Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Estado de México

Simulation of Skeletal Muscles in Real-Time with Parallel Computing in GPU

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science

Autor: Octavio Navarro Hinojosa

Advisor:	Dr. Moisés Alencastre Miranda
Thesis Committee:	Dr. Jesus Enrique Chong Quero
	Dr. Isaac Rudomin Goldberg
	Dr. Benjamín Hernández Arreguín
	Dr. Sergio Ruiz Loza

Atizapán de Zaragoza, Estado de México
27th May, 2020

Abstract

Simulation of Skeletal Muscles in Real-Time with Parallel Computing in GPU

by Octavio Navarro Hinojosa

The animation of humanoid characters is an ongoing research area, with a special focus within the entertainment industry. However, the need of detailed human models has extended to areas such as medicine, and biomechanics, each with its own set of performance and functional requirements. While visual realism is more desirable in the entertainment industry, biomechanical accuracy and interactivity are most crucial in designing medical applications.

Creating biomechanically accurate human models is a great challenge because it requires a precise reconstruction of the different structures of the human body, as well as the biological and physiological functions that control them. Specifically, the modeling and simulation of the skeletal muscles have received special attention because they generate movement and help maintain the poses of the human body.

Most of the approaches that attempt to simulate the skeletal muscles of the body are based on models that are not ideal due to several reasons: biomechanical models, which are mechanical simplifications of the actual behavior of the muscles, are used; focus is given to their macroscopic behavior, leaving behind the mechanics and internal structures of the muscles; simulations are not processed in real-time, which is not ideal for specific applications, such as Computer Assisted Surgery, where interactivity is crucial.

In this work, a meshfree model that simulates skeletal muscles considering their functioning and control based on electrical activity, their structure based on biological tissue, and that computes in real-time, is presented. Meshfree methods were used because they are able to surpass most of the limitations that are present in mesh-based methods. The muscular belly was modelled as a particle-based viscoelastic fluid, which is controlled using the monodomain model and shape matching. The smoothed particle hydrodynamics (SPH) method was used to solve both the fluid dynamics and the electrophysiological model. To analyze the accuracy of the method, a similar model was implemented with the Finite Element Method (FEM). Both FEM and SPH methods provide similar solutions of the models in terms of pressure and displacement, with an error of around 0.09, with up to a 10% difference between them. The model was tested with simulations of contraction and extension of the long head of the triceps brachii and the vastus lateralis. The muscle's geometry was able to return to its original configuration after being innvervated with a stimulus current, displaying contractions and bulging similar to that of a real muscle. Through the use of General-purpose computing on graphics processing units (GPGPU), real-time simulations, with at least 70 frames per second, that offer a viable alternative to mesh-based models for interactive biological tissue simulations was achieved.

Contents

List of Figures

List of Tables

Abbreviations

API	**A**pplication **P**rogram **I**nterface
CC	**C**ontractile **C**omponent
CUDA	**C**ompute **U**nified **D**evice **A**rchitecture
DFFD	**D**irichlet **F**ree **F**orm **D**eformations
EMG	**E**lectro**M**yo**G**raphy
FDM	**F**inite **D**iference **M**ethod
FEM	**F**inite **E**lement **M**ethod
FFD	**F**ree-**F**orm **Deformations**
FFT	**F**ast **F**ourier **T**ransform
FHP	**F**rish-**H**asslacher-**P**omeau
FPS	**F**rames **P**er **S**econd
FVM	**F**inite **V**olume **M**ethod
GPU	**G**raphics **P**rocessing **U**nit
HPC	**H**igh-**P**erformance **C**omputing
ICCM	**I**nterconnected **C**able **M**odel
LBM	**L**attice **B**oltzmann **M**ethod
LGCA	**L**attice **G**as **C**elular **A**utomata
LS	**L**evel **S**et
MRI	**M**agnetic **R**esonance **I**maging
MPI	**M**essage **P**assing **I**nterface
MOCAP	**Mo**tion **Cap**ture
NaN	**N**ot **a** **N**umber
ODE	**O**rdinary **D**ifferential **E**quations
PCSA	**P**hysiological **C**ross-**S**ectional **A**rea
PDE	**P**artial **D**ifferential **E**quations

PE	**P**arallel **E**lement
RMS	**R**oot **M**ean **S**quare
SE	**S**eries **E**lement
SPH	**S**moothed **P**article **H**ydrodynamics
VOF	**V**olume-**O**f-**Fluid**
WCSPH	**W**eakly **C**ompressible **SPH**

To my advisor, Moisés, for giving me his precious time. To Sergio, for the valuable insights. For my parents, who always offered to help me when I was working late at night.
And, specially, dedicated to you; for alone I would not have had the strength to go on.

Chapter 1

Introduction

1.1 Motivation

Computational human modeling is an ongoing research area: from films and video games, to surgical planning simulations and medical training. Such simulations must often depict all aspects of a human being, ranging from its appearance to making them move and function in a believable manner.

The human body is composed of an intricate and complex anatomical structure which is made up of a variety of interacting tissues. Computational human modeling requires accurate reconstruction of this anatomical structure, the relevant biological and physiological functions, and their mathematical formulation into practical physical and mechanical models. Among the various tissues composing the body, those that form muscles carry out diverse physiological functions and collectively perform body movement.

The skeletal muscular system forms one of the major components of the human body mainly responsible for movement. They serve as major body components which make up nearly 50% of total body weight, characterizing the shape of a body and its tone. They also provide physiological functions to stabilize body posture and drive body movement. While the former is a key feature for the representation of the body, which demands accurate modeling of muscle morphology, the latter is crucial to produce accurate simulations of body movement driven by muscle activation. Skeletal muscles are composed of different materials like muscle, tendon, connective tissues, fat, etc. which form an inter-dependent system to work together and perform different kinds of activities. Considerable efforts have been made in the past few decades, related to the modeling of the skeletal muscles represented by various suitable mechanical models, but development of mechanically accurate and computationally effective models is still an open area of research.

Researchers have developed increasingly sophisticated biomechanical models of individual body parts based on skeletal muscles, such as hands [1, 2], head and neck[3], legs [4], facial animation [5–7], and the upper body [8]. However, even in the most detailed anatomical models,such as the ones presented by Zordan and Lee [8, 9], muscles are grouped and treated as single rigid objects, and their behaviour is modelled following the Hill muscle model [10]; which is a mechanically inspired simplification of the muscle's behaviour based on mass-less spring systems. Even more recent applications and models, such as the ones that can be implemented in the OpenSim [11, 12] software represent muscles as rigid lines, and focus on the analysis of movement, not on the complete tissue deformation and internal force, which limit their application in simulations such as in Computer-Assisted Surgery (CAS). Applications that could benefit from such a detailed biomechanical model of the tissue include: dynamic simulations and analysis of human movement [11, 13, 14], gait analysis [15, 16], evaluation of musculoskeletal pathology and assessment of treatment efficacy [17–20], prevention and rehabilitation [21–23], and even for sport medicine [24, 25].

Different approaches to simulating humans respond to different performance and functional requirements. For example, interactivity is required for real-time applications, such as virtual surgery simulators, but visual realism is more desirable in the entertainment industry. Moreover, biomechanical accuracy is most crucial in designing medical applications. A notable example is in CAS, where a connection to biomechanics has helped by defining a theoretical and numerical framework that provides information about the mechanics of the tissues after a clinical treatment or surgical intervention [26]. CAS has addressed a larger spectrum of clinical domains such as cardiology [27], neurosurgery [28], urology [29], and abdominal surgery [30]. For these applications, biomechanics faces a new challenge since the involved tissues are required to move and be deformed by stress generated by clinical actions. Moreover, soft tissues are difficult to model accurately since they typically exhibit complex, time dependent, non-linear, inhomogeneous and anisotropic behaviors. Such models are very computationally demanding and are therefore limited to pre-operative use, since the simulations often require many minutes or hours to compute. For clinical applications interactivity is critical, and reduced computational times are essential.

Many problems in biomechanics can be solved computationally using mesh-based methods such as the finite element method (FEM) [31]. However, finite element (FE) techniques, much like many other mesh-based methods, suffer from certain drawbacks in the modeling and simulation of biological systems such as soft tissue and cell deformation, or minimally invasive surgical simulation. The FEM has been successfully applied to modeling simplified models of the muscles [32–35], but modeling of complicated 3D muscle geometries increases the complexity of mesh generation for FE analysis. Poorly built meshes lead to mesh distortion and significant errors in FE analysis due to mesh distortion. Standard FE approaches are still ineffective in handling extreme material distortions owing to severe mesh distortion as it could occur in muscle deformations. In 3D subject specific models, fiber direction is measured at each of the muscle pixel points, which need to be interpolated at the integration points in FE model, which introduces additional approximation errors in the FE analysis.

Additional issues with the use of FE to simulate soft tissues include:

- Contact between tool and tissue must correspond to nodal points; hence, to prevent loss of accuracy, the density of the nodal points must be relatively high. This leads to high computational costs.
- Mesh distortion and entanglement during the large deformation of soft materials such as muscles, internal organs, or skin results in reduced accuracy. Remeshing can be used to overcome this, but it leads to increased computational time.
- In situations where local stresses/strains are extremely large, the creation of quality meshes for use within the problem domain is a prerequisite with FEM, which ultimately leads to a loss of manpower time.
- It is difficult to represent complex geometry with unstructured meshes, which is necessary for the remodeling process of biological tissues and the rupture of such biomaterials.

Recently, meshfree, or meshless, methods became a focus of interest for solving partial differential equations. In meshless methods the solid domain can be discretized with an unstructured cloud of nodes [36–39]. Truly meshless methods [39, 40] allow to acquire the nodal cloud directly from the Computerized Axial Tomography (CAT) scan or the Magnetic Resonance Image (MRI) by considering the pixels (or voxels) position and then obtain the nodal connectivity, the integration points and the shape functions using only the nodal spatial information. Using the grey tones of medical images, truly meshless methods are even capable of recognizing distinct biomaterial and then affecting directly to the nodes the corresponding material properties.

Meshfree methods, contrary to the FEM, allow the simulation of biological fluid flow (hemodynamics, swallowing, respiration, among others), and can deal with the large distortions of soft materials (internal organs, muscles, tendons, skin, etc.). Additionally, the smoothness and accuracy of displacements, stresses, and strain obtained with meshless methods are very useful to predict the remodelling process of biological tissues. Moreover, recent works have shown that the combination of medical imaging techniques (CAT scan and MRI) with meshless methods are more efficient than using the FEM [41, 42].

With the goal of simulating the mechanics of musculoskeletal systems, this thesis develops a method and useful set of meshless biophysical primitives that can be applied in the fields of biomechanics, anatomy or entertainment. The key challenge will be the creation of a muscle model that can represent a large array of shapes, and that behaves as their real counterpart. Furthemore, anatomy and biomechanics, in addition to their need of real-time simulators, require a higher standard of accuracy than applications for entertainment purposes.

1.2 Problem statement

A dynamic musculoskeletal system consists of several biological structures (bones, muscles, tendons, ligaments, connective tissue, etc.), and is an essential component for the simulation and analysis of human movement. However, simulating these structures is somewhat complicated due to the inherent complexity of both the system and the elements that comprise it. Existing simulators typically model the muscles as rigid objects, joining the physical properties of these with other nearby structures, such as bones, and apply phenomenological models to simulate their activation. Although there are several works that apply biophysical models, which try to simulate the correct structure of the muscles and their interaction with the other structures of the musculoskeletal system, the use of method such as FEM limit their use for real-time applications.

The FEM was one of the traditional means of modeling deformable solid and fluid dynamics [43], as well as the electromechanics of the biological tissue [34, 44]. While FEM is considered to be the gold standard for accuracy in computational methods, it is an unrealistic expectation that with the current technology included in a reasonable desktop PC that FEM could be made to perform interactively [45]. Even with the use of GPUs and with speedups of more than 20 [46], the simulations still are not run in real-time. Other solutions still have this issue, as is the case of Johnsen et. al. [47], who take around 20 seconds to complete an individual simulation; or Strbac et. al. [48], who achieve speedups between 30 and 120, but still take at least $\tilde{4}0$ seconds for their simulations.

The computational cost of the FEM for real-time applications is such an issue that alternative methods for its solution have been proposed. For example, Lorente et. al. [49] proposed the use of machine learning techniques for modelling the biomechanical behaviour of human liver. A similar technique for soft tissue was proposed by Meister et. al. [50].

Another issue with FEM is that the generation of complex meshes is necessary, and it represents a bottleneck for the method; specifically for the clinical translation of biological modeling tools; since it is difficult to have a streamlined and automated pipeline to generate accurate FE simulations from imaging data [51] (even though works such as [34] used a three-dimensional, anatomically based representation of the muscle, obtained from the Visible Human male data set [52]). Another non-trivial step of FEM is the coupling between physical properties and mechanics when meshes with different resolution for both problems are used. It is expected that a way to overcome these difficulties could be through a meshless approach, such as smoothed particle hydrodynamics (SPH) [53].

