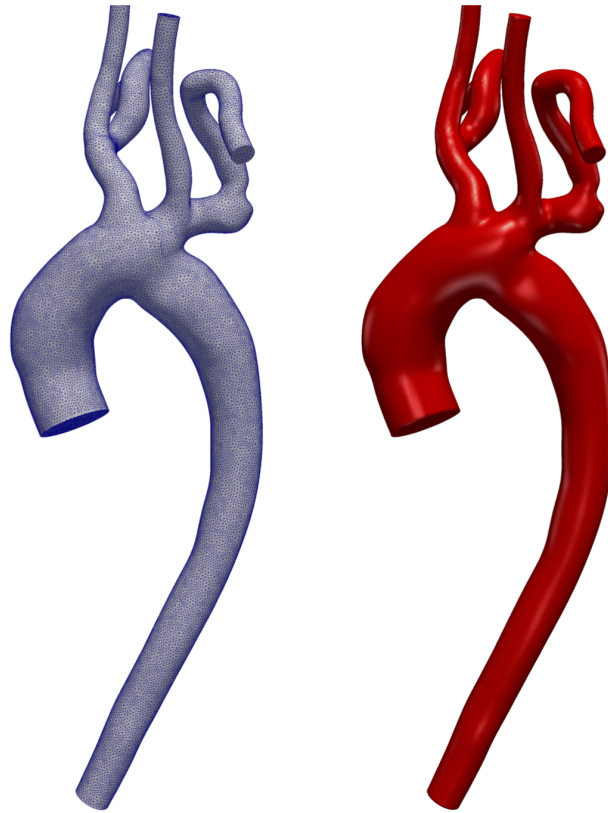


Figure 5.14: Comparison of original triangulated surface mesh (left) with the surface NURBS representation (right).

To calculate the Dice and Jaccard coefficients, the input surface and a very high resolution model of the output NURBS surface were converted to binary images at varying resolutions. The Dice and Jaccard coefficients were computed on binary images of increasingly higher resolution until the coefficients converged. All metrics are displayed for the model of an Aorta and results are displayed in Table 5.11. All the metrics indicate that the input triangulated surface and the final representation are similar with a maximum error of 0.012 553.

To help put these distance measures in perspective, the resolution of the medical image data for the aorta and other large arteries is typically around 0.1 *cm*. The average error is around two orders of magnitude smaller than the image-resolution, and hence original model fidelity. Thus, the conversion preserves the geometric detail of the model, while providing a definition more suitable for engineering design. It is possible to vary the resolution of the structured grid defined on the polycube that is mapped to the surface. In these cases, a structured grid of lower resolution than the input surface was used to demonstrate the efficiency of using a NURBS definition. For example, in the aorta model, the input surface contained 94,740 triangles and 47,372 vertices with an average edge size of 0.1 *cm*, and the NURBS surface was populated with a total of 10,002 control points. The error between the

Aorta Model			
Avg. Dist. (<i>cm</i>)	Haus. Dist. (<i>cm</i>)	Dice	Jaccard
0.000 734	0.012 553	0.983662	0.967849

Table 5.11: The average distance, Hausdorff distance, Dice metric and Jaccard coefficient between the NURBS representation and the original input surface mesh.

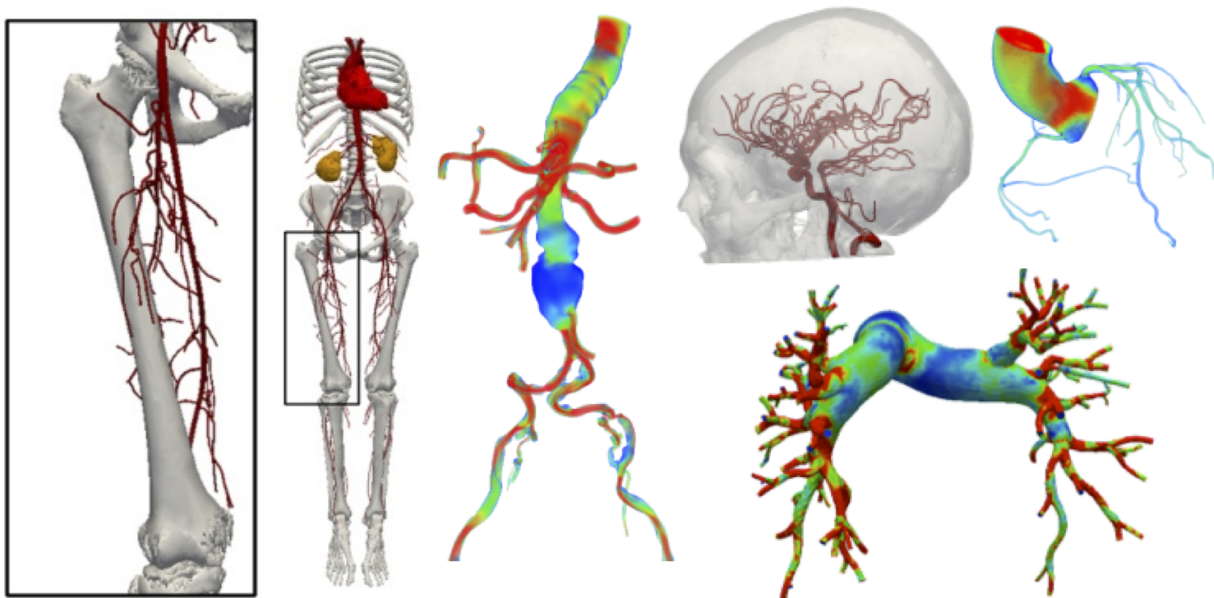


Figure 5.15: A sampling of the wide variety of model categories and simulation results available online in the vascular model repository at www.vascularmodel.com.

output NURBS and the input mesh representation could be further decreased in many ways. The input triangulated surface could be refined to a smaller mesh edge size. In addition, the resolution of the control points could be decreased or redistributed to minimize an error field between the input surface and the NURBS representation. This leads to a tradeoff between resolution/complexity of the NURBS parameterization and the amount of error incurred by the conversion. While the error for any given application will depend on the chosen NURBS grid resolution and model morphology, we have shown for a typical example that a grid size of reduced complexity as compared to the input surface achieves an error below the resolution of the image data and preserves the geometric detail of the model.

To further demonstrate the usability and robustness of this framework, the conversion

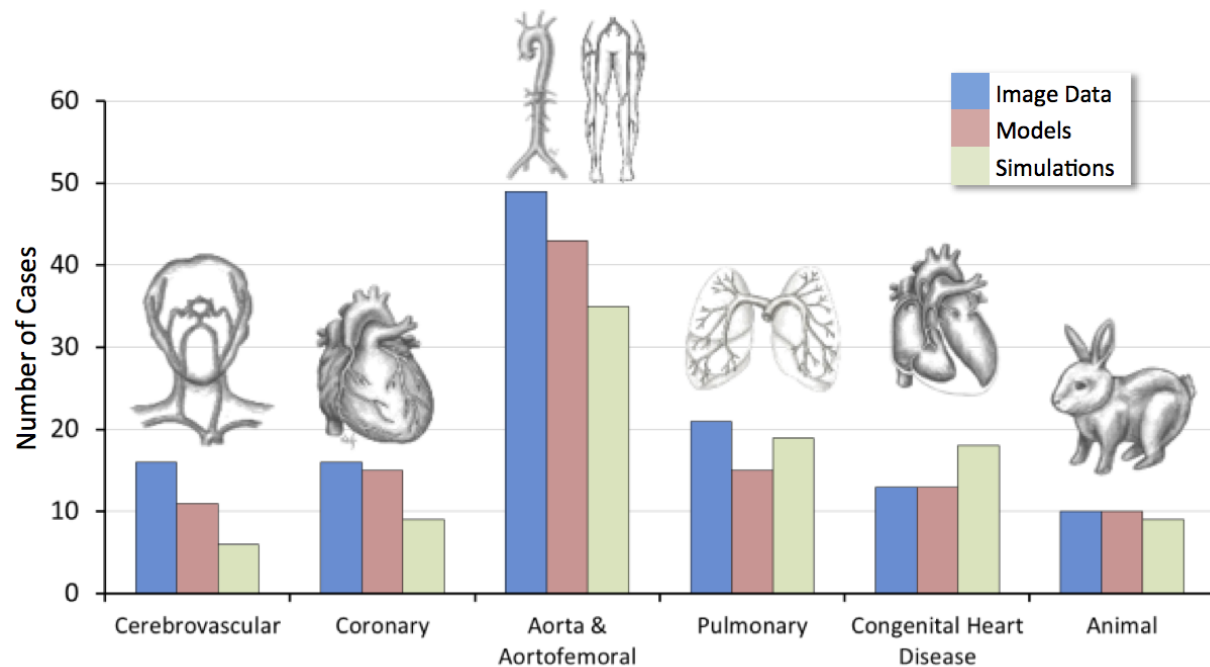


Figure 5.16: The vascular model repository combines the results of over 100 studies of varying image data, model complexity, and simulation type.

was attempted on the 127 complex vascular geometries in the Open Source Medical Software Corporation (OSMSC) repository of (Fig. 5.15). Figure 5.16 characterizes the breakdown of this repository.

The current set of tools do not allow non genus-zero geometries or geometries with too large of a size-scale difference between adjacent vessels. Specifically, the cutoff for these algorithms has been placed to return an error if a vessel is 5 times larger or smaller by radius than an adjacent vessel. This is most often experienced in coronary models in which both the aorta and the coronary are modeled. There are 22 non genus-zero geometries in the repository and 24 geometries with too large of a size-scale difference between adjacent vessels, resulting in a total of 81 complex vascular geometries that are suitable for these conversion tools. Triangulated surfaces of the 81 geometries were obtained through the open-source pipeline in the new SimVascular 3.0 graphical user interface [29]. Segmentations in the OSMSC repository of the 81 geometries were lofted using the lofted 2D segmentation approach, triangulated, and remeshed to a *coarse* edge size. In this context, a *coarse* edge size refers to a mesh edge size in which there are approximately 12-15 triangles around the exterior circumferentially of the smallest vessel of the model. For the pulmonary model in Fig. 5.17, this results in a triangulated surface of 170,000 triangles, 150,000 triangles for the aortofemoral model in Fig. 5.18, and 110,000 triangles for the cerebrovascular model in Fig. 5.19. It is important to note that the methods are more robust for finer triangulations;

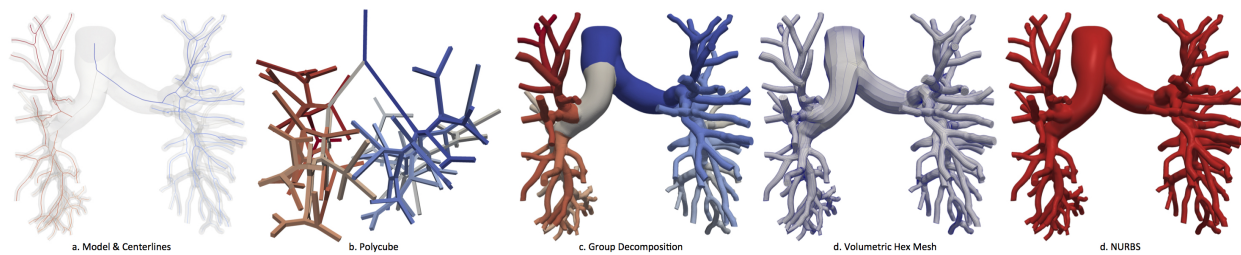


Figure 5.17: The pipeline for patient-specific volumetric NURBS creation on a pulmonary model.

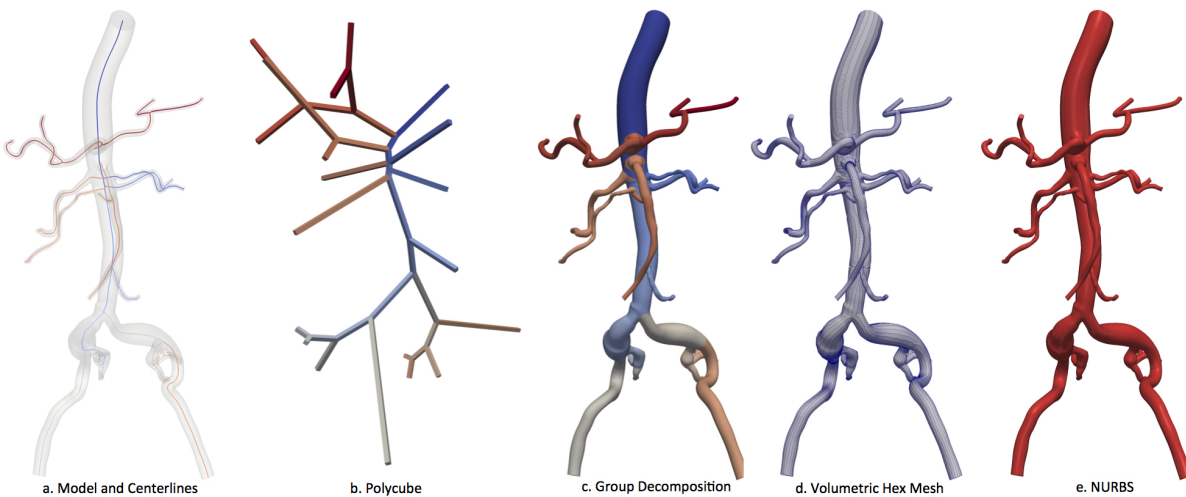


Figure 5.18: The pipeline for patient-specific volumetric NURBS creation on an aortofemoral model.

however, this increases computation time.

Of the 81 geometries, 76 succeed in automatic conversion (94%) using the default parameters and a *coarse* model representation. The 5 remaining geometries require a finer mesh, changes to the parameters for the centerline merge radius (Sec. 5.3), and/or changes to the clustering vector factor (Sec. 5.8). It is also important to note that the models in the open-source model repository represent a very complex set of models. Many vascular models are of a smaller, more specific region, such as a cerebral aneurysm, carotid bifurcation, or abdominal aortic aneurysm (AAA).

Lastly, because these methods mainly rely on the geometry containing a 1D centerline structure, the algorithms are also applicable to a subset of non-vascular geometries. Figs. 5.20-5.23 demonstrate the use of these automatic methods on a variety of non-vascular geometries in the mesh segmentation benchmark dataset [197].

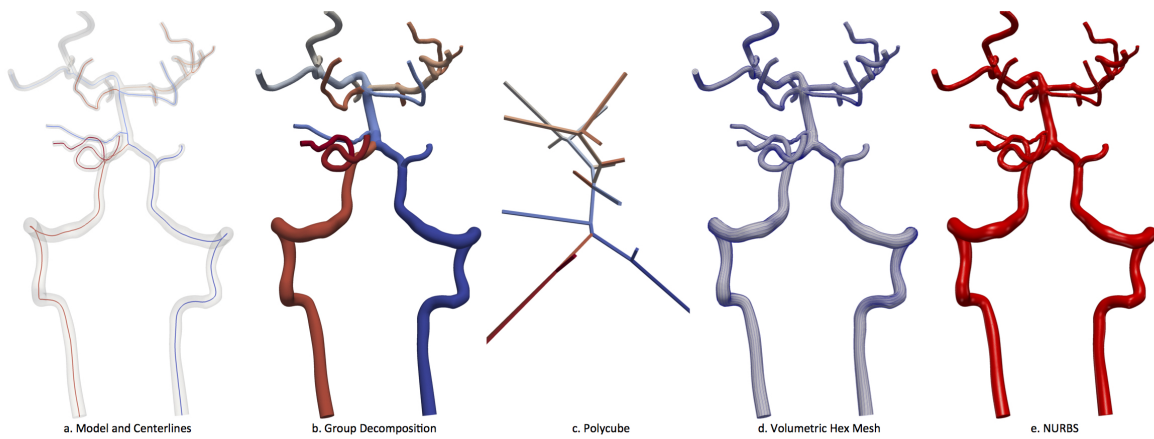


Figure 5.19: The pipeline for patient-specific volumetric NURBS creation on a cerebrovascular model.

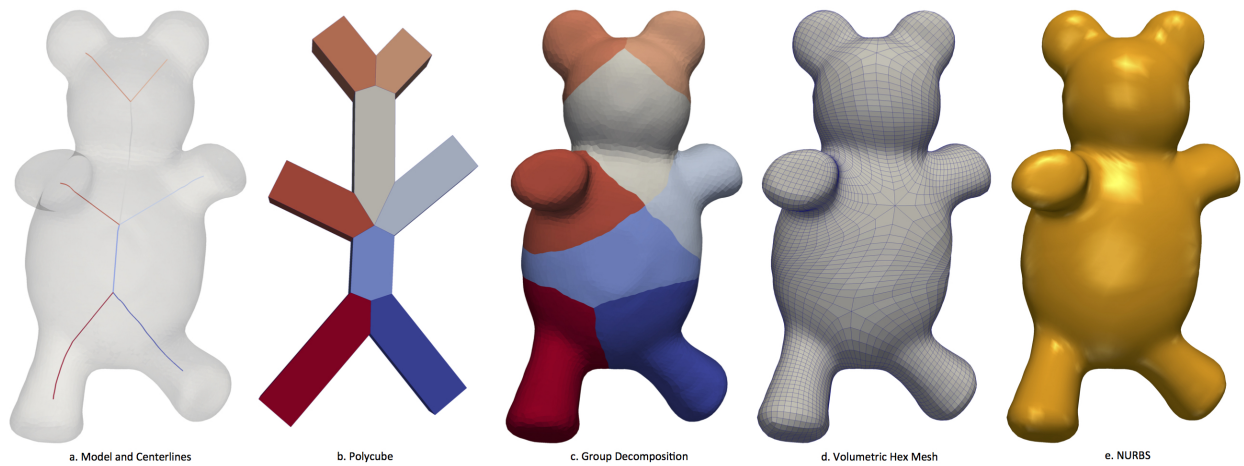


Figure 5.20: The pipeline for NURBS creation of the teddy bear model.