In recent years, SPH has become increasingly popular in computer graphics. It has been successfully used for the simulation of various fluid phenomena, such as multiphase fluids, rigid and elastic solids, deformable objects, fluid features, such as spray, foam and tiny air bubbles, granular materials, and other complex scenes that use multi millions of sampling points [54–57]. Recently, the SPH based method has been applied to solve non-hydrodynamic partial differential equations such as the wave equation, the diffusion equation, Maxwell's equations and Poisson's equation [58], as well as to electronic structure calculations [59].

Regarding the simulation of biological tissue, meshless approaches, such as SPH, have proven to be ideal to model the complex structures needed for soft tissue simulations [45, 60]. Joldes et. al. [61] have even suggested that because of the simplicity of meshless methods, they are better suited than FEM for integration with clinical workflows. Several works tackled such simulations while exposing some advantages of using SPH: Gastélum et al. [62, 63] integrated the effect of internal and external forces, and demonstrated the advantage of using SPH for large tissue deformations; Boyer and Joslin [45] considered the fibers of articular cartilage, and considered elastic and stress models to represent them; Palyanov et al. [64] used SPH to simulate different types of tissue, both solid and fluid, and introduced contractile fibers based on mass-spring systems; Rausch et al. [60] used SPH to simulate tissue that experience large deformations and damage, to the point of failure. Most of these works results were in agreement with analytical solutions, as well as FEM solutions. However, mass-spring models are impractical for tissue simulations, since they are non-volumetric methods where spring elements connecting point masses must be tuned for each desired scenario [45, 65].

These solutions do not consider biophysical models to control the activation and deformation of the tissues. Additionally, the tissues are modeled using mass-spring, elastic, or stress-based models that are not necessarily applicable to biological tissue simulations. Furthermore, previous models do not consider the physical properties of the tissue, or use approaches which are not completely able to capture the properties of tissue [64]. These are issues present in many works that simulate skeletal muscles. Current computational models of skeletal muscle models typically focus on simplified phenomenological relationships mimicking the overall (mechanical) behavior of a single skeletal muscle.

There are many challenges that have to be solved before being able to properly simulate the skeletal muscles. Some of the main ones are mentioned below:

- Most existing simulations simplify the musculoskeletal system, either by simplifying its architecture, its internal components, or its form of activation and control. Models such as Hill's 2.2.3 do not consider the mass of the muscle, since they are based on mass-less spring systems.

- Most musculoskeletal systems often ignore the muscle's anatomical characteristics.
- Even though some simulations of biological tissue presented have achieved real-time processing, most have not. The resulting simulations are used for offline processing of specific conditions, but are not suitable for interactive simulations, such as training environments.
- Almost all the tissues of the system are deformable, and have physical properties. Some effects such as the bulging of active muscles depends on the fact that the muscles are incompressible solids. In most previous work, the volumetric tissue is ignored or simulated using techniques such as FEM, requiring additional techniques or models to represent the tissue's deformation.
- Simulators based on solids mechanics, such as FEM, are not ideal for interactive applications because they require collision detection and conflict resolution, which are computationally expensive when applied to FEM models.
- The main methods to control the activation of the muscles are phenomenological. These are simplifications of the actual behavior of the muscles, and they have been shown to have a strong error level when compared to actual muscle activation data. These models are more similar to simple mechanical systems, and only model the force generated between two points, the origin and the insertion of the muscle. In addition, the muscles are controlled by the nervous system, and there are few works that focus on simulating their activation in such a way.
- Most of the simulations use FEM, the finite difference method (FDM), or the finite volume method (FVM), to model the geometry, and to solve the electrophysiology of the tissue. However, these methods have several disadvantages, from the formation of the mesh, the computational cost, to the significant errors in the analysis due to mesh distortion that occurs in muscle deformation.
- The use of parallel techniques for computing have dramatically improved execution times, but real-time applications require more processing power. The use of general purpose graphics processing unit (GPGPU) can lead to the reductions in time needed to develop such applications.
- The use of alternative solution methods have to be explored to model biological tissues. Methods such as the lattice Boltzmann method (LBM) have already been used to solve the electrophysiology of the tissue, and show, through good agreement with analytical solutions and numerical results, their viability. Meshfree methods and, in particular, SPH,

have also been used to simulate volume conserving solids. Because of its adaptability, it could also be used to model biological tissue more effectively.

Currently there is no model of the musculoskeletal system that simulates the muscles of the human body considering its internal structure, the biological tissues that compose them, their physical properties, and that are activated using biophysical models similar to activation by the nervous system.

Additionally, previous work focuses on modeling a specific muscle, or a specific group of muscles, without seeking to generate more general methods that allow modeling a wide variety of muscles, or even different biological tissues. An example of this is the work done using FEM, since they have a strong limitation regarding the shape of the muscles, since the most complex are not easily modelled using hexahedrons or tetrahedra. Similarly, several works simplify the muscles to the extent that they are only modeled as a line segment, regardless of its shape or configuration in three dimensions.

The computational costs to solve the different mathematical models, as well as to render the muscles, make obtaining simulations in real-time difficult; very few works use GPGPU to parallelize the calculations related to the simulations in order to be able to execute them in real-time. This makes it difficult to use the proposed works in more interactive solutions, such as training or learning simulators of anatomy and medicine, or in video games or animated films with humanoid characters.

1.3 Hypothesis

The use of meshless methods and GPGPU to simulate biological tissue, specifically, skeletal muscle, will allow the development of accurate, real-time, interactive simulations. Additionally, by defining the tissue as a viscous fluid, the model will be based more like an actual muscle, while being able to include biophysical properties.

1.4 Objectives

Develop and implement a computational framework designed for the real-time physical simulation of biomechanical tissue, specifically, skeletal muscle belly, that must consider the biophysical properties of the tissue. This is achieved by developing a meshfree method, based on SPH, that simulates tissues as viscoelastic fluids. The simulation model can be obtained from a number of sources, including MRI, without the need of additional mesh generation. To activate and produce contractions and extensions in the skeletal muscles, a biophysical model of electrophysiology, particularly, the monodomain model, is to be integrated into the method. The electrophysiology model will also be solved using the SPH formulation. Further, develop the method considering parallel paradigms, specifically using GPGPU, to achieve efficient and interactive simulations of tissues. Additionally, the proposed model aims to be an alternative to mesh-based methods, such as FEM, by reducing the computational costs and pre-processing and post-processing steps needed.

1.5 Specific objectives

1) Define a meshfree biophysical tissue model that can guarantee the conservation of volume, and that considers the biophysical properties of the tissue to be simulated. The tissue will be simulated as a viscoelastic fluid.

2) Integrate the properties, and necessary control formulations, of skeletal muscle belly.

3) Present efficient algorithms, and optimize the use of computational resources to achieve real-time and interactive simulations. Integrate GPGPU to accelerate the model and achieve real-time simulations.

4) Develop, as a case study, the simulation of the muscle belly of the long head of the triceps brachii and of the vastus lateralis, which are able to contract and relax when an electric current innervates them.

5) Develop a simulation of tissue using FEM to validate the accuracy of the proposed solution.

Chapter 2

Background

For this work different concepts and methods are used in the areas of anatomy, biomechanics, computer graphics, parallel and concurrent programming, among others. This chapter details, for each of the areas, relevant information that will be used for modeling muscles and to understand the concepts and terms of the following chapters in this proposal.

2.1 Anatomy

The biology of humans is a fairly broad area of study. Therefore, there is a set of relevant tissues that define the shape and function of the muscles of the body: bones, ligaments, tendons, and muscles. These structures are the ones that allow the movement of different parts of the body and that directly impact the changes in its surface. These structures are categorized into active and passive. Muscles are considered active structures as they are capable of producing forces on their own. Passive structures consist of materials that do not actively produce force on their own; these materials exhibit tension when pulled by other external forces. The bones, tendons, and ligaments belong to this group. An example of such structures can be seen in the Figure 2.1.

All these structures form a cohesive and efficient organic system. The bones create a skeleton that provides a structural support of the body, as well as protection of internal organs. Mechanically, together with the muscles and tendons, a complex system of pulleys is created that allows the locomotion of the body. Ligaments provide stability at joints by preventing segments of adjacent bones from being separated. They act as a guide for movement as the joints move. The muscles are the main generators of force and are attached to the tendons, which transmit the force of the muscles to the bones where they are united [67].

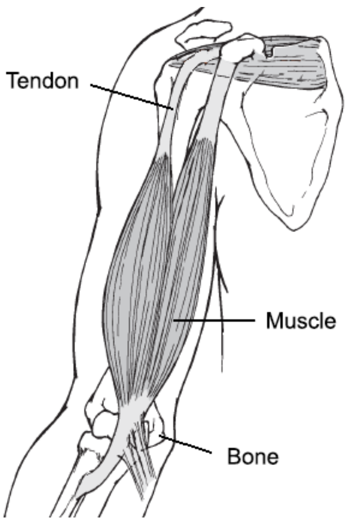


Figure 2.1: Example of active and passive structures of the arm. Adapted from [66].

2.1.1 Muscles

The muscular system consists of three types of muscles: the cardiac muscle, which makes up the heart; smooth muscle (non-striated or involuntary), which make up the walls of hollow internal organs and blood vessels; and skeletal muscle (striated or voluntary), which is attached to the skeleton by the tendons, and is responsible for generating the necessary force to generate movements. The first two types of muscles are controlled by the autonomic nervous system, and contract without the need for conscious effort. Unlike the first two types of muscle, the skeletal muscle is controlled through the somatic nervous system, and contractions are mainly done consciously. These voluntary contractions produce forces that are transferred to the skeleton, resulting in body movements [67]. Due to this property of movement generation of the skeletal muscles, this work will study the structure and properties only of this type of muscles.

2.1.1.1 Skeletal muscles

Skeletal muscles are among the most abundant tissues in the human body, consisting of between 40 % and 45 % of the total body weight. There are more than 430 skeletal muscles, found in pairs on the left and right sides of the body. The most vigorous movements are produced by less than 80 pairs. These muscles provide protection and strength to the skeleton by distributing loads and absorbing impacts. They allow the bones to move in the joints (through dynamic work), and maintain the body posture (through static work). Usually, these actions represent the joint action of muscle groups, not just individual muscles.

2.1.1.2 Composition, organization, and structure of muscles

The muscles of the body consist of two discrete units: the muscle belly, and two tendinous limbs that attach the muscle belly to the bone. The muscular belly consists of the muscle cells, or fibers, that produce contraction, and the connective tissues that wrap around the muscle fibers.

The muscles have a hierarchical structure, which can be seen in the Figure 2.2. The outer layer of connective tissue that covers the muscle belly is called the epimysium; this joins the muscle belly with the tendon. Internally, the muscle belly is composed of a large number of groups of muscular fibers, called fascicles, which are separated from each other by a layer of connective tissue called perimysium. At the same time, each fascicle is composed of muscle fibers, which are isolated from each other by the endomysium. These muscle fibers are the main structures of the muscles; each fiber ranges from 1 to 400 mm in length, and from 10 to 60 μm in diameter. Each muscle fiber consists of parallel groups of myofibrils. Finally, each myofibril is composed of a series of contractile units, called sarcomeres, which are responsible for producing contractions associated with muscles.

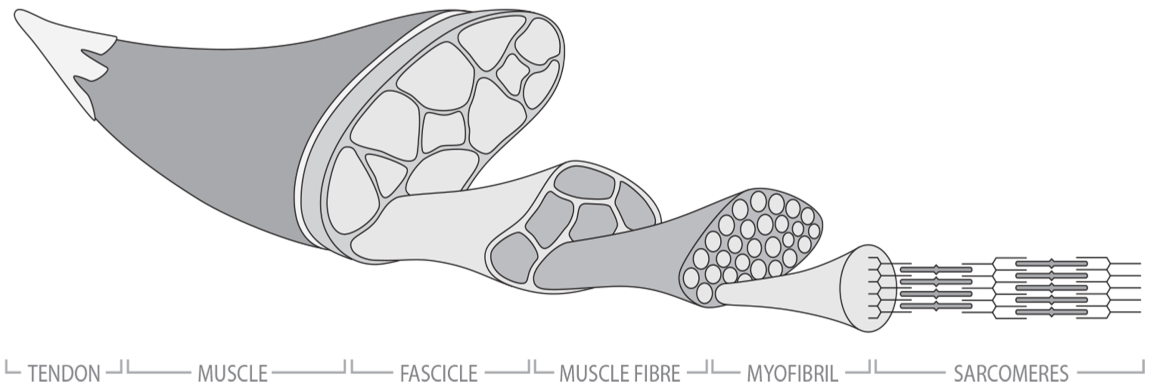


FIGURE 2.2: Hierarchical structure of skeletal muscles. [68].

The architecture of the muscles refers to the internal arrangement of the fascicles. A small number of muscles have simple architectures, where the fascicles fit in parallel to each other along the muscle. These are typically long muscles, such as the biceps. However, in most muscles, the fascicles have an orientation characterized by the angle they form respect to the tendons that are attached to them. This accommodation of fibers is known as pennate muscle. There are several types of patterns in pennate muscles, as can be seen in the Figure 2.3. These different types of architectures determine the range of movements and the force that is produced by each muscle. A muscle will contain a greater number of small muscle fibers in a pennate configuration than in a parallel one. Because of this, the pennate muscles do not contract enough, but can produce much more force than parallel muscles of the same size [66].

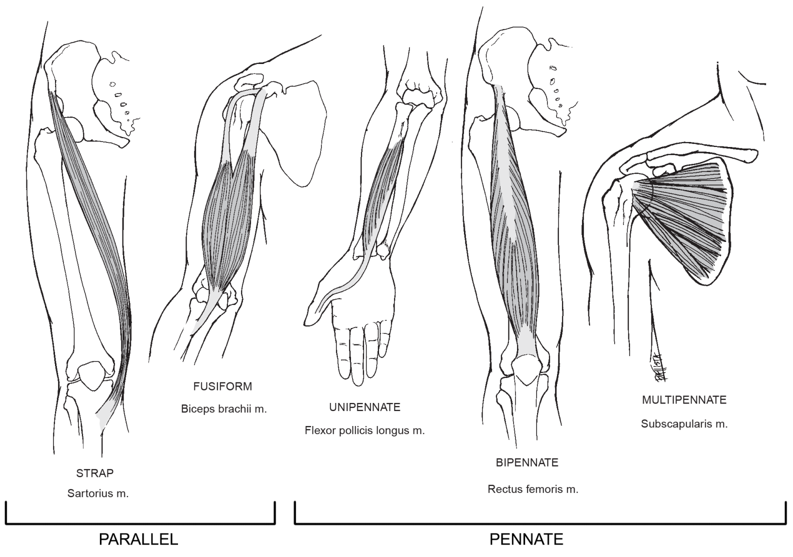


Figure 2.3: Examples of different architectures of muscles. [66].

In general, each end part of the muscle is connected to a bone through tendons, which have no active contractile properties. The muscles are those that form the contractile components and the tendons have elastic elements in series. The epimysium, perimysium, endomysium, and sarcolemma act as elastic elements in parallel. The forces that are produced by the contraction of the muscles are transmitted to the bones through the tendons. These transmit the forces produced from the muscle to which they are attached towards the bones. Tendons connect muscles either in a narrow area or along a broad, flat area, known as the aponeurosis. The union of the muscles to more stationary bones (proximal place) is called the origin, while the part that joins more movable bones (distal place) is called the insertion.

2.1.1.3 The motor unit

The functional unit of the skeletal muscles is the motor unit, which includes a motor neuron, and all the muscle fibers that are innervated by it. This unit is the smallest part of a muscle that can be made to contract independently. When stimulated, all muscle fibers in the motor unit respond as one. When the fibers of a motor unit receive a stimulus, they have one of two possible behaviors: they contract to the maximum, or they do not contract [67, 69].

The number of muscle fibers that make up a motor unit is related to the level of control needed for each muscle in particular. In small muscles that perform very fine movements, such as extraocular muscles, each motor unit may contain less than a dozen muscle fibers, whereas a large muscle that performs rough movements, each motor unit can contain between 2000 muscle fibers. The muscles that control subtle yet complex movements have fewer fibers per motor unit (less than 10 fibers), while the muscles that control larger movements have more fibers per motor unit (between 100 and 1000 fibers). Normally, motor units with less fibers are activated before motor units with more fibers.

The fibers of each motor unit are not contiguous, but are scattered along the muscle with fibers from other units. Therefore, if a motor unit is stimulated, a long portion of the muscle appears to contract. If additional motor units are stimulated, the muscle contracts with greater force. Using additional motor units in response to increased stimulation is called recruiting.

Voluntary contractions of a muscle start in the frontal motor cortex of the brain, where the impulses travel along the corticospinal track to the muscles. These impulses of the motor cortex are called *action potentials*, and each impulse is related to only one motor unit.

A muscle can be represented as n motor units that are controlled by n nerve axons that originate from the central nervous system, each with its own neural excitation function $u(t)$, which generates a muscular activation $a(t)$. The muscle fibers of each motor unit i together generate a force F_i^M, which always add up with the forces of other motor units to produce the force of the muscle F^M [70].

2.1.1.4 The muscle-tendon unit

Muscles and tendons work together to create a functional unit of force generation and transmission, that structure is called the muscle-tendon unit. Hill [10] demonstrated that tendons represent a spring-like elastic component located in series, a series element (SE), with a contractile element (CE) representing the contractile proteins of myofibrils, actin, and myosin, whereas the epimysium, perimysium, endomysium, and sarcolemma are represented by a second elastic component in parallel, a parallel element (PE), to the CE. An example of the muscle-tendon unit can be seen in the Figure 2.4.

2.1.1.5 Muscle contraction

The mechanism that allows muscles to generate movement is muscle contraction. The contraction of a complete muscle is the sum of singular contractions that occur within each of the sarcomeres. It is controlled by the central nervous system, where nerve impulses are generated which travel through motor neurons to the sensory somatic branch in the muscle. The place where the termination of a motor neuron and the muscle fiber are connected is called the neuromuscular junction. Each motor neuron innervates a set of muscle fibers where each impulse stimulates the flow of calcium to the sarcomeres, causing the filaments to slide. Sarcomeres have protein-based structures, thin filaments with high tensile strength (actin), coarse filaments (myosin), elastic filaments (tiltin), and inelastic filaments (nebulin). The actin and myosin are stacked on top of each other alternately, forming cross-bridges to produce force.

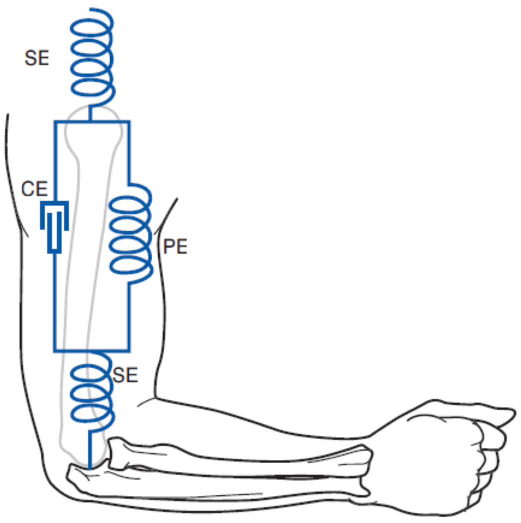


FIGURE 2.4: The muscle-tendon unit consists of a CE in parallel with an elastic component, PE, and in series with another elastic component, SE. [67].

The theory of sliding filaments and cross-bridge [71, 72], describes the process of muscle contraction. During muscle contraction, the length of the actin and myosin filaments remains constant and they slide on each other to increase their overlap, generating a shortening in the muscle, as can be seen in Figure 2.5. The force of contraction is generated by the myosin heads, or cross bridges, in the region where actin and myosin overlap. The cross bridges rotate in an arc around their position on the myosin filaments. This movement of the crossed bridges in contact with the filaments of actin causes the sliding of these towards the center of the sarcomere. Muscle fiber contracts when the sarcomeres are shortened simultaneously.

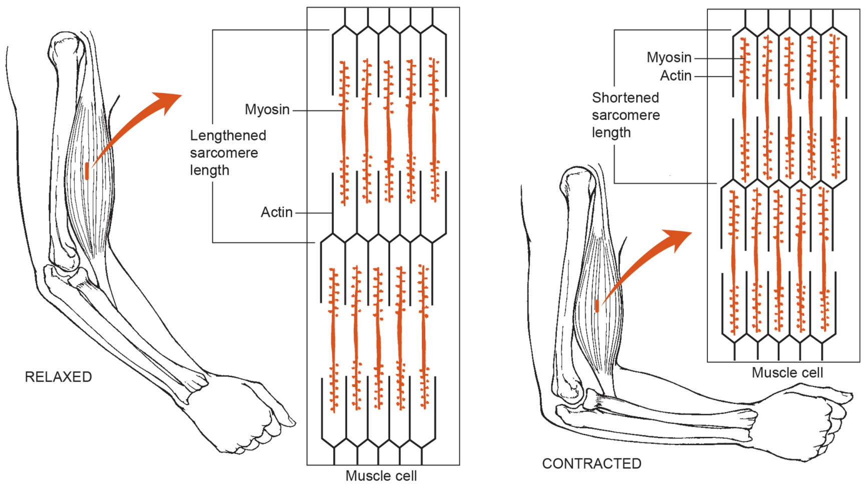


FIGURE 2.5: Example of muscle contraction. [66].

The contraction of the muscles can be classified according to the change in length of the muscle fibers or the level of force produced. In isotonic contraction, the length of the muscle changes and a force is generated; the muscle either contracts or expands, depending on whether the force produced is sufficient to withstand an external load. In isometric contraction, the muscle remains unchanged while producing force, such as, for example, when holding an object without movement. Although no movement is generated in this type of contraction, energy is used and is almost always dissipated as heat.

The amount a muscle fiber can shorten is proportional to its length [66, 73–75]. A fiber can shorten roughly 50 to 60% of its length [73, 76], although there is some evidence that fibers exhibit varied shortening capabilities [75]. An individual whole muscle is composed mostly of fibers of similar lengths [75]. However there is a wide variation in fiber lengths found in the human body, ranging from a few centimeters to approximately half a meter [77, 78].

2.2 Biomechanics

Biomechanics is the science that examines the forces acting on and within a biological structure, and the effects produced by those forces. External forces acting on a system are quantified using sophisticated measuring devices. Internal forces, which are generated by muscle activity, by external forces, or both, are evaluated using measurement devices implanted in areas of interest, or with estimates of a mathematical model. Possible results of internal or external forces are:

- Movements of segments of interest.
- Deformation of biological material.
- Biological changes in the tissues on which they act.

For this reason, biomechanics studies or quantifies the following:

- Movements of different body segments, and factors that influence movement, body alignment, weight distribution, among others.
- Deformation of biological structures, and the factors that influence their deformation.
- The biological effects of forces acting locally on living tissue; effects such as growth and development, or overload and injury.

In the present work, we will focus on quantifying and simulating the behavior of skeletal muscles. For this, it is necessary to mention the different models with which the generation of force in the muscle is calculated.

2.2.1 Phenomenological models and biophysical models

Based on the modeling and muscle simulations approach, most models can be grouped into two categories: phenomenological models, and biophysical models [34, 79]. Phenomenological models use mathematical representations to describe the mechanical properties of skeletal muscles, based on experimental measurements of them. One of the most used phenomenological models is the Hill muscle model [10].

In contrast, biophysical models seek to predict the response of muscles to a given stimulus, considering the underlying physiology of skeletal muscles. An example of a biophysical model is Huxley's muscle contraction theory, which considers the function and interaction of muscle fibers to generate movement. These models focus on simulating the internal arrangement of muscle fibers, allowing visualization of the breeding patterns within the volume of a muscle, as well as a more detailed analysis of the distribution and size of the fibers within the muscles [80]. This knowledge would be essential for physiotherapists, surgeons and orthopedists, especially for rehabilitation programs for patients with neuromuscular problems.

2.2.2 Spring-shock absorber-based muscle actuators

One way to model the forces generated by muscles is to use linear actuators whose direction is determined by a line segment connecting two members at the points of origin and insertion. Alexander [81] refers to muscles as springs in various situations, particularly when the length of the tendon is relatively long compared to the muscle.

In computer graphics, the force lines are modeled as spring-damper systems. These systems have already been used to simulate animals, such as fish [82], and snakes [83]. However, modeling muscles and tendons requires a model that has parameters similar to the characteristics of such structures. This model would allow the use of empirical measures to parameterize and capture different characteristics of any muscle in the body. In addition, it would allow the generation a large number of movements without having to adjust the different parameters. One such model is the three-element model of Hill [10].

2.2.3 The three-element model of Hill

Hill's three-element model, or Hill's model of muscles, is a phenomenological model, based on a series of controlled experiments on frog muscles, which models the force-length and force-velocity dependencies observed in an activated muscle. Even though it is one of the main models used, which is able to estimate qualitative patterns of muscular activation, as well as the mechanical properties of the muscle, it does not capture all the characteristics of the muscles. The model has three main components: a SE representing the tendon, a PE representing the connective tissues, and a CE representing the contractile proteins [67]. The Figure 2.6 shows the three-element model of Hill.