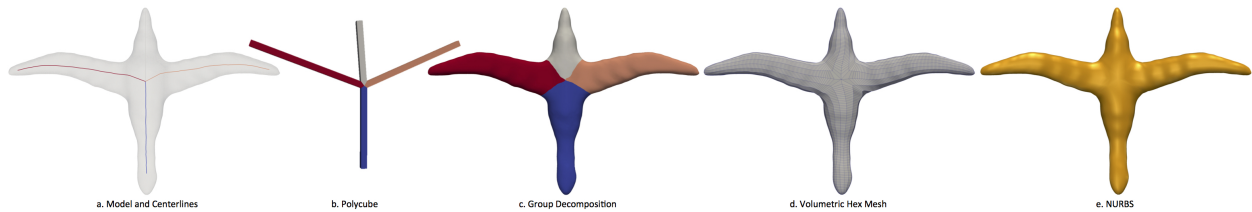


Figure 5.21: The pipeline for NURBS creation of the bird model.

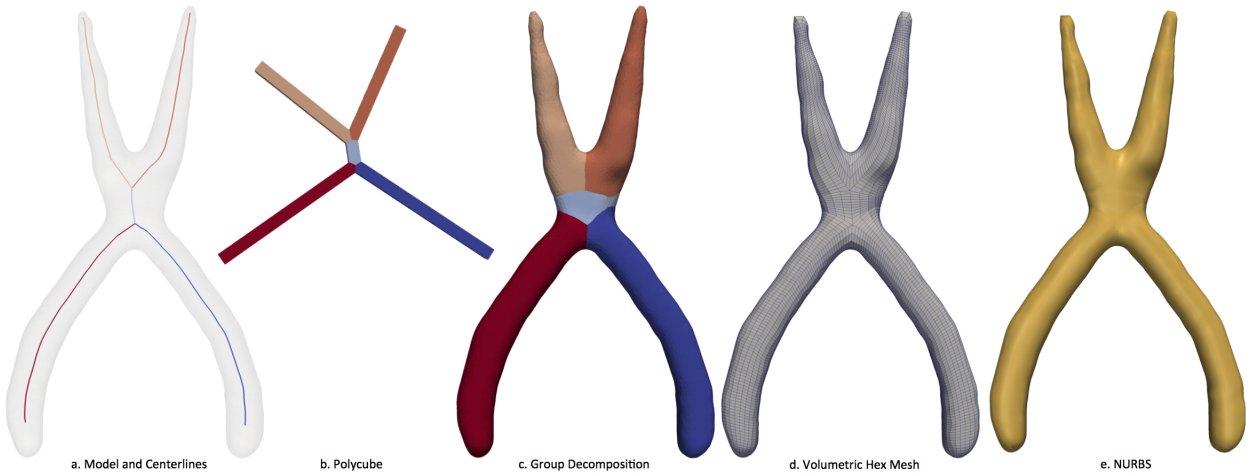


Figure 5.22: The pipeline for NURBS creation of the pliers model.

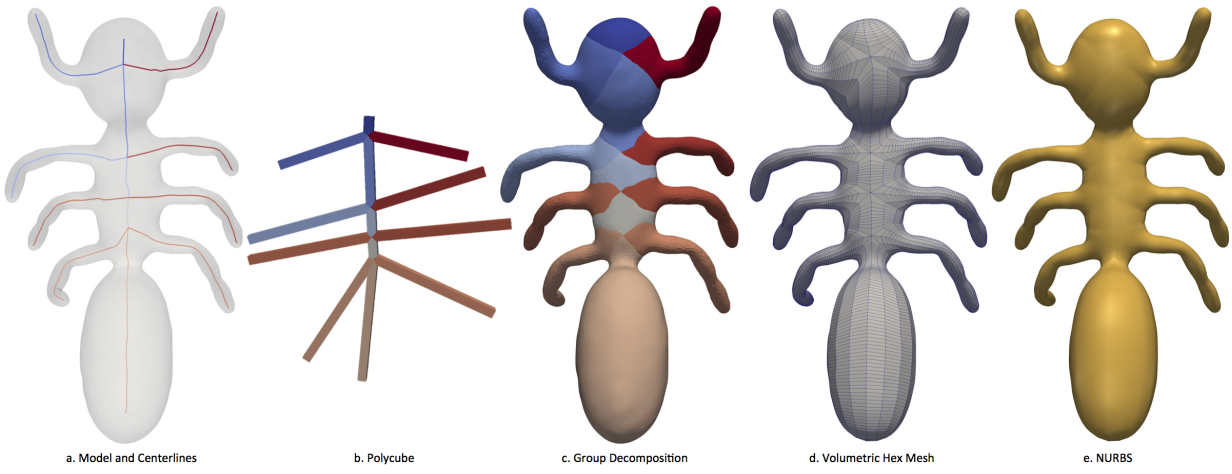


Figure 5.23: The pipeline for NURBS creation of the ant model.

5.13 Discussion

This chapter presents a novel framework to create analysis suitable volumetric NURBS representations of complex vascular geometries that are suitable for fluids computation in an IGA framework. The methods are promising; they result in a representation that is highly accurate (average error 0.001 cm). Additionally, the methods have been tested on 81 complex vascular models and 94% of the models achieve successful conversion with zero manual intervention, while the remaining 5 models require minimal manual intervention. There are drawbacks to the current methods and future work can improve them greatly. First, the framework does not allow for non-genus zero geometries which can be present in a variety of vascular geometries. For example, a complete circle of willis in the cerebral vasculature, an aortic dissection, and a model of an artery with a bypass graft all have higher genus. Future work would include modifying the centerline processing and polycube construction algorithms to handle non-genus zero geometries. In addition, vasculature with very large changes in size scale between adjacent vessels is an issue. Future work would require modifying the polycube structure where large size-scale differences occur, allowing smaller vessels to additionally have smaller representations on the polycube structure. It is important to note that the algorithms presented have been implemented in a very modular fashion. The breakdown within the methods is also present in the code, thus, it should be very easy to modify and improve portions of the pipeline. For example, the current branch clustering algorithm described in Sec. 5.7 simply uses a modified distance function to compute the closest centerline point for each triangle on the surface and apply a corresponding label to that triangle. There are variety of other decomposition and clustering algorithms [198, 199, 197] that could be ported into the pipeline and used. Another potential improvement to the pipeline would be including methods to smooth the parameterization after it is computed. Distortions in the parameterization can lead to poor quality NURBS representations; thus, performing a smoothing of the parameter space after mapping to the original input surface would improve the results. Lastly, these tools are available in an open source github repository called *vtkSV*, and specific functionalities will be added to the graphical user interface of *SimVascular*.

Chapter 6

Summary

The major contributions of this body of work include the development of a revised SimVascular cardiovascular modeling software and new methods for the conversion of discrete vascular geometries to analysis suitable representations. As open-source tools, the accurate and efficient techniques developed in this dissertation will help promote vascular modeling and simulation for academic research and use in the clinic. The algorithms and methods developed include a variety of tools for open-source modeling including Boolean techniques and constrained smoothing methods for triangulated surfaces. Open-source meshing tools were also implemented and include a range of options such as boundary layer meshing, radius-based meshing, and adaptive mesh refinement. Finally, novel techniques to convert discrete vascular geometries to analysis suitable representations were developed. It is also important to note that these tools have been created specifically for a blood flow simulation pipeline; however, many of the tools, such as the Boolean techniques and discrete surface manipulation operations, are generally useful geometric tools and can easily be applied to other disciplines. For example, the tools in SimVascular have been used in simulation particle deposition in the human lungs [57] and creation of structural lattices of cell growth [200]. Finally, to help emphasize the global impact of this work, a plot of the locations that have visited the SimVascular webpage in the last 180 days is displayed in Fig. 6.1. In addition, since the open-source release in 2014, SimVascular has garnered over 2,500 users and the software has been downloaded over 9,000 times.