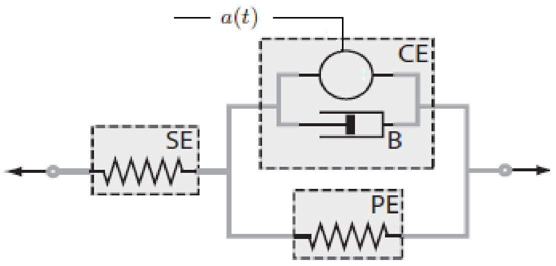


FIGURE 2.6: The Hill model describes the strength of a muscle as the sum of three elements, the CE, the SE, and the PE. B is the viscosity of the muscle. $a(t)$ is the activation signal. Adapted from [68].

2.2.3.1 Element in series

The SE groups several of the effects of various biological materials on the muscle. This element mainly represents the elastic effects of the tendon, and the elasticity of the structures within the sarcomere; the latter is usually omitted since the elasticity of the tendon dominates. Another property of the tendon that is omitted from the simulations is its viscosity. This since the damping factor offered by the viscosity is negligible.

2.2.3.2 Parallel element

This element represents the passive elastic properties of the muscles, regardless of the active contraction of the muscle. PE represents the passive elasticity of the connective tissues (endomysium, perimysium, and epimysium) of the muscles. The PE is responsible for the passive behavior of the muscle when stretched. Due to the material properties of these tissues, tension only occurs when the PE is actively stressed beyond its resting length.

2.2.3.3 Contractile Element

The CE is responsible for the active generation of force, which depends on the length of the muscle l^m, and the time-varying activation signal $a(t)$, which originates from the nervous system central and reaches the muscle motor units.

Since a neural stimulation occurs as a pulse train, the frequency determines whether a complete activation can occur. At low frequencies, each pulse is followed by a nerve contraction. If contractions occur at short intervals of time as the frequency increases, the contractions are joined in a continuous flow of force production. Usually there are several milliseconds of delay before an initial stimulation is made and the muscle contracts.

2.2.4 Cellular model of skeletal muscles

The study of interactions between different molecular components (such as calcium, sodium, myosin, or actin) leads to the generation of mathematical models that are typically described by Ordinary Differential Equations (ODEs). At the cellular scale, these phenomenological models have to describe the interactions of ionic concentrations at the intra- and extra-cellular levels. The electrical potentials generated by variations in concentration levels trigger the molecular bonds of actin and myosin and thus influence the mechanical properties and the generation of force in the muscle [84, 85].

Hodgkin and Huxley [86] formulated a first model based on experiments with giant squid axons. The transport of ions along cell membranes (membranes separating the interior of cells from an outside environment) is mathematically described by a set of ODEs. The Hodgkin and Huxley model is limited to the interactions of Sodium (Na) and potassium (K), in addition, leaks can occur along the cell membrane.

The transmembrane potential (ie potential that occurs or occurs through a cell membrane) V_m is found in:

$$C_m V_m = Q \tag{2.1}$$

where C_m is the capacitance of the membrane, and Q is the electric charge. Assuming that the capacitance does not change over time, you get:

$$C_m \frac{dV_m}{dt} = \frac{d}{dt} Q = I \tag{2.2}$$

Using Kirchhoff's law, you can get:

$$C_m \frac{dV_m}{dt} = I_{ext} - I_{Na} - I_K - I_l \tag{2.3}$$

Being I_{Na}, I_K the ionic currents for the sodium and potassium molecules, respectively, I_l the leakage current, and I_{ext} is the excitation current (current required for the model to start working).

2.2.4.1 FitzHugh/Nagumo model

FitzHugh [87] and Nagumo et al. [88] independently published a generalisation of Van der Pol's equation (the first model that represented the heart's dynamics) for a relaxation oscillator to provide a simplified unifying concept for the theoretical study of axon physiology. This model has become known as the FitzHugh-Nagumo model of nerve membrane [89].

By considering the Hodgkin & Huxley model as one member of a large class of non-linear systems showing excitable and oscillatory behaviour and through the application of phase space methods, FitzHugh reduced the four state variable Hodgkin & Huxley model to a two state variable model. This model can be taken as representative of a wide class of non-linear excitable-oscillatory systems which show threshold and refractory properties as well as oscillations or infinite trains of responses.

While the original two-variable model described a non-dimensional activation variable (x or u) and a non-dimensional recovery variable (y or v), here the model is formulated in terms of the 'real' action potential given by the time course of the transmembrane potential Vm. In so doing, the time rate of change of the activation variable describes the total 'ionic current' through the membrane with the original model parameters adjusted to give the correct dimensionality.

2.2.5 Bidomain Model

The most common approach for modeling the electrical activity of biological tissue [34, 84], which do not describe the electrophysiology of a single cell, is to solve the bidomain model.

The bidomain model [90–92] is the most complete description of cardiac electrical activity. It describes both the intracellular and extracellular potential fields, linking them through membrane behavior. The bidomain model, coupled with accurate models of cell membrane kinetics, is generally believed to provide a reasonable basis for numerical simulations of electrophysiology [93]. This model has also been used to simulate skeletal muscle electrophysiology [34, 84].

In general, mathematical models of biological electrophysiology consist of a system of partial differential equations (PDEs), coupled nonlinearly to a system of ODEs modeling the membrane dynamics.

The electrical activity of a cell is usually described by a system of ODEs of the form

$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = \mathbf{f}(\mathbf{u}, V_m) \tag{2.4}$$

where **u** is a vector of dependent variables, **f** is a prescribed vector-valued function, V_m is the transmembrane electrical potential, and t is time. This system of ODEs is coupled to a differential equation used to calculate V_m.

The basic bidomain equations [94–96] are given by

$$\chi\left(C_m \frac{\partial V_m}{\partial t} + I_{ion}\right) - \nabla \cdot (\sigma_i \nabla (V_m + \phi_e)) = I_{s_i} \tag{2.5}$$

$$\nabla \cdot ((\sigma_i + \sigma_e)\nabla \phi_e + \sigma_i \nabla V_m) = I_{s_e} \tag{2.6}$$

where σ_i and σ_e are respectively the intracellular and extracellular conductivity tensors, χ is the surface to volume ratio of the cardiac cells, ϕ_e is the extracellular potential, C_m is the membrane capacitance per unit area, V_m is an electrical potential, I_{ion} is the ionic current, I_{s_i} is the external stimulus applied to the intracellular space, and I_{s_e} is the external stimulus applies to the extracellular space. The ionic current I_{ion} is calculated using an electrophysiological cell model. There are a large number of cell models that give rise to systems of equations such as Equation 2.4; refer to [89] for a comprehensive collection of cell models for cardiac electrophysiology.

2.2.5.1 Monodomain Model

Since the bidomain model consists of a complex PDE system, that involves computational expensive numerical solution, it is common to assume that the intra and extracellular domains have equal anisotropy ratios to obtain a simplified model called the monodomain model [94, 97]. Following this assumption, the monodomain model can be obtained by a reduction from the bidomain model and is entirely written in terms of the transmembrane potential, defined as the difference between the intra and extracellular potentials.

The monodomain model is given by

$$\chi\left(C_m \frac{\partial V_m}{\partial t} + I_{ion}\right) = \nabla \cdot (\sigma \nabla V_m) \tag{2.7}$$

where σ is a conductivity tensor given by

$$\sigma = \sigma_i(\sigma_i + \sigma_e)^{-1}\sigma_e \tag{2.8}$$

This simplification is motivated by the reduced computational effort required to solve the single PDE of the monodomain model compared with the two coupled PDEs of the bidomain model. However, this gain may be at the cost of reduced accuracy (especially close to the propagating wavefront where V_m is most likely to be non-constant), hence it is important to verify the accuracy of monodomain simulations compared with the equivalent bidomain simulation [98].

2.3 Deformable objects simulations

Since Terzopoulos' pioneering work on simulating deformable objects in the context of computer graphics [99], many deformable models have been proposed. These approaches focus on an accurate material representation, on stability aspects of the dynamic simulation and on versatility in terms of advanced object characteristics that can be handled, e. g. plastic deformation or fracturing.

This inherently interdisciplinary field combines newtonian dynamics, continuum mechanics, numerical computation, differential geometry, vector calculus, approximation theory and Computer Graphics (to name a few) into a vast and powerful toolkit, which is being further explored and extended. The field is in constant flux and, thus, active and fruitful, with many visually stunning achievements to account for.

In this section, some of the main techniques and tools that have been used to visualize, simulate, and animate deformable objects will be mentioned (for detailed summaries please refer to [100, 101]. These techniques can be applied to represent muscle geometry, to encourage muscle contraction, and to deform the shape of tissues when they interact with other elements of a scene.

2.3.1 Finite Difference Method (FDM)

A method that approximates solutions to PDEs that are used to simulate solids is the FDM [102]. A FDM proceeds by replacing the derivatives in the differential equations with finite difference approximations. This gives a large but finite algebraic system of equations to be solved in place of the differential equation.

The principle of the FDM is close to the numerical schemes used to solve differential equations. It consists in approximating the differential operator by replacing the derivatives in the equation using differential quotients. The domain is partitioned in space and in time and approximations of the solution are computed at the space or time points. The error between the numerical solution and the exact solution is determined by the error that is commited by going from a differential operator to a difference operator. This error is called the discretization error or truncation error. The term truncation error reflects the fact that a finite part of a Taylor series is used in the approximation.

For the sake of simplicity, the one-dimensional case only will be considered. The main concept behind any finite difference scheme is related to the definition of the derivative of a smooth function u at a point $x \in \mathbb{R}$:

$$u'(x) = \lim_{h \to 0} \frac{u(x+h) - u(x)}{h} \tag{2.9}$$

and to the fact that when h tends to 0 (without vanishing), the quotient on the right-hand side provides a "good" approximation of the derivative. In other words, h should be sufficiently small to get a good approximation. It remains to indicate what exactly is a good approximation, in what sense. Actually, the approximation is good when the error commited in this approximation (i.e. when replacing the derivative by the differential quotient) tends towards zero when h tends to zero.

The FDM is well suited for problems with simple geometry, and was widely used before the invention of a more efficient, robust method: the Finite Element Method (FEM).

2.3.2 Finite Element Method

One of the most important advances in the field of numerical methods was the development of the FEM. In the FEM, a continuum with a complicated shape is divided into finite elements (ie 3D hexes, or tetrahedrons in 3D, quadrilaterals or triangles in 2D). The individual elements are connected together by a topological map called a mesh. The FEM is a robust and thoroughly developed method, and hence it is widely used in engineering fields due to its versatility for complex geometry and flexibility for many types of linear and non-linear problems. Most practical engineering problems related to solids and structures are currently solved using well developed FEM packages that are commercially available.

The displacements and positions of an element are approximated from discrete values using interpolation functions:

$$\Phi(x) \approx \sum_i h_i(x)\Phi_i \tag{2.10}$$

Where h_i is the interpolation function for the x element and Φ_i is a scalar weight associated with h_i. There are many options for element type and interpolation functions. The choice depends on the geometry of the objects, the desired precision, and the computing power available. Larger order interpolation functions, and more complex elements require more processing time for each element, but can generate more accurate approximations [31]. Given a dynamic problem to be solved, the equilibrium equations are derived in terms of quantities of interest (ie, stress or stress) and are expressed as PDE. These PDEs are subsequently approximated by the FEM model. The result are algebraic equations representing linear or non-linear systems. While small linear systems can be solved with simple methods (eg, Gaussian elimination), large or non-linear systems require iterative methods (eg, Newton's method) [103].

There are many algorithms for the implementation of the FEM, but they all contain the basic steps: Preparation of input, the formation of global matrices of stiffness and forces, and the solution of the system of equations. The overall computational complexity of the method, as obtained by Farmaga et al. [104], is $O(NW^2)$, where N is the number of nodes, and W is the stiffness matrix bandwidth.

However, the FEM shares some limitations of numerical methods that rely on meshes or elements that are connected together in a predefined manner [38]:

1. **High cost in creating a FEM mesh:** The creation of a mesh for a problem domain is a prerequisite in using any FEM code and package. Usually the analyst has to spend most of the time in such a mesh creation, and it becomes the major component of the cost of a computer aided design (CAD) project. Since operator costs now outweigh the cost of CPU time of the computer, it is desirable that the meshing process can be fully performed by the computer without human intervention. This is not always possible without compromising the quality of the mesh for the FEM analysis, especially for problems of complex three-dimensional domains.