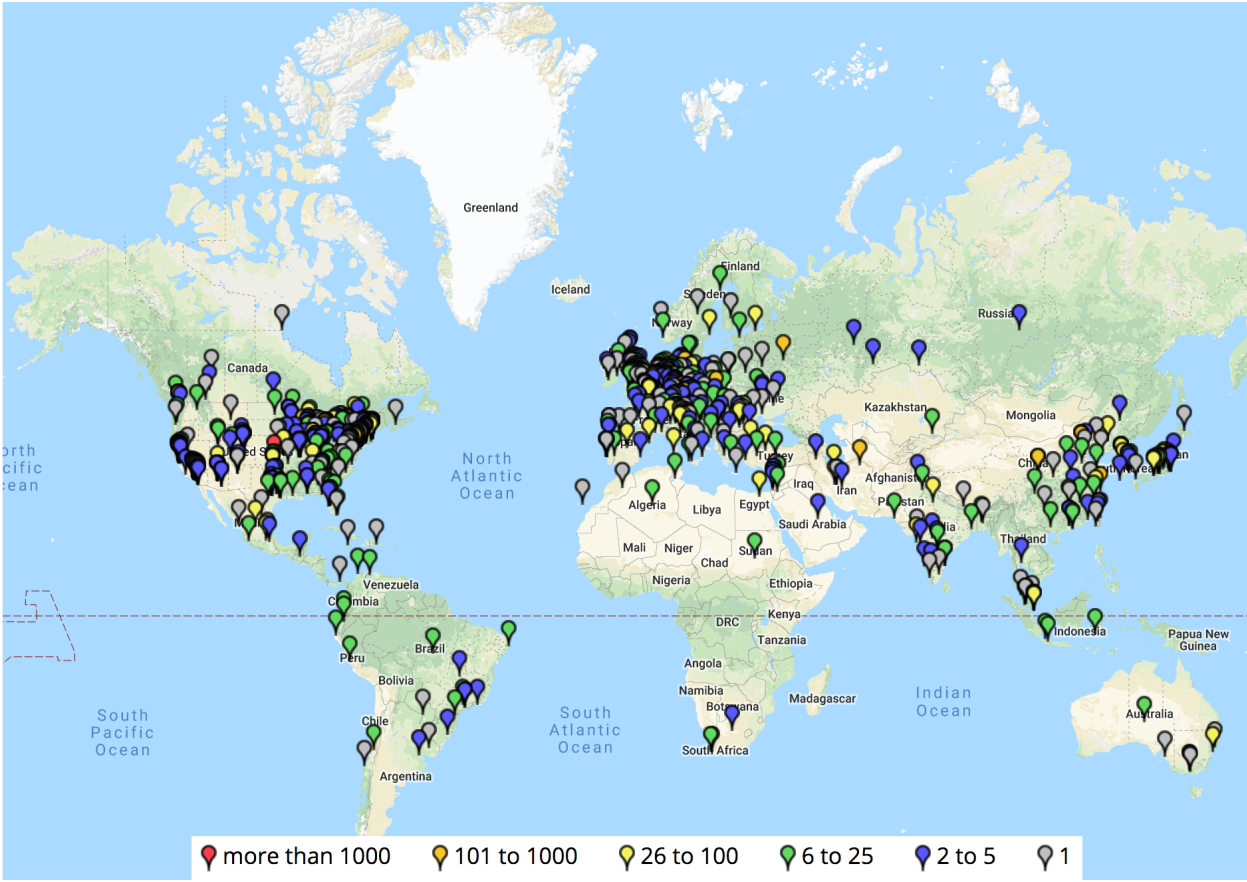


Figure 6.1: A display of SimVascular webpage hits in the last 180 days.

Bibliography

- [1] World Health Organization. *World Health Statistics 2016: Monitoring Health for the SDGs Sustainable Development Goals*. World Health Organization, 2016.
- [2] E. J. Benjamin, M. J. Blaha, S. E. Chiuve, M. Cushman, S. R. Das, R. Deo, S. D. de Ferranti, J. Floyd, M. Fornage, C. Gillespie, et al. Heart disease and stroke statistics—2017 update: a report from the american heart association. *Circulation*, 135(10):e146–e603, 2017.
- [3] C. A. Taylor, T. J.R. Hughes, and C. K. Zarins. Computational investigations in vascular disease. *Computers in Physics*, 10(3):224–232, 1996.
- [4] C. A. Taylor, M. T. Draney, J. P. Ku, D. Parker, B. N. Steele, K. Wang, and C. K. Zarins. Predictive medicine: computational techniques in therapeutic decision-making. *Computer Aided Surgery*, 4(5):231–247, 1999.
- [5] C. A. Taylor, T. A. Fonte, and J. K. Min. Computational fluid dynamics applied to cardiac computed tomography for noninvasive quantification of fractional flow reserve: scientific basis. *Journal of the American College of Cardiology*, 61(22):2233–2241, 2013.
- [6] A. Lonyai, A. M. Dubin, J. A. Feinstein, C. A. Taylor, and S. C. Shadden. New insights into pacemaker lead-induced venous occlusion: Simulation-based investigation of alterations in venous biomechanics. *Cardiovascular Engineering*, 10(2):84–90, 2010.
- [7] A. L. Marsden, V. M. Reddy, S. C. Shadden, F. P. Chan, C. A. Taylor, and J. A. Feinstein. A new multiparameter approach to computational simulation for fontan assessment and redesign. *Congenital Heart Disease*, 5(2):104–117, 2010.
- [8] A. Arzani, P. Dyverfeldt, T. Ebberts, and S. C. Shadden. In vivo validation of numerical prediction for turbulence intensity in an aortic coarctation. *Annals of Biomedical Engineering*, 40(4):860–870, 2012.
- [9] S. Sankaran, M. Esmaily-Moghadam, A. M. Kahn, J. Guccione, E. Tseng, and A. L. Marsden. Patient-specific multiscale modeling of blood flow for coronary artery bypass graft surgery. *Annals of Biomedical Engineering*, 40(1):2228–2242, 2012.

- [10] M. D. Bockman, A. P. Kansagra, S. C. Shadden, E. C. Wong, and A. L. Marsden. Fluid mechanics of mixing in the vertebrobasilar system: Comparison of simulation and MRI. *Cardiovascular Engineering and Technology*, 3(4):450–461, 2012.
- [11] I. A. Carr, N. Nemoto, R. S. Schwartz, and S. C. Shadden. Size-dependent predilections of cardiogenic embolic transport. *American Journal of Physiology-Heart and Circulatory Physiology*, 305(5):H732–H739, 2013.
- [12] A. Arzani, G. Y. Suh, R. L. Dalman, and S. C. Shadden. A longitudinal comparison of hemodynamics and intraluminal thrombus deposition in abdominal aortic aneurysms. *American Journal of Physiology- Heart and Circulatory Physiology*, 307(12):H1786–H1795, 2014.
- [13] M. Esmaily-Moghadam, T.-Y. Hsia, and A. L. Marsden. The Assisted Bidirectional Glenn: a novel surgical approach for first stage single ventricle heart palliation. *Journal of Thoracic and Cardiovascular Surgery*, 149(3):699–705, 2015.
- [14] D. Mukherjee, J. Padilla, and S. C. Shadden. Numerical investigation of fluid–particle interactions for embolic stroke. *Theoretical and Computational Fluid Dynamics*, pages 1–17, 2015.
- [15] A. Wittek, N. M. Grosland, G. R. Joldes, V. Magnotta, and K. Miller. From finite element meshes to clouds of points: a review of methods for generation of computational biomechanics models for patient-specific applications. *Annals of biomedical engineering*, 44(1):3–15, 2016.
- [16] A. Updegrave, N. M. Wilson, J. Merkow, H. Lan, A. L. Marsden, and S. C. Shadden. Simvascular: An open source pipeline for cardiovascular simulation. *Annals of Biomedical Engineering*, 45(3):525–541, 2017.
- [17] *Using a Science Gateway to Deliver SimVascular Software as a Service for Classroom Instruction*, 2018.
- [18] W. Nichols, M. O’Rourke, and C. Vlachopoulos. *McDonald’s blood flow in arteries: theoretical, experimental and clinical principles*. CRC Press, 2011.
- [19] A. M. Malek, S. L. Alper, and S. Izumo. Hemodynamic shear stress and its role in atherosclerosis. *Jama*, 282(21):2035–2042, 1999.
- [20] J. J. Wentzel, E. Janssen, J. Vos, J. C.H. Schuurbiens, R. Krams, P. W. Serruys, P. J. de Feyter, and C. J. Slager. Extension of increased atherosclerotic wall thickness into high shear stress regions is associated with loss of compensatory remodeling. *Circulation*, 108(1):17–23, 2003.

- [21] P. F. Davies, M. Civelek, Y. Fang, and I. Fleming. The atherosusceptible endothelium: endothelial phenotypes in complex haemodynamic shear stress regions in vivo. *Cardiovascular research*, page cvt101, 2013.
- [22] B. R. Kwak, M. Bäck, M.-L. Bochaton-Piallat, G. Caligiuri, M. J.A.P. Daemen, P. F. Davies, I. E. Hofer, P. Holvoet, H. Jo, R. Krams, et al. Biomechanical factors in atherosclerosis: mechanisms and clinical implications. *European heart journal*, page ehu353, 2014.
- [23] U. Morbiducci, A. M. Kok, B. R. Kwak, P. H. Stone, D. A. Steinman, J. J. Wentzel, et al. Atherosclerosis at arterial bifurcations: evidence for the role of haemodynamics and geometry. *Thrombosis and haemostasis*, 115(3):484–492, 2016.
- [24] R. Krams, J. J. Wentzel, J. A.F. Oomen, R. Vinke, J. C.H. Schuurbiens, P. J. De Feyter, P. W. Serruys, and C. J. Slager. Evaluation of endothelial shear stress and 3D geometry as factors determining the development of atherosclerosis and remodeling in human coronary arteries in vivo Combining 3D reconstruction from angiography and IVUS (ANGUS) with computational fluid dynamics. *Arteriosclerosis, Thrombosis, and Vascular Biology*, 17(10):2061–2065, 1997.
- [25] J. S. Milner, J. A. Moore, B. K. Rutt, and D. A. Steinman. Hemodynamics of human carotid artery bifurcations: computational studies with models reconstructed from magnetic resonance imaging of normal subjects. *Journal of Vascular Surgery*, 28(1):143–156, 1998.
- [26] K. Perktold, M. Hofer, G. Karner, W. Trubel, and H. Schima. Computer simulation of vascular fluid dynamics and mass transport: optimal design of arterial bypass anastomoses. In *Proceedings of ECCOMAS*, volume 98, pages 484–489, 1998.
- [27] J. A. Moore, D. A. Steinman, D. W. Holdsworth, and C. R. Ethier. Accuracy of computational hemodynamics in complex arterial geometries reconstructed from magnetic resonance imaging. *Annals of Biomedical Engineering*, 27(1):32–41, 1999.
- [28] C. A. Taylor, T. A. Fonte, and J. K. Min. Computational fluid dynamics applied to cardiac computed tomography for noninvasive quantification of fractional flow reserve: scientific basis. *Journal of the American College of Cardiology*, 61(22):2233–2241, 2013.
- [29] H. Lan, A. Updegrove, N. M. Wilson, G. D. Maher, S. C. Shadden, and A. L. Marsden. A re-engineered software interface and workflow for the open-source simvascular cardiovascular modeling package. *Journal of biomechanical engineering*, 140(2):024501, 2018.
- [30] A. S. Les, S. C. Shadden, C. A. Figueroa, J. M. Park, M. M. Tedesco, R. J. Herfkens, R. L. Dalman, and C. A. Taylor. Quantification of Hemodynamics in Abdominal Aortic Aneurysms During Rest and Exercise Using Magnetic Resonance Imaging and