2. **Difficulty in adaptive analysis:** One of the current new demands on FEM analysis is to ensure the accuracy of the solution. To achieve this purpose, a so-called adaptive analysis must be performed. In an adaptive analysis using FEM, re-meshing (re-zoning) is required to ensure proper connectivity. For this re-meshing purpose, complex, robust and adaptive mesh generation processors have to be developed. These processors are limited to two-dimensional problems. Technical difficulties have precluded the automatic creation of hexahedron meshes for arbitrary three-dimensional domains. In addition, for three-dimensional problems, the computational cost of re-meshing at each step is very expensive, even if an adaptive scheme were available. Moreover, an adaptive analysis requires "mappings" of field variables between meshes in successive stages of the analysis. This mapping process can often lead to additional computation as well as a degradation of accuracy in the solution.

3. **Limitation in the analyses of some problems:** Under large deformations, considerable loss in accuracy in FEM results can arise from the element distortions: it is difficult to simulate crack growth with arbitrary and complex paths which do not coincide with

the original element interfaces; it is very difficult to simulate the breakage of material with large number of fragments; the FEM is based on continuum mechanics, in which the elements cannot be broken; an element must either stay as a whole, or disappear completely. This usually leads to a misrepresentation of the breakage path.

The root of these problems is the use of elements or mesh in the formulation stage. The idea of getting rid of the elements and meshes in the process of numerical treatments has naturally evolved, and the concepts of meshfree or meshless methods have been shaped up.

2.4 Meshfree methods

A Meshfree method is a numerical method used to approximate the solution to differential equations without the use of a predefined mesh for the domain discretization [38]. Meshfree methods use a set of nodes scattered within the problem domain as well as sets of nodes scattered on the boundaries of the domain to represent (not discretize) the problem domain and its boundaries. These sets of scattered nodes are called field nodes, and they do not form a mesh, meaning it does not require any a priori information on the relationship between the nodes for the interpolation or approximation of the unknown functions of field variables.

When compared to the FEM, there are several key differences:

1. Meshfree methods do not require a mesh. The problem domain and its boundaries are fist modelled and represented by using a set of nodes scattered within. Since these nodes carry the values of the field variables, they are often called field nodes. The density of the nodes depends on the accuracy required and resources available. This allows the meshfree formulations to be more accurate than FEM.

2. Since there is no mesh of elements in an meshfree method, the field variable (e.g., a component of the displacement) u at any point at $\mathbf{x}$ within the problem domain is interpolated using function values at field nodes within a small local support domain of the point at x, i.e.,

$$u(\mathbf{x}) = \sum_{i=1}^{n} \phi_i(\mathbf{x}) u_i \tag{2.11}$$

where n is the number of nodes that are included in the local support domain of the point at $\mathbf{x}$, u_i is the nodal field variable at the ith node, and $\phi_i(\mathbf{x})$ is the shape function of the ith node determined using these nodes.

2.4.1 Meshless deformations based on Shape Matching

A relatively new approach for simulating deformable objects, called Shape Matching, was presented by Müller et al. [105]. The underlying model is geometrically motivated, handles point-based objects, and does not need connectivity information. Additionally, no pre-processing is needed, it is simple to compute, and provides unconditionally stable dynamic simulations.

The main idea of the deformable model is to replace energies by geometric constraints and forces by distances of current positions to goal positions. These goal positions are determined via a generalized shape matching of an undeformed rest state with the current deformed state of the point cloud. Since points are always drawn towards well-defined locations, the overshooting problem of explicit integration schemes is eliminated. The versatility of the approach in terms of object representations that can be handled, the efficiency in terms of memory and computational complexity, and the unconditional stability of the dynamic simulation make the approach particularly interesting for video games or real-time simulations.

The basic idea behind is simple: all that is needed as input is a set of particles with masses m_i and an initial configuration (i. e. the initial positions $\mathbf{x}_i^0$ of the particles). No connectivity information or mesh is needed. The particles are simulated as a simple particle system without particle-particle interactions, but including response to collisions with the environment and including external forces such as gravity. After each time step, each particle is pulled towards its goal position $\mathbf{g}_i$. To compute the individual goal positions, the original configuration (or shape) defined by the $\mathbf{x}_i^0$ with the actual configuration defined by the actual positions of the particles $\mathbf{x}_i$ (see Figure 2.7) is matched.

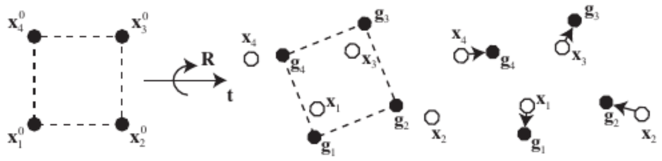


Figure 2.7: First, the original shape $\mathbf{x}_i^0$ is matched to the deformed shape $\mathbf{x}_i$. Then, the deformed points $\mathbf{x}_i$ are pulled towards the matched shape g_i.[105].

2.4.1.1 Shape Matching Extensions

The method can be extended to simulate several additional properties. Some of them are:

- **Rigid Body Dynamics** The method can be used to imitate a rigid body simulator by setting $\alpha = 1$. In this case, the points are moved to the goal positions $\mathbf{g}_i$ exactly at each time step. These positions represent a rotated and translated version of the initial shape.

Given an arbitrary surface mesh, only a small subset of the vertices need to be animated as particles.

- **Linear Deformations** The method described so far can only simulate small deviations from the rigid shape. To extend the range of motion, the linear transformation matrix $\mathbf{A}$ is used. This matrix describes the best linear transformation of the initial shape to match the actual shape in the least squares sense.
- **Volume conservation** To make sure that volume is conserved, $\mathbf{A}$ is divided by $\sqrt[3]{det(\mathbf{A})}$ ensuring that $det(\mathbf{A}) = 1$. For the standard approach, only $\mathbf{A}_{pq}$ is computed. For extended approaches, the matrix $\mathbf{A}_{qq} = (\sum_i m_i \mathbf{q}_i \mathbf{q}_i^T)^{-1}$ is also needed.
- **Quadratic Deformations** Linear transformations can only represent shear and stretch. To extend the range of motion by twist and bending modes, quadratic transformations are used instead of linear transformations.

2.4.2 Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) [53] is a meshfree method originally used to simulate astrophysical phenomena. A remarkable feature of this method is that its computational structure involves a large number of common abstractions, while at the same time it is distinguished by the fact that it is inherently linked to the physics of the simulated systems.

The movement of nodes, or particles, is based on the efficient calculation of the forces and pressures each particle experiences every time step. Each particle has an associated spatial distance h (the smoothing distance, or core radius) over which the forces that act upon it are estimated by a smoothing kernel function. The capability of SPH to deal with the issues of varying density and unbounded flows stemmed from its meshfree nature, since the central idea of this method is to follow moving particles, free of any mesh/grid constraints.

The success of SPH is due to the relative ease with which SPH simulations have been able to produce results for cases involving complicated nonlinear and often multi-phase phenomena. With little modification of the basic methodology, SPH has been able to generate results in close agreement with reference solutions/data in validation tests, without highly sophisticated algorithms required in mesh-based schemes. Most of the aforementioned fields are deemed too difficult (not to say impossible) for other numerical methods. For these reasons, in complex free-surface flow modelling, SPH has challenged the dominance of volume-of-fluids (VOF), level set (LS) or other promising approaches dedicated to these special kinds of flows.

Moreover, recent progress in the numerical features of SPH has increased its credibility and made it increasingly attractive to mathematicians, so far exclusively concerned with more traditional, well-established techniques like the FEM and FVM. An important issue with these approaches, which approximate the body by a mesh of nodes of fixed topology, is that they are not adapted to animate substances able to undergo large inelastic deformations [106].

In many situations, SPH has recently proven to be just as efficient as FEM and FVM [45, 57, 107–109]. However, the SPH method is inappropriate for certain applications and is known to provide poor predictions for some phenomena, such as long-distance water wave propagation [110].

As a relatively young computational method, SPH has some disadvantages [110], including:

- Large computational time, particularly in 3-D simulations. This can be overcome by using parallel computing, specially on the Graphics Processing Unit (GPU).
- Difficulties in prescribing wall boundary conditions, and even greater problems at open (inflow/outflow) boundaries.
- Lack of a consistent theory in relation to the mathematical foundation of the method (convergence, stability).
- Inaccuracy of pressure prediction, at least for the original WCSPH variant.
- Difficulties in dealing with variable space resolution for (nearly) incompressible flows.

2.4.3 SPH approximation techniques

The conventional SPH method was originally developed for hydrodynamics problems in which the governing equations are in strong form of PDEs of field variables such as density, velocity, energy, etc. There are basically two steps in obtaining a SPH formulation. The first step is to represent a function and/or its derivatives in continuous form as integral representation, and this step is usually termed as kernel approximation. In this kernel approximation step, the approximation of a function and its derivatives are based on the evaluation of the smoothing kernel function and its derivatives. The second step is usually referred to as particle approximation. In this step, the computational domain is first discretized by representing the domain with a set of initial distribution of particles representing the initial settings of the problem. After discretization, field variables on a particle are approximated by a summation of the values over the nearest neighbor particles. For an in depth introduction to SPH refer to [53, 54, 111].

2.4.4 SPH method description

With SPH, field quantities that are only defined at discrete particle locations can be evaluated anywhere in space. For this purpose, SPH distributes quantities in a local neighborhood of each particle using radial symmetrical smoothing kernels. According to SPH [112], a scalar quantity A is interpolated at location r by a weighted sum of contributions from all particles:

$$A_S(r) = \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h) \tag{2.12}$$

where j iterates over all other particles, m_j is the mass of particle j, r_j its position, ρ_j the density, and A_j the field quantity at r_j. The function $W(\mathbf{r}, h)$ is called the smoothing kernel with core radius h. Smoothing Kernels will be further explained in Section 2.4.4.1.

The particle mass and density appear in Eqn. 2.12 because each particle i represents a certain volume $Vi = m_i/\rho_i$. While the mass m_i is constant throughout the simulation, the density ρ_i varies and needs to be evaluated at every time step. SPH is also be used to calculate the density: through substitution of ρ into A_S in Eqn. 2.12 we get, for the density at location r:

$$\rho_S(r) = \sum_j m_j \frac{\rho_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h) = \sum_j m_j W(\mathbf{r} - \mathbf{r}_j, h) \tag{2.13}$$

With the SPH approach, derivatives only affect the smoothing kernels. The gradient of A is simply

$$\nabla A_S(r) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(\mathbf{r} - \mathbf{r}_j, h) \tag{2.14}$$

while the Laplacian of A evaluates to

$$\nabla^2 A_S(r) = \sum_j m_j \frac{A_j}{\rho_j} \nabla^2 W(\mathbf{r} - \mathbf{r}_j, h) \tag{2.15}$$

2.4.4.1 Smoothing kernels

One of the central issues for the mesh-free methods is how to effectively construct a proper shape function using only nodes scattered in an arbitrary manner without using a predefined mesh that provides the connectivity of the nodes. For the SPH method, the smoothing kernel is of utmost importance since it not only determines the pattern to interpolate, but also defines the width of the influencing area of a particle. To choose or construct a suitable kernel for a given problem, the following properties [113, 114] are required:

- The smoothing function must be normalized:

$$\int W(\mathbf{r},h)\mathrm{d}\mathbf{r} = 1 \tag{2.16}$$

  where $W(\mathbf{r},h)$ is the smoothing kernel, $\mathbf{r}$ is the position vector, h is the smoothing length that determines the supporting area of the smoothing kernel.

- The smoothing function should have the compact support property, which is defined by the smoothing length h and a scale factor k that determines the spread of the specified smoothing kernel. The compact support property can be defined as:

$$\begin{aligned} W(\mathbf{r}-\mathbf{r}_j) &= 0 \quad \text{for} \quad |\mathbf{r}-\mathbf{r}_j| > kh \\ W(\mathbf{r}-\mathbf{r}_j) &\geq 0 \quad \text{for} \quad |\mathbf{r}-\mathbf{r}_j| \leq kh \end{aligned} \tag{2.17}$$

- The kernel function should tend to the Dirac delta function as the smoothing length tends to zero:

$$\lim_{h\to 0} W(\mathbf{r},h) = \delta(\mathbf{r}) \tag{2.18}$$

- The function should be even (symmetric):

$$W(\mathbf{r},h) = W(-\mathbf{r},h) \tag{2.19}$$

The use of different kernels is the SPH analogue of the use of different difference schemes in FDM [53]. The advantage of SPH is that the kernel can be calculated in a subroutine, or a table, and a code can be changed from one kernel to another. Typical kernels are the B-splines [115] and the Wendland [116, 117] kernels.

The kernels based on B-spline functions have some advantages: they have compact support; the second derivatives are continuous, and the dominant error term in the integral interpolant is $O(h^2)$. The compact support means that interactions are exactly zero for $r > 2h$; the continuity of the second derivative means that the kernel is not sensitive to disorder and the errors in approximating the integral interpolants by summation interpolants are small provided the particle disorder is not too large. The cubic spline kernel is defined as:

$$W(\mathbf{r},h) = \frac{\sigma}{h^v}\begin{cases} 1-\frac{3}{2}q^2+\frac{3}{4}q^3 & 0 \leq q < 1 \\ \frac{1}{4}(2-q)^3 & 1 \leq q \leq 2 \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

where v is the number of dimensions, q is defined as $\frac{r}{h}$, and σ is a normalization constant with the values $\frac{2}{3}, \frac{10}{7\pi}, \frac{1}{\pi}$ in one, two, and three dimensions respectively.

The Wendland kernel has continuous and smooth first second derivatives, and is defined as:

$$W(\mathbf{r}, h) = \frac{\alpha_d}{h^d}(1 - \frac{1}{2}q)^4(2q + 1) \quad 0 \leq q \leq 2 \tag{2.21}$$

where $q = \frac{r}{h}$, d is the spatial dimension ($d = 2, 3$), and α_d is a normalization constant ($\alpha_d = \frac{7}{4\pi}$ for $d = 2$, and $\frac{21}{16\pi}$ for $d = 3$).

2.4.5 SPH neighbor search optimization

Since the smoothing kernels used in SPH have finite support h, a common way to reduce the computational complexity [112] is to use a grid of cells of size h. Then potentially interacting partners of a particle i only need to be searched in i's own cell and all the neighboring cells. This technique reduces the time complexity of the force computation step from $O(n^2)$ to $O(nm)$, m being the average number of particles per grid cell.

2.5 Parallel and concurrent programming

For several years the interest in parallel and concurrent computation has been increasing. With the paradigm shift of CPU manufacturers to only increase the clock speed to add multiple processing cores to their chips (multi-core processors), and with the introduction of the use of programmable video cards (Graphical Processing Units, GPUs), application and systems developers have been forced to adapt the [118] application development paradigms in order to take full advantage of the resources at their disposal.

2.5.1 Concurrence and parallelism

At its simplest level, when talking about concurrency, it refers to two or more activities that are happening at the same time. Concurrence can be found as a natural part of life: you can walk on a street and perform actions with your hands, for example. Concurrence in systems development refers to a system performing different activities at the same time, rather than sequentially (one after another); a concurrent system can be said to be parallel if more than one task is physically active, ie with more than one [119] processor.

This is not a new concept: multitasking operating systems allow a computer to run multiple applications at the same time by scheduling task execution; as the task changes are so fast, it can not be said at what point one task was suspended and the processor changed to another. For several years now, computers with multi-core processors are becoming the standard. Unlike

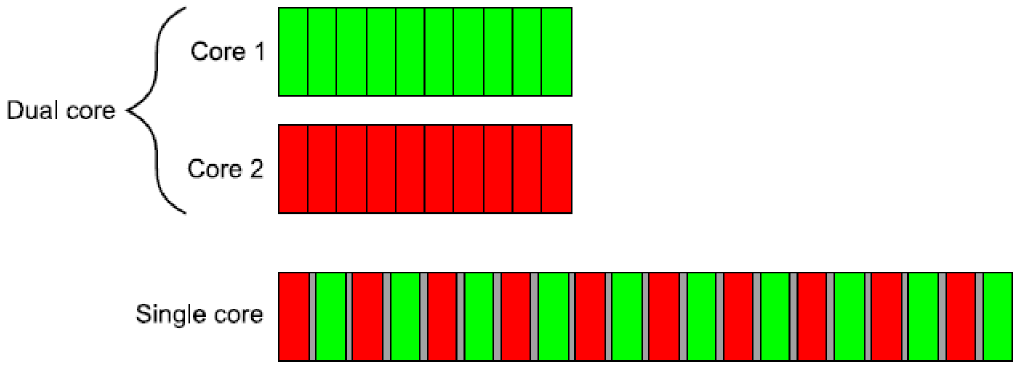


FIGURE 2.8: Parallel execution on a dual core CPU against scheduling tasks on a single core CPU. [119].

single-core processors, on a computer with a multi-core processor each task can be run at its own core. This has the name of *hardware concurrency* [119]. In the Figure 2.8 you can see an idealized scenario of a computer with two tasks, and how tasks are divided into a single core and a multicore processor.

Each core has a certain number of execution threads. A thread can be viewed as a process that runs independently of others, and executes a different sequence of instructions. Considering this, there are two approaches to using concurrency in an application: concurrency with multiple processes, and concurrency with multiple threads [119].

- **Concurrence with multiple processes**: This approach seeks to divide the tasks into several execution threads that are processed at the same time. An example would be to run a word processor and an Internet browser at the same time.
- **Concurrence with multiple threads**: This approach seeks to execute multiple threads for a single process, and to divide the instructions of that process into the different threads. All threads share the same memory space, and many of the data can be accessed directly from all threads. An example would be to divide the calculation of an IP approximation into several threads of execution.

Having defined what the concurrence is, you can glimpse the main objectives of parallel and concurrent computing: improve the processing times of software applications, and make a separation of tasks in applications[119].

- **Improve processing times**: Multiprocessor systems have been around for decades (on supercomputers, mainframes, and servers). However, with the current prevalence of multi-core processors in personal computers, the computing power of newer systems is perceived by running many tasks in parallel. To improve processing times, developers have several ways to take advantage of the different cores and threads available: either by separating a process into several parts, and running them in parallel, or by using the threads to solve bigger problems; for example, instead of processing one file at a time, multiple files can be processed in each thread. This is known as *data parallelism*.

- **Separation of tasks**: This refers to the separation of code that does a certain task in different threads of execution. It can be seen mainly when separating different functionalities of a system, although these are executed at the same time. For example, you can see the separation of tasks in a video game. One thread takes care of the graphics engine, another run the sounds, and another of the processing of actions within the game. This is known as *task parallelism*.

2.5.2 Compute using GPUs

Until a few years ago, computers contained only one processor designed to perform general tasks. Since the last decade, it has been given importance to the use of other elements of processing, being the most prevalent the GPU. These were originally designed to perform specialized tasks of computer graphics in parallel. However, GPUs have become more powerful and widespread, allowing for general-purpose parallel computing to be realized, resulting in considerable improvements in performance and energy efficiency [120]. This concept is also known as GPGPU, General Purpose Graphical Processing Unit.

Compute using GPU looks to use both GPUs and CPUs of a computer to improve the performance of different applications by using data parallelism. Unlike multi-core CPUs, GPUs can count thousands of processing cores, and each core can run hundreds of threads of execution. Despite this difference, the GPUs are not intended to replace the CPU; each has certain advantages for certain types of programs. The CPU is good for processing intensive control tasks, while the GPU is good for intensive data parallelization tasks, where many calculations with parallel data are required. Therefore, to obtain optimum performance when running an application, you have to use both CPU and GPU, leaving the sequential code to the CPU and the code parallel to the GPU, allowing the characteristics of both to complement each other (see Figure 2.9). To support the overall execution of the CPU and GPU, NVIDIA designed a programming model called CUDA [120].

2.5.2.1 CUDA

CUDA, which stands for Compute Unified Device Architecture, is a general-purpose parallel computing platform and programming model that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way. Using

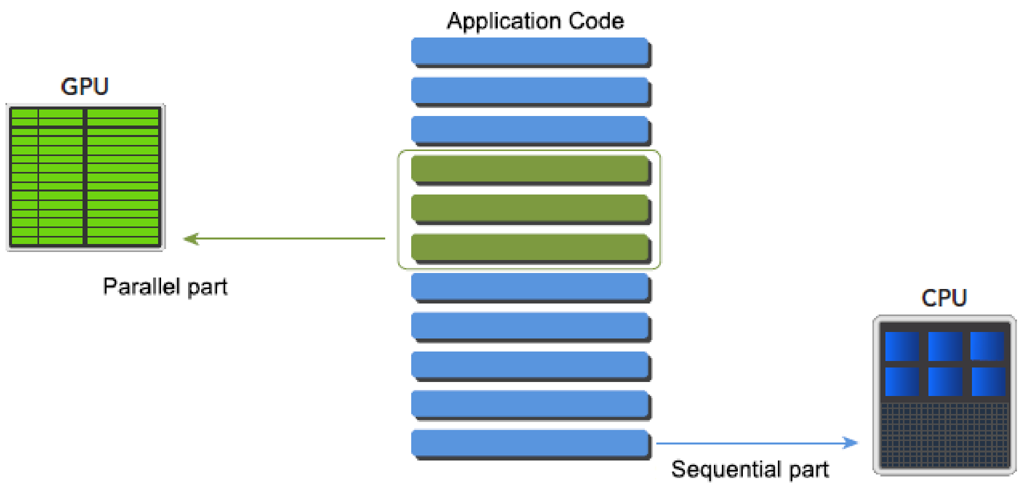


FIGURE 2.9: Running different parts of a code in parallel on a GPU, and in sequence on a CPU. [120].

CUDA, a programmer may access the GPU for computation, as is usually done for the CPU. The CUDA platform is accessible through CUDA-accelerated libraries, compiler directives, application programming interfaces, and extensions to industry-standard programming languages, including C, C++, Fortran, and Python [120].

CUDA provides two Application Program Interfaces (API) for handling the GPU, and the organization of the cores and their respective threads. The *driver* API is a low-level API, relatively difficult to program, but provides a lot of control over the devices used. The *runtime* API is a high-level API, implemented over the API driver, which provides relatively simple tools and directives for GPU programming. The runtime is the most used by the application developers.

A CUDA C program consists of a mixture of two parts: host code that runs on the CPU, and device code that runs on the GPU. The NVIDIA CUDA compiler, *nvcc*, separates the host code from the device code at compile time. The host code is standard C code and compiled with C compilers, in RAM. The device code is written using CUDA C with specific keywords to define the parallel functions, called *kernels*. The device code is compiled using nvcc, using video memory. In the bind process, CUDA runtime libraries are added to handle calls from *kernels* (it does not have to do with the operating system kernel) and explicit manipulation of the GPU. This can be seen in the Figure 2.10.

When a kernel function is launched from the host side, execution is moved to a device where a large number of threads are generated and each thread executes the statements specified by the kernel function. CUDA exposes a two-level thread hierarchy decomposed into blocks of threads and grids of blocks, which can be seen in Figure 2.11.

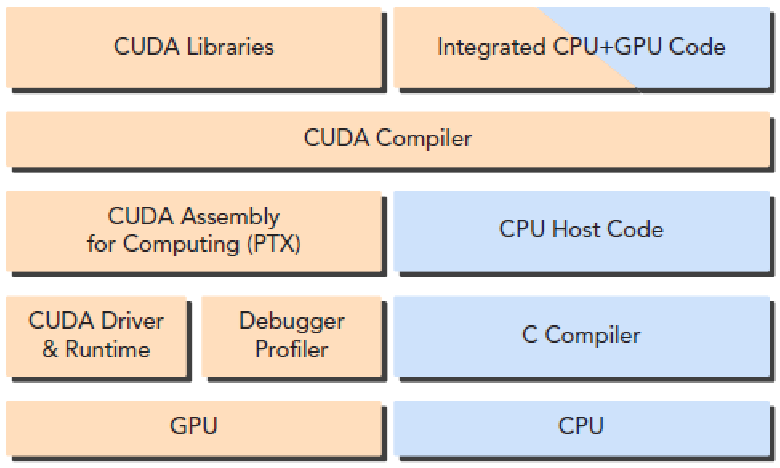


FIGURE 2.10: CUDA C code compilation. [120].

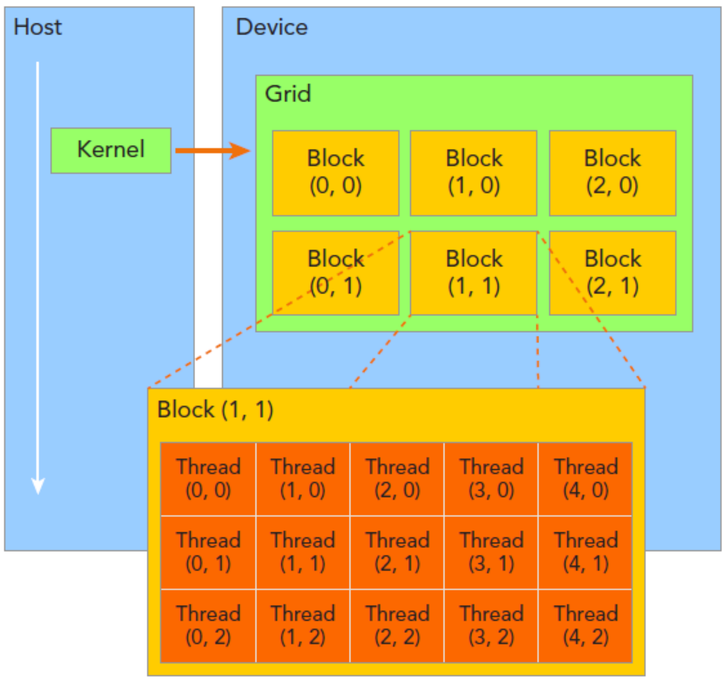


FIGURE 2.11: CUDA Thread hierarchy.