- Computational Fluid Dynamics. *Annals of Biomedical Engineering*, 38(4):1288–1313, April 2010.
- [31] J. C. Bezdek, L. O. Hall, and L. P. Clarke. Review of MR image segmentation techniques using pattern recognition. *Medical Physics*, 20(4):1033–1048, 1992.
- [32] K. C.Y. Wang. *Level set methods for computational prototyping with application to hemodynamic modeling*. PhD thesis, Stanford University, 2001.
- [33] L. M. Lorigo, O. D. Faugeras, W. E. L. Grimson, R. Keriven, R. Kikinis, A. Nabavi, and C.-F. Westin. Curves: Curve evolution for vessel segmentation. *Medical Image Analysis*, 5(3):195–206, 2001.
- [34] C. Li, R. Huang, Z. Ding, J. C. Gatenby, D. N. Metaxas, and J. C. Gore. A level set method for image segmentation in the presence of intensity inhomogeneities with application to mri. *IEEE Transactions on Image Processing*, 20(7):2007–2016, 2011.
- [35] A. Updegrave, N. M. Wilson, and S. C. Shadden. Boolean and smoothing of discrete polygonal surfaces. *Advances in Engineering Software*, 95:16–27, 2016.
- [36] O. Sahni, J. Müller, K. E. Jansen, M. S. Shephard, and C. A. Taylor. Efficient anisotropic adaptive discretization of the cardiovascular system. *Computer Methods in Applied Mechanics and Engineering*, 195(41-43):5634–5655, August 2006.
- [37] N. M. Wilson, F. R. Arko, and C. A. Taylor. Predicting changes in blood flow in patient-specific operative plans for treating aortoiliac occlusive disease. *Computer Aided Surgery*, 10(4):257–277, 2005.
- [38] H. J. Kim, I. E. Vignon-Clementel, J. S. Coogan, C. A. Figueroa, K. E. Jansen, and C.A. Taylor. Patient-specific modeling of blood flow and pressure in human coronary arteries. *Annals of Biomedical Engineering*, 38(10):3195–3209, 2010.
- [39] I. E. Vignon-Clementel, C. A. Figueroa, K. E. Jansen, and C. A. Taylor. Outflow boundary conditions for three-dimensional finite element modeling of blood flow and pressure in arteries. *Computer Methods in Applied Mechanics and Engineering*, 195(29-32):3776 – 3796, 2006.
- [40] M. Esmaily-Moghadam, I. E. Vignon-Clementel, R. Figliola, and A. L. Marsden. A modular numerical method for implicit 0D/3D coupling in cardiovascular finite element simulations. *Journal of Computational Physics*, 244:63–79, 2013.
- [41] A. L. Marsden and M. Esmaily-Moghadam. Multiscale modeling of cardiovascular flows for clinical decision support. *Applied Mechanics Reviews*, 67(3):030804, 2015.

- [42] D. E. Schiavazzi, E. O. Kung, A. L. Marsden, C. Baker, G. Pennati, T.-Y. Hsia, A. Hlavacek, A. L. Dorfman, Modeling of Congenital Hearts Alliance (MOCHA) Investigators, et al. Hemodynamic effects of left pulmonary artery stenosis after superior cavopulmonary connection: a patient-specific multiscale modeling study. *The Journal of thoracic and cardiovascular surgery*, 149(3):689–696, 2015.
- [43] E. O. Kung, A. Baretta, C. Baker, G. Arbia, G. Biglino, C. Corsini, S. Schievano, I. E. Vignon-Clementel, G. Dubini, G. Pennati, et al. Predictive modeling of the virtual Hemi-Fontan operation for second stage single ventricle palliation: two patient-specific cases. *Journal of Biomechanics*, 46(2):423–429, 2013.
- [44] C. A. Figueroa, I. E. Vignon-Clementel, K. E. Jansen, T. J.R. Hughes, and C. A. Taylor. A coupled momentum method for modeling blood flow in three-dimensional deformable arteries. *Computer methods in applied mechanics and engineering*, 195(41):5685–5706, 2006.
- [45] C. H. Whiting and K. E. Jansen. A stabilized finite element method for the incompressible Navier-Stokes equations using a hierarchical basis. *International Journal for Numerical Methods in Fluids*, 35(1):93–116, 2001.
- [46] L. P. Franca and S. L. Frey. Stabilized finite element methods: Ii. the incompressible navier-stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 99(2-3):209–233, 1992.
- [47] C. A Taylor, T. J.R. Hughes, and C. K. Zarins. Finite element modeling of blood flow in arteries. *Computer methods in applied mechanics and engineering*, 158(1):155–196, 1998.
- [48] M. Esmaily-Moghadam, Y. Bazilevs, T.-Y. Hsia, I. E. Vignon-Clementel, and A. L. Marsden. A comparison of outlet boundary treatments for prevention of backflow divergence with relevance to blood flow simulations. *Computational Mechanics*, 48:277–291, 2011.
- [49] M. Esmaily-Moghadam, Y. Bazilevs, and A. L. Marsden. A new preconditioning technique for implicitly coupled multidomain simulations with applications to hemodynamics. *Computational Mechanics*, 52:1141–1152, 2013.
- [50] M. Zhou, O. Sahni, H. J. Kim, C. A. Figueroa, C. A. Taylor, M. S. Shephard, and K. E. Jansen. Cardiovascular flow simulation at extreme scale. *Computational Mechanics*, 46(1):71–82, 2010.
- [51] S. C. Shadden and C. A. Taylor. Characterization of coherent structures in the cardiovascular system. *Annals of Biomedical Engineering*, 36:1152–1162, 2008.

- [52] A. Arzani and S. C. Shadden. Characterization of the transport topology in patient-specific abdominal aortic aneurysm models. *Physics of Fluids (1994-present)*, 24(8):081901, 2012.
- [53] M. Astorino, J. Hamers, S. C. Shadden, and J. Gerbeau. A robust and efficient valve model based on resistive immersed surfaces. *International Journal for Numerical Methods in Biomedical Engineering*, 28(9):937–959, 2012.
- [54] A. Arzani, A. S. Les, R. L. Dalman, and S. C. Shadden. Effect of exercise on patient specific abdominal aortic aneurysm flow topology and mixing. *International Journal for Numerical Methods in Biomedical Engineering*, 30(2):280–295, 2014.
- [55] G. Y. Suh, A. S. Les, A. S. Tenforde, S. C. Shadden, R. L. Spilker, J. J. Yeung, C. P. Cheng, R. J. Herfkens, R. L. Dalman, and C. A. Taylor. Quantification of particle residence time in abdominal aortic aneurysms using magnetic resonance imaging and computational fluid dynamics. *Annals of Biomedical Engineering*, 39:864–883, 2011.
- [56] G. Y. Suh, A. S. Tenforde, S. C. Shadden, R. L. Spilker, C. P. Cheng, R. J. Herfkens, R. L. Dalman, and C. A. Taylor. Hemodynamic changes in abdominal aortic aneurysms with increasing exercise intensity using MR exercise imaging and image-based computational fluid dynamics. *Annals of Biomedical Engineering*, 39:2186–2202, 2011.
- [57] J. M. Oakes, A. L. Marsden, C. Grandmont, S. C. Shadden, C. Darquenne, and I. E. Vignon-Clementel. Airflow and particle deposition simulations in health and emphysema: From in vivo to in silico animal experiments. *Annals of Biomedical Engineering*, 42(4):899–914, 2014.
- [58] D. Mukherjee, N.D. Jani, K. Selvaganesan, C.L. Weng, and S.C. Shadden. Computational assessment of the relation between embolism source and embolus distribution to the circle of Willis for improved understanding of stroke etiology. *Journal Of Biomechanical Engineering*, 138(8):081008–1–13, 2016.
- [59] K. B. Hansen and S. C. Shadden. A reduced-dimensional model for near-wall transport in cardiovascular flows. *Biomechanics and Modeling in Mechanobiology*, 15(3):713–722, 2016.
- [60] A. Arzani, A. M. Gambaruto, G. Chen, and S. C. Shadden. Lagrangian wall shear stress structures and near wall transport in high schmidt aneurysmal flows. *Journal of Fluid Mechanics*, 790:158–172, 2016.
- [61] E. O. Kung, A. S. Les, C. A. Figueroa, F. Medina, K. Arcaute, R. B. Wicker, M. V. McConnell, and C. A. Taylor. In vitro validation of finite element analysis of blood flow in deformable models. *Annals of Biomedical Engineering*, 39(7):1947–1960, 2011.