All threads spawned by a single kernel launch are collectively called a grid. All threads in a grid share the same global memory space. A grid is made up of many thread blocks. A thread block is a group of threads that can cooperate with each other using block-local synchronization and block-local shared memory. CUDA employs a Single Instruction Multiple Thread (SIMT) architecture to manage and execute threads in groups of 32 called warps. All threads in a warp execute the same instruction at the same time. These characteristics make it possible to process a problem with a GPU by dividing it into blocks and threads, as well as by using the different memory models to improve performance. Refer to [120–122] for additional details on CUDA and its programming model.

Chapter 3

Simulation and control of skeletal muscles

In this chapter the different approaches and techniques that have been used to tackle the problem of simulation and control of the muscles of the human body are discussed. Based on its fundamental methodology, the different approaches were classified into the following categories: muscle deformation, control and simulation, and fiber-based simulation. Additionally, a review of the work that has been done with the bidomain model, was presented. Finally, the use of the SPH method for the simulation of biological tissue was reviewed. Here, works that model solids, biological tissue, or electromagnetic problems using the SPH method, were considered. Special attention was given to simulations that use GPGPU for computing.

3.1 Muscle deformation

The muscles provide the physiological functions that generate the movement of the body and give it shape, making them a key component in the animation and modeling of human figures. A realistic deformation of the muscles was necessary to have high quality animated humanoid characters. There were many works that develop mathematical, physical, and computational models for the simulation of muscles (from a muscle to complete bodies) with the purpose of increasing their realism and accuracy in a wide range of applications: video games and movies, virtual and augmented reality, telepresence, medicine, biomechanics, among others.

Depending on different performance and visualization requirements, each of the works takes a different approach with respect to the modeling of their simulations. For example, visual realism was desirable in film productions, or video games, so there was a greater focus on graphing and visualizing the muscles; while anatomical or biomechanical accuracy was preferable in medical applications, so focusing on its proper functioning was the priority.

Despite the amount of work, the complexity of the human body and its muscles, make its modeling a challenge. In this part of this chapter, emphasis was placed on different works that have attacked the problem of muscle deformation with different approaches: geometric approaches, physics-based approaches, and data-based approaches.

3.1.1 Geometric approaches

Geometric approaches were used in the first simulations performed because they were practical and efficient. Most of the work done focused on modeling the effects of muscle contraction animation, such as bulging or swelling, which can be fundamental factors for skin deformation, or for facial animations. These were successful in modeling simple muscles (for example, spindle muscles) but there may not be a direct extension to the complex muscles [123, 124]. In addition, as the deformation of the muscles was determined by the arrangement of the bones, these techniques had problems achieving a high level of realism from physiological or biomechanical perspectives. Because of this, in order to better manage these problems, the muscles were constructed in several layers, or physical approaches had to be applied.

Chadwick et al. [125] used Free Form Deformations (FFD)s to represent muscle deformation. An articulated skeleton transformed a lattice of FFDs, which in turn represented the change of shape of a muscle. Although FFDs provided a simple form of control, they did not allow direct manipulation, and they did not allow producing more complex forms. Moccozet et al. [126] addressed this limitation by introducing Dirichlet Free Form Deformations (DFFDs), which were based on a dispersed data interpolation technique. They removed the requirement of regularly spaced control points by replacing local rectangular coordinates with natural neighboring coordinates (ie, Sibson coordinates). Given a point, its natural neighbors were collected based on Delaunay and Dirichlet / Voronoi diagrams, and its displacement was calculated using interpolation. Authors used a multi-layered deformation model to generate hand animations where the muscle layer was modeled by a set of DFFD control points that corresponds to a simplified topography of a hand.

Komatsu [127] used Bezier surfaces to model the deformation of the body. The surfaces were patched cylindrically around a skeleton, and were controlled together to transform the body. Wilhelms [128] and Scheepers et al. [123] used parametric ellipsoids as the basic primitive to model the muscular bellies of the skeletal muscles of the human body. They adjusted three main axes to represent the bulge of the muscles, while the volume was preserved using predefined relationships between those axes. Although an ellipsoid was sufficient to model simple shapes, such as a fusiform muscle, it could not be easily adapted to model more complex shapes. The

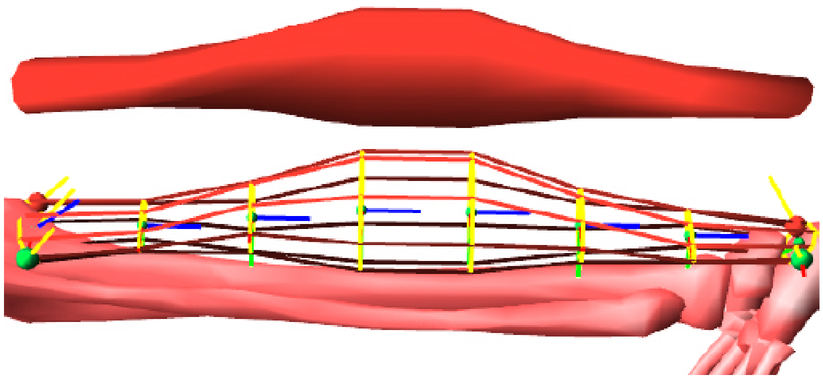


FIGURE 3.1: The shape of the muscle was defined by the control of the cross sections of the cylinder. The upper part shows the muscle with a skin, while the lower part shows the deformed cylinder. [124].

work of Scheepers et al. was distinguished since it extended the basic model of ellipsoids to represent muscles with multiple bellies (for example, the pectoralis) where n pairs of points of origin and insertion were specified, and n ellipsoids were laterally aligned along each pair. That model was used to represent more complex muscles that were bent or wrapped around a structure (for example, brachioradialis in the forearm). The direct path between the points of origin and insertion was replaced by a Bezier curve that represents the direction of muscle strength, and by ellipses of varying sizes along that curve to define the volume and shape of the muscle. Wilhelms and Gelder [124] presented a work where cylinders with transversal cuts were used to represent the muscles. Each cross section was modeled using B-Spline curves and its radius was controlled to express the volumetric changes in the muscle. Cylinders can also be bent to model muscles that bend at the joints of the body. In addition, the length, width, and thickness of the muscle were scaled to maintain a constant volume. This form of modeling can be seen in Figure 3.1.

Ramos and Larboulette [129] presented a method to deform the skin of characters using the underlying muscles. To simulate the muscles, authors used parametric curves to generate the different forms of the muscles. The general shape of your muscle model was a generalized cylinder defined by the sweep of a thickness curve $C_T(t)$ along a sweeping curve $C_S(t)$. $C_S(t)$ was a three-dimensional Bezier curve that represented the profile of the muscle. That curve was defined by two points: the origin $\mathbf{O}$ and the insertion $\mathbf{I}$ of the muscle. $C_S(t)$ was a function that represents the thickness of the muscle as a function of t. In this way, each muscle section in t was defined by an ellipse that was along $C_S(t)$ whose thickness was given by $C_T(t)$. Each of Bezier's curves $C_S(t)$ and $C_T(t)$ were composed of two Bezier curves called *Origin segment* and *Insertion segment*. Each of these curves had four control points $p_j, j \in [0, 3]$.

An important part of this muscle model was that it considered the tension generated by isometric and isotonic contractions. In the case of isometric contractions, the bones did not move and the muscle length does not change. However, its form did change. When the tension increased, the muscle bulged in one direction while tapering perpendicularly. In an isotonic contraction, the bones moved but the tension in the muscle remained constant. The change in shape occured only by shortening or elongation of the muscle. If it shortens, the muscular belly bulged; it decreased in the opposite case. Disadvantages of this model were that it was not considered an interaction between muscles or between muscles and bones, and only the most superficial muscles were modeled.

Bloomenthal et al. [130] used convolution surfaces to model a human hand and arm, by bringing bones, muscles, tendons, and veins closer to the underlying skeleton. Thalmann et al. [131] presented a model of a human based on multiple layers, where the primitives of the body (ie, muscles, extremities, tissues, etc.) were constructed from a skeleton of lines that was covered with ellipsoidal surfaces of metaballs. Although the implied surfaces were soft and continuous when modeling objects, undesirable mixing effects could occur when modeling deformations in body joints.

3.1.2 Approaches based on physics

Although geometric models had proven to be sufficient for some graphic applications that demand acceptable visual quality, their simplicity and the need for human intervention to configure models, made it difficult to extend them to represent complex scenes involving dynamic behaviour. Additionally, they did not have the physical or mechanical precision that was required to generate more realistic models, animations, and simulations. For these reasons, many researchers have worked on models where complicated problems such as the dynamics of muscle and tissue properties were solved with physics. To model physics-based muscles, the following problems have to be considered: 1) determining the contractile forces of the muscles, and 2) representing the changing geometry of the muscles during a contraction. To solve these problems, several muscle models have been generated based on a variety of computational methods, such as mass-spring systems, FEM, and FVM.

3.1.2.1 Mass-spring systems

In the case of mass-spring systems, an object was modeled as a collection of mass points linked together by massless springs. Chadwick et al. [125], linked FFD control points to mass points in a mass-spring system, allowing the dynamic system to influence deformations based on geometry. By increasing the muscle model based on FFD with the mass-spring system, authors were able to represent visco-elastic properties that simulations based on articulated skeletons did not normally have. Lee et al. [6] and Albrecht et al. [2] inserted a muscular layer based on a mass-spring system between the surface of the skin and the skeletal structure to be able to model facial

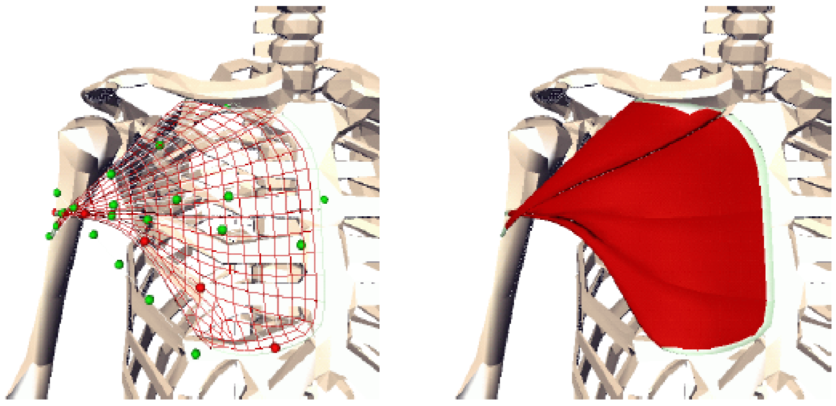


Figure 3.2: Use of mass-spring system for muscle simulation. The behavior of the lines of action on the related surfaces of a pectoral muscle was simulated.[133].

expressions and hands, respectively. The spring forces generated by the movement of skeletal bones caused the attached skin surface to deform more realistically. Nedel and Thalmann [132], similarly to Aubel and Thalman [133], proposed a muscle model consisting of a line of action and the muscular surface. The action line was modeled using either a straight line [132] or a one-dimensional mass-spring [133] to define the profile of a muscle (that is, its orientation and the points of attachment to the bone). The skeleton kinematically controls the lines of action to deform the adjacent muscular surfaces, which in turn were based on mass-spring systems. In Figure 3.2 an example of this behavior can be seen.

In addition to linear springs that represent a surface, angular springs have been incorporated to control the volume of the muscles [132]. Ng-Thow-Hing [134] proposed a more sophisticated model that was based on anatomical and biomechanical considerations. Their solid muscles were extracted from medical imaging data or sectional images, and were modeled using volumetric B-Splines. For the interior of the muscles, a muscle fiber architecture was built, which was based on data from digitally scanned fibers; in this case, a system based on the Hill model [10] to express the dynamics of the muscle fibers, and a mass-spring system to represent the viscoelastic deformations of the muscle. Zordan et al. [9] developed a human torso that was capable of animating breathing movements, such as inhaling and exhaling. The interaction between the muscles of the thoracic cage, the diaphragm, and those of the abdomen at the time of breathing, was developed based on the mechanics of breathing and simulated using a mass-spring system; this can be seen in the Figure 3.3. In order to preserve the volume of the human body in the simulation, pressure forces were incorporated based on anticipated volume changes.