- [62] E. O. Kung, A. M. Kahn, J. C. Burns, and A. L. Marsden. In vitro validation of patient-specific hemodynamic simulations in coronary aneurysms caused by Kawasaki disease. *Cardiovascular Engineering and Technology*, 5(2):189–201, 2014.
- [63] W. Yang, F. P. Chan, V. M. Reddy, A. L. Marsden, and J. A. Feinstein. Flow simulations and validation for the first cohort of patients undergoing the Y-graft Fontan procedure. *The Journal of Thoracic and Cardiovascular Surgery*, 149(1):247–255, 2015.
- [64] D. A. Steinman, Y. Hoi, P. Fahy, L. Morris, M. T. Walsh, N. Aristokleous, A. S. Anayiotos, Y. Papaharilaou, A. Arzani, S. C. Shadden, et al. Variability of computational fluid dynamics solutions for pressure and flow in a giant aneurysm: the ASME 2012 Summer Bioengineering Conference CFD Challenge. *Journal of biomechanical engineering*, 135(2):021016, 2013.
- [65] A. L. Marsden, A. J. Bernstein, V. M. Reddy, S. C. Shadden, R. L. Spilker, F. P. Chan, C. A. Taylor, and J. A. Feinstein. Evaluation of a novel Y-shaped extracardiac Fontan baffle using computational fluid dynamics. *Journal Of Thoracic and Cardiovascular Surgery*, 137(2):394–U187, 2009.
- [66] M. H. Martin, J. A. Feinstein, F. P. Chan, A. L. Marsden, W. Yang, and V. M. Reddy. Technical feasibility and intermediate outcomes of a hand-crafted, area-preserving, bifurcated “Y-Graft” Fontan. *Journal of Thoracic and Cardiovascular Surgery*, 149(1):247–255, 2015.
- [67] C. P. Cheng, R. J. Herfkens, A. L. Lightner, C. A. Taylor, and J. A. Feinstein. Blood flow conditions in the proximal pulmonary arteries and vena cavae: healthy children during upright cycling exercise. *American Journal of Physiology-Heart and Circulatory Physiology*, 287(2):H921–H926, 2004.
- [68] D. Sengupta, A. M. Kahn, J. C. Burns, S. Sankaran, S. C. Shadden, and A. L. Marsden. Image-based modeling of hemodynamics in coronary artery aneurysms caused by Kawasaki disease. *Biomechanics and Modeling in Mechanobiology*, 11(6):915–932, 2012.
- [69] C. P. Cheng, R. J. Herfkens, C. A. Taylor, and J. A. Feinstein. Proximal pulmonary artery blood flow characteristics in healthy subjects measured in an upright posture using MRI: the effects of exercise and age. *Journal of Magnetic Resonance Imaging*, 21(6):752–758, 2005.
- [70] V. L. Morgan, R. J. Roselli, and C. H. Lorenz. Normal three-dimensional pulmonary artery flow determined by phase contrast magnetic resonance imaging. *Annals of Biomedical Engineering*, 26(4):557–566, 1998.

- [71] B. T. Tang, S. S. Pickard, F. P. Chan, P. S. Tsao, C. A. Taylor, and J. A. Feinstein. Wall shear stress is decreased in the pulmonary arteries of patients with pulmonary arterial hypertension: an image-based, computational fluid dynamics study. *Pulmonary Circulation*, 2(4):470–476, 2012.
- [72] M. S. Olufsen, C. S. Peskin, W. Y. Kim, E. M. Pedersen, A. Nadim, and J. Larsen. Numerical simulation and experimental validation of blood flow in arteries with structured-tree outflow conditions. *Annals of Biomedical Engineering*, 28(11):1281–1299, 2000.
- [73] D. N. Ku, D. P. Giddens, C. K. Zarins, and S. Glagov. Pulsatile flow and atherosclerosis in the human carotid bifurcation. positive correlation between plaque location and low oscillating shear stress. *Arteriosclerosis, thrombosis, and vascular biology*, 5(3):293–302, 1985.
- [74] E. Braunwald, E. M. Antman, J. W. Beasley, R. M. Califf, M. D. Cheitlin, J. S. Hochman, R. H. Jones, D. Kereiakes, J. Kupersmith, T. N. Levin, et al. Acc/aha 2002 guideline update for the management of patients with unstable angina and non-st-segment elevation myocardial infarction—summary article: a report of the american college of cardiology/american heart association task force on practice guidelines (committee on the management of patients with unstable angina). *Journal of the American College of Cardiology*, 40(7):1366–1374, 2002.
- [75] J. S. Coogan, J. D. Humphrey, and C. A. Figueroa. Computational simulations of hemodynamic changes within thoracic, coronary, and cerebral arteries following early wall remodeling in response to distal aortic coarctation. *Biomechanics and modeling in mechanobiology*, 12(1):79–93, 2013.
- [76] S. Roccabianca, C.A. Figueroa, G. Tellides, and J.D. Humphrey. Quantification of regional differences in aortic stiffness in the aging human. *Journal of the mechanical behavior of biomedical materials*, 29:618–634, 2014.
- [77] H. G. Bogren, R. H. Klipstein, D. N. Firmin, R. H. Mohiaddin, S. R. Underwood, R. S. O. Rees, and D. B. Longmore. Quantitation of antegrade and retrograde blood flow in the human aorta by magnetic resonance velocity mapping. *American heart journal*, 117(6):1214–1222, 1989.
- [78] A. B. Ramachandra, A. M. Kahn, and A. L. Marsden. Patient-specific simulations reveal significant differences in mechanical stimuli in venous and arterial coronary grafts. *Journal of Cardiovascular Translational Research*, 9(4):279–290, 2016.
- [79] J. R. Cezbral and R. Löhner. From medical images to anatomically accurate finite element grids. *International Journal for Numerical Methods in Engineering*, 51(8):985–1008, 2001.

- [80] H. M. Ladak, J. S. Milner, and D. A. Steinman. Rapid three-dimensional segmentation of the carotid bifurcation from serial mr images. *Journal of biomechanical engineering*, 122(1):96–99, 2000.
- [81] K. C. Wang, R. W. Dutton, and C. A. Taylor. Improving geometric model construction for blood flow modeling. *IEEE Engineering in Medicine and Biology Magazine*, 18(6):33–39, 1999.
- [82] L. Antiga, M. Piccinelli, L. Botti, B. Ene-Iordache, A. Remuzzi, and D. A. Steinman. An image-based modeling framework for patient-specific computational hemodynamics. *Medical & Biological Engineering & Computing*, 46(11):1097–1112, 2008.
- [83] *Triangulated Surface Boolean Operations for Combining 2-D AND 3-D Image Segmentation for Patient-Specific Blood Flow Analysis*. SB3C, 2015.
- [84] L. Piegl and W. Tiller. *The NURBS book*. Springer Science & Business Media, 2012.
- [85] A. Tayebi, J. Gomez Perez, and F. Catedra. Boolean operations implementation over 3d parametric surfaces to be included in the geometrical module of an electromagnetic solver. In *Antennas and Propagation (EUCAP), Proceedings of the 5th European Conference on*, pages 2137–2141. IEEE, 2011.
- [86] R. P.K. Banerjee and J. R. Rossignac. Topologically exact evaluation of polyhedra defined in csg with loose primitives. *Computer Graphics Forum*, 15(4):205–217, 1996.
- [87] S. Fang, B. Bruderlin, and X. Zhu. Robustness in solid modelling: a tolerance-based intuitionistic approach. *Computer-Aided Design*, 25(9):567–576, 1993.
- [88] S. Fortune and C. J. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proceedings of the ninth annual symposium on Computational geometry*, pages 163–172. ACM, 1993.
- [89] S. Fortune. Polyhedral modelling with exact arithmetic. In *Proceedings of the third ACM symposium on Solid modeling and applications*, pages 225–234. ACM, 1995.
- [90] H. Biermann, D. Kristjansson, and D. Zorin. Approximate boolean operations on free-form solids. In *Siggraph*, volume 1, pages 185–194, 2001.
- [91] J.M. Smith and N. A. Dodgson. A topologically robust algorithm for boolean operations on polyhedral shapes using approximate arithmetic. *Computer-Aided Design*, 39(2):149–163, 2007.
- [92] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 249–254. ACM Press/Addison-Wesley Publishing Co., 2000.

- [93] K. Museth, D. E. Breen, R. T. Whitaker, and A. H. Barr. Level set surface editing operators. *ACM Transactions on Graphics (TOG)*, 21(3):330–338, 2002.
- [94] C. C.L. Wang. Approximate boolean operations on large polyhedral solids with partial mesh reconstruction. *Visualization and Computer Graphics, IEEE Transactions on*, 17(6):836–849, 2011.
- [95] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H. Seidel. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 57–66. ACM, 2001.
- [96] N. K. Govindaraju, M. C. Lin, and D. Manocha. Fast and reliable collision culling using graphics hardware. *Visualization and Computer Graphics, IEEE Transactions on*, 12(2):143–154, 2006.
- [97] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, et al. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1):61–81, 2005.
- [98] F. Faure, S. Barbier, J. Allard, and F. Falipou. Image-based collision detection and response between arbitrary volume objects. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 155–162. Eurographics Association, 2008.
- [99] M. Trapp and J. Döllner. Real-time volumetric tests using layered depth images. In *Proc. Eurographics*, pages 235–238, 2008.
- [100] P. Yang and X. Qian. Direct boolean intersection between acquired and designed geometry. *Computer-Aided Design*, 41(2):81–94, 2009.
- [101] M. Schiffko, B. JÄEttler, and B. Kornberger. Industrial application of exact boolean operations for meshes. In *Proceedings of the 26th Spring Conference on Computer Graphics*, pages 165–172. ACM, 2010.
- [102] P. Cignoni, M. Callieri, M. Corsini, Matteo Dellepiane, F. Ganovelli, and G. Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian Chapter Conference*, volume 2008, pages 129–136, 2008.
- [103] C. Quammen, C. Weigle, and R. Taylor, M. II. Boolean Operations on Surfaces in VTK Without External Libraries. *Insight Journal*, 3.00, May 2011.
- [104] G. Mei and J. C. Tipper. Simple and robust boolean operations for triangulated surfaces. *arXiv:1308.4434*, 2013.
- [105] *Combining 2-D and 3-D Image Segmentation Techniques Using Triangulated Surface Boolean Operations.*, 2015.

- [106] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *ACM SIGGRAPH computer graphics*, volume 21, pages 153–162. ACM, 1987.
- [107] D. A. Field. Laplacian smoothing and delaunay triangulations. *Communications in applied numerical methods*, 4(6):709–712, 1988.
- [108] Alexander Belyaev and Yutaka Ohtake. A comparison of mesh smoothing methods. In *Israel-Korea Bi-national conference on geometric modeling and computer graphics*, volume 2. Citeseer, 2003.
- [109] P. S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. Technical report, DTIC Document, 1997.
- [110] H. Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *Proceedings of the conference on Visualization'99: celebrating ten years*, pages 59–66. IEEE Computer Society Press, 1999.
- [111] K. Vlachkova and P. Terziev. A comparison of surface subdivision algorithms for polygonal meshes. In *Application of Mathematics in Technical and Natural Sciences*, volume 1487, pages 343–350. AIP Publishing, 2012.
- [112] M. Chen, J. C. Hart, and S. C. Shadden. Hierarchical watershed ridges for visualizing lagrangian coherent structures. In *Topological Methods in Data Analysis and Visualization*, 2015.
- [113] A. Maneesh. Subdivision surfaces, 2010.
- [114] P. Bézier. Définition numérique des courbes et surfaces i. *Automatisme*, 11(12):625–632, 1966.
- [115] P. De Casteljau. Outillages méthodes calcul. *Andr e Citro en Automobiles SA, Paris*, 1959.
- [116] K. J. Versprille. Computer-aided design applications of the rational b-spline approximation form. 1975.
- [117] M. G. Cox. The numerical evaluation of b-splines. *IMA Journal of Applied Mathematics*, 10(2):134–149, 1972.
- [118] C. De Boor. On calculating with b-splines. *Journal of Approximation theory*, 6(1):50–62, 1972.
- [119] A. A.G. Requicha and H. B. Voelcker. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proceedings of the IEEE*, 73(1):30–44, 1985.

- [120] J. F. Thompson. Grid generation techniques in computational fluid dynamics. *AIAA journal*, 22(11):1505–1523, 1984.
- [121] R. Löhner. Extensions and improvements of the advancing front grid generation technique. *Communications in Numerical Methods in Engineering*, 12(10):683–702, 1996.
- [122] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. *Computing in Euclidean Geometry*, 1:23–90, 1992.
- [123] J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry*, 22(1):21–74, 2002.
- [124] G. O. Roberts. Computational meshes for boundary layer problems. In *Proceedings of the Second International Conference on Numerical Methods in Fluid Dynamics*, pages 171–177. Springer, 1971.
- [125] M. Ainsworth and J. T. Oden. *A posteriori error estimation in finite element analysis*, volume 37. John Wiley & Sons, 2011.
- [126] E. Bänsch. Local mesh refinement in 2 and 3 dimensions. *IMPACT of Computing in Science and Engineering*, 3(3):181–191, 1991.
- [127] G. Kunert. Toward anisotropic mesh construction and error estimation in the finite element method. *Numerical Methods for Partial Differential Equations*, 18(5):625–648, 2002.
- [128] T. J.R. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering*, 194(39):4135–4195, 2005.
- [129] A. Hrennikoff. Solution of problems of elasticity by the framework method. *Journal of applied mechanics*, 8(4):169–175, 1941.
- [130] R. Courant et al. Variational methods for the solution of problems of equilibrium and vibrations. *Bull. Amer. Math. Soc*, 49(1):1–23, 1943.
- [131] R. W. Clough. The finite element method in plane stress analysis. 1960.
- [132] T. J.R. Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.
- [133] J. A. Cottrell, T. J.R. Hughes, and Y. Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons, 2009.
- [134] T. W. Sederberg, Jianmin Zheng, A. B., and A. Nasri. T-splines and t-nurccs. In *ACM transactions on graphics (TOG)*, volume 22, pages 477–484. ACM, 2003.

- [135] Y. Bazilevs, V. M. Calo, J. A. Cottrell, J. A. Evans, T. J. R. Hughes, S. Lipton, M. A. Scott, and T. W. Sederberg. Isogeometric analysis using t-splines. *Computer Methods in Applied Mechanics and Engineering*, 199(5):229–263, 2010.
- [136] J. Warren and H. Weimer. *Subdivision methods for geometric design: A constructive approach*. Morgan Kaufmann, 2001.
- [137] J. Peters and U. Reif. *Subdivision surfaces*. Springer, 2008.
- [138] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-aided design*, 10(6):350–355, 1978.
- [139] F. Cirak, M. Ortiz, and P. Schroder. Subdivision surfaces: a new paradigm for thin-shell finite-element analysis. *International Journal for Numerical Methods in Engineering*, 47(12):2039–2072, 2000.
- [140] D. R. Forsey and R. H. Bartels. Hierarchical b-spline refinement. In *ACM Siggraph Computer Graphics*, volume 22, pages 205–212. ACM, 1988.
- [141] D. Schillinger, L. Dede, M. A. Scott, J. A. Evans, M. J. Borden, E. Rank, and T. J.R. Hughes. An isogeometric design-through-analysis methodology based on adaptive hierarchical refinement of nurbs, immersed boundary methods, and t-spline cad surfaces. *Computer Methods in Applied Mechanics and Engineering*, 249:116–150, 2012.
- [142] J. Deng, F. Chen, and Y. Feng. Dimensions of spline spaces over t-meshes. *Journal of Computational and Applied Mathematics*, 194(2):267–283, 2006.
- [143] Ping Wang, Jinlan Xu, Jiansong Deng, and Falai Chen. Adaptive isogeometric analysis using rational pht-splines. *Computer-Aided Design*, 43(11):1438–1448, 2011.
- [144] T. Dokken and V. Skytt. Locally refined splines. *Preprint*, 2010.
- [145] K. A. Johannessen, T. Kvamsdal, and T. Dokken. Isogeometric analysis using lr b-splines. *Computer Methods in Applied Mechanics and Engineering*, 269:471–514, 2014.
- [146] W. P. Thurston and S. Levy. *Three-dimensional geometry and topology*, volume 1. Princeton university press, 1997.
- [147] Y. Bazilevs, V. M. Calo, Y. Zhang, and T. J.R. Hughes. Isogeometric fluid–structure interaction analysis with applications to arterial blood flow. *Computational Mechanics*, 38(4-5):310–322, 2006.
- [148] Y. Zhang, Y. Bazilevs, S. Goswami, C. L. Bajaj, and T. J.R. Hughes. Patient-specific vascular nurbs modeling for isogeometric analysis of blood flow. *Computer methods in applied mechanics and engineering*, 196(29):2943–2959, 2007.

- [149] J.M. Escobar, J.M. Cascón, E. Rodríguez, and R. Montenegro. A new approach to solid modeling with trivariate t-splines based on mesh optimization. *Computer Methods in Applied Mechanics and Engineering*, 200(45):3210–3222, 2011.
- [150] Y. Zhang, W. Wang, and T. J.R. Hughes. Solid t-spline construction from boundary representations for genus-zero geometry. *Computer Methods in Applied Mechanics and Engineering*, 249:185–197, 2012.
- [151] R. MacCracken and K. I. Joy. Free-form deformations with lattices of arbitrary topology. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 181–188. ACM, 1996.
- [152] S. Schaefer, J. Hakenberg, and J. Warren. Smooth subdivision of tetrahedral meshes. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 147–154. ACM, 2004.
- [153] D. Burkhart, B. Hamann, and G. Umlauf. Adaptive tetrahedral subdivision for finite element analysis. *Comput. Graph. International, Electronic Proceedings*, 2010.
- [154] B. Li, X. Li, K. Wang, and H. Qin. Generalized polycube trivariate splines. In *Shape Modeling International Conference (SMI), 2010*, pages 261–265. IEEE, 2010.
- [155] T. Martin, E. Cohen, and M. Kirby. Volumetric parameterization and trivariate b-spline fitting using harmonic functions. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 269–280. ACM, 2008.
- [156] Y. He, H. Wang, C.-W. Fu, and H. Qin. A divide-and-conquer approach for automatic polycube map construction. *Computers & Graphics*, 33(3):369–380, 2009.
- [157] L. Shapira, A. Shamir, and D. Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer*, 24(4):249–259, 2008.
- [158] I. Temizer, P. Wriggers, and T. J.R. Hughes. Contact treatment in isogeometric analysis with nurbs. *Computer Methods in Applied Mechanics and Engineering*, 200(9):1100–1112, 2011.
- [159] J. Lu. Isogeometric contact analysis: Geometric basis and formulation for frictionless contact. *Computer Methods in Applied Mechanics and Engineering*, 200(5):726–741, 2011.
- [160] L. De Lorenzis, I. Temizer, P. Wriggers, and G. Zavarise. A large deformation frictional contact formulation using nurbs-based isogeometric analysis. *International Journal for Numerical Methods in Engineering*, 87(13):1278–1300, 2011.

- [161] W. A Wall, M. A. Frenzel, and C. Cyron. Isogeometric structural shape optimization. *Computer methods in applied mechanics and engineering*, 197(33):2976–2988, 2008.
- [162] X. Qian. Full analytical sensitivities in nurbs based isogeometric shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 199(29):2059–2071, 2010.
- [163] N. D. Manh, A. Evgrafov, A. R. Gersborg, and J. Gravesen. Isogeometric shape optimization of vibrating membranes. *Computer Methods in Applied Mechanics and Engineering*, 200(13):1343–1353, 2011.
- [164] Y. Bazilevs, J.R. Gohean, T. J.R. Hughes, R.D. Moser, and Y. Zhang. Patient-specific isogeometric fluid–structure interaction analysis of thoracic aortic blood flow due to implantation of the jarvik 2000 left ventricular assist device. *Computer Methods in Applied Mechanics and Engineering*, 198(45):3534–3550, 2009.
- [165] A. A. Nada. Use of b-spline surface to model large-deformation continuum plates: procedure and applications. *Nonlinear dynamics*, 72(1-2):243–263, 2013.
- [166] G. G. Sanborn and A. A. Shabana. On the integration of computer aided design and analysis using the finite element absolute nodal coordinate formulation. *Multibody System Dynamics*, 22(2):181–197, 2009.
- [167] H. Yamashita and H. Sugiyama. Numerical convergence of finite element solutions of nonrational b-spline element and absolute nodal coordinate formulation. *Nonlinear Dynamics*, 67(1):177–189, 2012.
- [168] A. Updegrave, N. M. Wilson, and S. C. Shadden. Construction of analysis suitable vascular models using axis-aligned polycubes. *Journal of biomechanical engineering*, 2018.
- [169] C. A. Taylor and D. A. Steinman. Image-based modeling of blood flow and vessel wall dynamics: Applications, methods and future directions. *Annals of Biomedical Engineering*, 38(3):1188–1203, 2010.
- [170] J. M. Oakes, S. C. Shadden, C. Grandmont, and I. E. Vignon-Clementel. Aerosol transport throughout inspiration and expiration in the pulmonary airways. *International Journal for Numerical Methods in Biomedical Engineering*, pages e2847–n/a, 2017.
- [171] S. H. Pahlavian, F. Loth, M. Luciano, J. Oshinski, and B. A. Martin. Neural tissue motion impacts cerebrospinal fluid dynamics at the cervical medullary junction: A patient-specific moving-boundary computational model. *Annals of Biomedical Engineering*, 43(12):2911–2923, Dec 2015.

- [172] T. M. Keaveny. Biomechanical computed tomography—noninvasive bone strength analysis using clinical computed tomography scans. *Annals of the New York Academy of Sciences*, 1192(1):57–65, 2010.
- [173] A. Pérez del Palomar, B. Calvo, and M. Doblaré. An accurate finite element model of the cervical spine under quasi-static loading. *Journal of Biomechanics*, 41(3):523 – 531, 2008.
- [174] Z. Ma, J. M. Tavares, R. N. Jorge, and T. Mascarenhas. A review of algorithms for medical image segmentation and their applications to the female pelvic cavity. *Computer Methods in Biomechanics and Biomedical Engineering*, 13(2):235–246, 2010.
- [175] D. L. Pham, C. Xu, and J. L. Prince. Current methods in medical image segmentation. *Annual Review of Biomedical Engineering*, 2:315–337, 2000.
- [176] Y. Zhang. *Geometric Modeling and Mesh Generation from Scanned Images*, volume 6. CRC Press, 2016.
- [177] M. Aigner, C. Heinrich, B. Jüttler, E. Pilgerstorfer, B. Simeon, and A.-V. Vuong. Swept volume parameterization for isogeometric analysis. In *IMA international conference on mathematics of surfaces*, pages 19–44. Springer, 2009.
- [178] Y. Zhang, W. Wang, and T. J.R. Hughes. Solid t-spline construction from boundary representations for genus-zero geometry. *Computer Methods in Applied Mechanics and Engineering*, 249:185–197, 2012.
- [179] B. Li, X. Li, K. Wang, and H. Qin. Surface mesh to volumetric spline conversion with generalized polycubes. *IEEE Transactions on Visualization and Computer Graphics*, 19(9):1539–1551, 2013.
- [180] H. Al Akhras, T. Elguedj, A. Gravouil, and M. Rochette. Towards an automatic isogeometric analysis suitable trivariate models generation—application to geometric parametric analysis. *Computer Methods in Applied Mechanics and Engineering*, 316:623–645, 2016.
- [181] K. Hu, Y. J. Zhang, and T. Liao. Surface segmentation for polycube construction based on generalized centroidal voronoi tessellation. *Computer Methods in Applied Mechanics and Engineering*, 316:280–296, 2017.
- [182] W. J. Schroeder and K. M. Martin. *The Visualization Toolkit-3.0*. Elsevier Inc., 1996.
- [183] I. Bitter, A. E. Kaufman, and M. Sato. Penalized-distance volumetric skeleton algorithm. *IEEE Transactions on Visualization and computer Graphics*, 7(3):195–206, 2001.

- [184] Y. Zhou and A. W. Toga. Efficient skeletonization of volumetric objects. *IEEE Transactions on visualization and computer graphics*, 5(3):196–209, 1999.
- [185] N. D. Cornea, D. Silver, X. Yuan, and R. Balasubramanian. Computing hierarchical curve-skeletons of 3d objects. *The Visual Computer*, 21(11):945–955, 2005.
- [186] L. Liu, E. W. Chambers, D. Letscher, and T. Ju. A simple and robust thinning algorithm on cell complexes. In *Computer Graphics Forum*, volume 29, pages 2253–2260. Wiley Online Library, 2010.
- [187] A. Sobiecki, A. Jalba, and A. Telea. Comparison of curve and surface skeletonization methods for voxel shapes. *Pattern Recognition Letters*, 47:147–156, 2014.
- [188] M. De Berg, Marc Van K., Mark O., and Otfried C. S. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 2000.
- [189] J. A. Sethian et al. Level set methods and fast marching methods. *Journal of Computing and Information Technology*, 11(1):1–2, 2003.
- [190] L. Antiga. Patient-specific modeling of geometry and blood flow in large arteries. *Politecnico di Milano*, 2002.
- [191] R. L. Bishop. There is more than one way to frame a curve. *The American Mathematical Monthly*, 82(3):246–251, 1975.
- [192] K. Hu and Y. J. Zhang. Centroidal voronoi tessellation based polycube construction for adaptive all-hexahedral mesh generation. *Computer Methods in Applied Mechanics and Engineering*, 305:405–421, 2016.
- [193] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. In *Computer Graphics Forum*, volume 21, pages 209–218. Wiley Online Library, 2002.
- [194] F. Hausdorff. Grundzuge der mengenlehre (leipzig: Veit). Technical report, ISBN 978-0-8284-0061-9, 1914.
- [195] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence*, 15(9):850–863, 1993.
- [196] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.
- [197] X. Chen, A. Golovinskiy, and T. Funkhouser. A benchmark for 3D mesh segmentation. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), August 2009.
- [198] L. Shapira, A. Shamir, and D. Cohen-Or. Consistent mesh partitioning and skeletonization using the shape diameter function. *The Visual Computer*, 24(4):249, 2008.

- [199] A. Hatcher and W. Thurston. A presentation for the mapping class group of a closed orientable surface. *Topology*, 19(3):221–237, 1980.
- [200] A. C. Ford, W. F. Chui, A. Y. Zeng, A. Nandy, E. Liebenberg, C. Carraro, G. Kazakia, T. Alliston, and G. D. O’Connell. A modular approach to creating large engineered cartilage surfaces. *Journal of biomechanics*, 67:177–183, 2018.