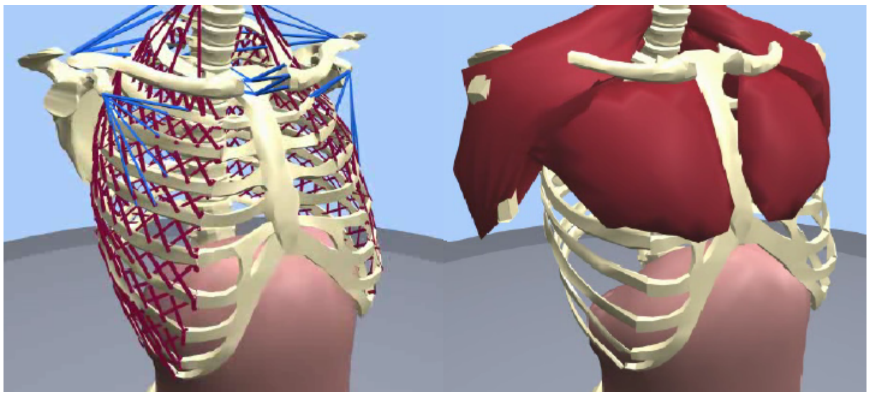


FIGURE 3.3: Use of mass-spring system for breathing simulation. The lines of action and the surfaces related to a torso that simulates breathing are shown. [9].

3.1.2.2 Simulations with the FEM

Chen and Zeltzer [135] proposed a biomechanical approach by integrating Hill's muscle model with a solid elastic model. The active forces of the muscles were approximated as parametric functions and inserted at specific edges between the vertices of a solid based on FEM. While muscle flexion was encouraged, the biomechanical validity of the model was emphasized when compared to experimental measurements, such as strength length, and rapid release properties. Zhu et al. [136] used the Stern muscle model [137] where simplified bone-union-muscle behaviors were described. Both works were computationally efficient, but they were only valid for infinitesimal deformations. Conversely, Hirota et al. [138] and Lemos et al. [139] adopted models of nonlinear materials that allowed them to simulate a robust deformation of large deformations, and they express the passive response of the tissues during a contact.

In biomechanics, FEM has been also been used for the study of skeletal muscles. Several muscle models have been proposed to analyze and predict the distribution of muscle tension during contraction. Yucesoy et al. [140] modeled the mechanical behavior of the skeletal muscles as an interaction between an intracellular domain (ie, muscle fibers) and an extracellular matrix (ie, connective tissue). In that way, the geometry of the muscles was represented as two separate meshes that were elastically linked to take into account the force transmissions between those two domains. Because most models of skeletal muscles represented the geometry of muscles using line segments, Blemker and Delp [141] developed models of the muscles and their underlying bones (see Figure 3.4, using FEMs of three dimensions, from magnetic resonance imaging (MRI) of a living subject.

Courtecuisse et al. [32] presented several contributions to the field of real-time simulations of soft tissue biomechanics using the FEM. Specifically, soft tissue deformation, contact modelling, simulation of cutting, and haptic rendering, which were all relevant to a variety of medical interventions. All these contributions relied on a co-rotational implicit FEM formulation and

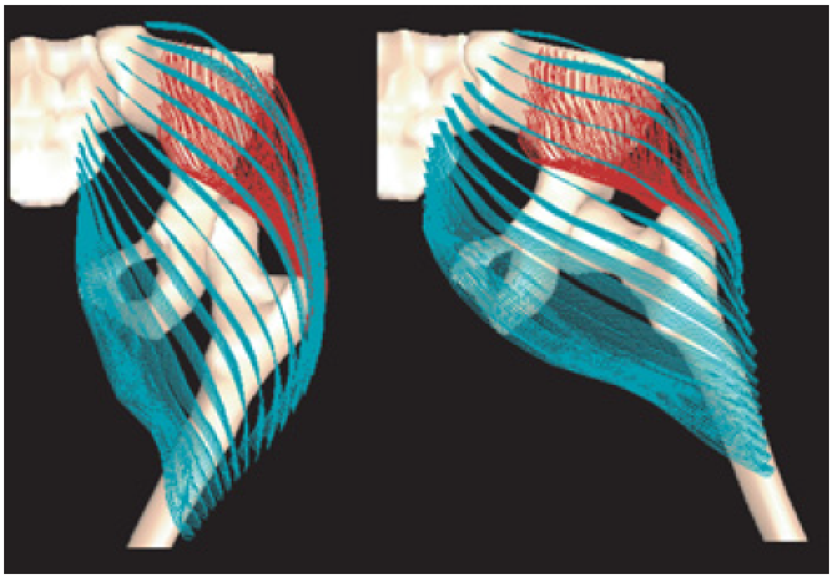


FIGURE 3.4: Simulation based on FEM of the gluteus maximus and gluteus medius during flexion.[141].

efficient GPU parallelizations. The simulations, demonstrated on a patient-specific laparoscopic hepatectomy training system, ran consistently below 35 msec.

The work of Spyrou and Aravas [33] described a constitutive model to generate muscles and tendons. Author's model considered the dependence of the length of the muscle fibers, the levels of activation, and the speed of deformation, on the tension of the muscle fibers. The model was applied to a mesh of FEM muscles, whose geometry was obtained by MRI.

Röhrle et al. [34] presented a biophysically based model, which included several structural and functional characteristics of skeletal muscle. The result was a physiologically based, multi-scale skeletal muscle FEM that was capable of representing detailed, geometrical descriptions of skeletal muscle fibers and their grouping. Together with a well-established model of motor-unit recruitment, the electro-physiological behavior of single muscle fibers within motor units was computed and linked to a continuum-mechanical constitutive law.

In [35], Spyrou and Aravas described a FEM scheme to simulate the movement of human feet, which was able to estimate internal stress levels, as well as changes in the shape of the tissues during movement. They develop a three-dimensional FEM of a foot and a leg, and foot movement was generated based only on the contraction of the plantar flexor muscle. Although the scheme was able to more realistically represent the movement of the foot during a flexion, it had an important limitation: the models of the feet and the leg were based on computed tomography images, which limited the segmentation of soft tissues and prevented the interactions between them from being simulated correctly. In addition, it was not possible to simulate the deformation that those tissues generated. Therefore, the use of MRI was preferred for the development of three-dimensional methods.

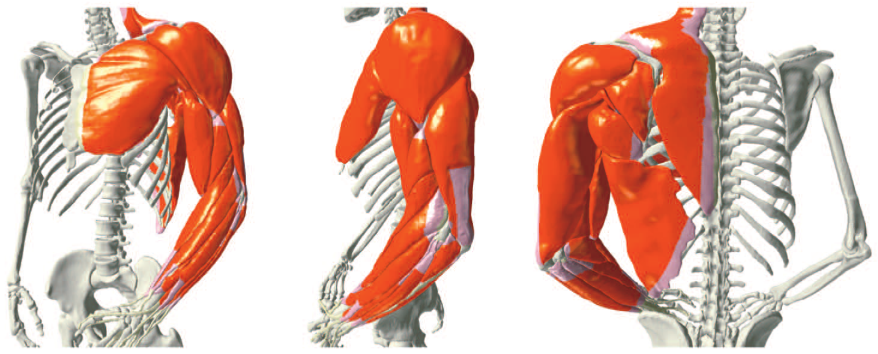


FIGURE 3.5: Simulation based on FVM of the muscular deformation, where around 30 muscles were shown [143].

3.1.2.3 Simulations with the FVM

Teran et al. [142, 143] proposed a FVM-based approach to simulate the deformation of muscles, as can be seen in Figure 3.5. It was argued that using FVM required less computational resources for processing, as well as less memory. To represent a non-linear response of the muscles, authors used a model similar to that of Hirota [138]. In addition, they incorporated anisotropic properties based on a fiber architecture, which were modeled using the B-spline solids technique [134].

3.1.3 Data-based approaches

Instead of developing methods that focus on modeling the physical components and processes of humans, data-based methods focus directly on the shape of the skin, deformed by the underlying muscle of a human who makes certain movements or poses. The data was captured on the surface using MOCAP systems or using different types of sensors and measuring devices. With these data, several techniques were used to generate a new surface to simulate the skin, given a pose of the skeleton. Although these approaches were relatively new, several works have shown the advantage they provide.

Min et al. [144] presented a model based on the fact that the shape of the skin in a human was determined by the underlying skeleton and muscles, and uses an anatomical model based on different layers: skeleton, muscles, and skin. When the skeleton was moved, the isosurface of the muscle was deformed, preserving its volume, which in turn deformed the skin layer. In this work the upper body was modeled, and the resulting animation showed the bending and stretching of an arm. Another approach to arm animation was that of Sloan et al. [145]. Here, several example arm shapes were used, as well as an interpolation scheme based on linear and radial functions to create a continuous range of poses.

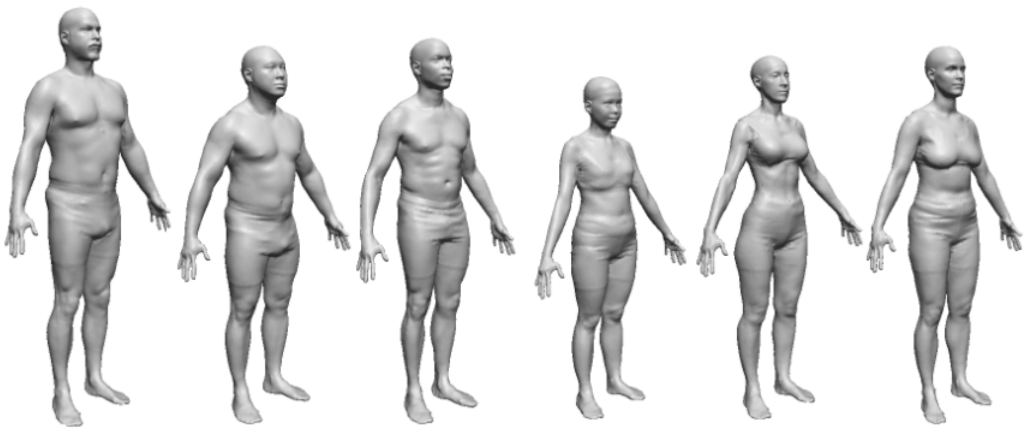


FIGURE 3.6: Simulation of humanoid bodies based on data. [148].

Ma et al. [146] took advantage of the fact that more data sources with human poses began to exist. Using the technique of range scanning, where a person poses for a short time while a scanner creates thousands of data points on the subject with a density of a few millimeters, they were able to create an animation model of the human body. The resulting model allowed the generation of animations in real-time, by manipulating the skeleton while maintaining the level of detail of the surface of the human body. Allen et al. [147] created a high quality model of the upper body, which was capable of generating several poses, based on a range scan. In [148] previous work was extended to include data from the full-body full-range scan data base, CAESAR (Civilian American and European Surface Anthropometry Resource project). Morphing (changing one image or object to another by means of gradual steps) by interpolation between two scans, or adjusting a model to scant MOCAP marker data, were two important results of this technique. The transfer of textures, surface data, or animations between two models was also allowed, in order to correct scanning problems, alter the appearance, or to animate a wide range of characters. In the same way, several extra parameters of a person could be defined, such as their height, or their weight, in order to be considered when modifying a part of a character. An example of the characters that can be created can be seen in Figure 3.6.

Seo and Thalmann [149] presented a similar system based on templates, with additional parameters to generate new human forms that were animatable in real-time. Sand et al. [150] proposed an alternative technique, in which silhouettes of a video were used instead of range scan data to generate a human form that was animatable. Anguelov et al. [151] extended that work, by focusing on representing the muscular deformation that was generated as a result of body movement, in order to perform an animation of the shape of people. A limitation of this system was that the same model of muscle deformation was used for all people generated, so a person who was more muscular may not show as much muscle deformation as he should.

Park and Hodgins [152, 153] further refined the deformation of muscles and skin based on MOCAP data. Authors modeled static deformations as a function of skeleton pose, and dynamic deformations as a function of the acceleration of each part of the body. For this, movements were captured, both slow and fast, of an actor using 350 markers on his body. The two types of deformation were modeled and new animations could be generated from between 40 and 50 markers, in subsequent sessions of MOCAP. Although this approach has the limitations that it was based on the fact that a skeleton generates movement, and that it does not express a movement of the muscles without there being changes in the angles of the joints of the body, high quality animations were produced.

3.2 Control and simulation

While the previous section focused on examining several works related to the deformation of skeletal muscles, this section will review the works that focus on the control of muscles, in order to generate more realistic human movements.

In general, the musculo-skeletal system was modeled as a combination of three models: activation dynamics, contraction dynamics, and skeletal dynamics. Activation dynamics describe the dynamic relationships between neural excitation and activation of muscles.

The dynamics of contraction relate the activation to the resulting muscular forces, taking into consideration the physical characteristics of the muscles, such as the arrangement of the muscle fibers and the passive properties of the tissues. Hill's model was commonly used to model contraction dynamics.

In biomechanics, the computation of muscle functions has been studied through several experiments, and several models have been generated and validated against experimental data. However, the determination of functions that model the behavior of muscles was a challenge due to the large amount of redundancy of the muscular system: the number of muscles that contribute to a movement was greater than the degrees of freedom related to the movement of the skeletal muscles, so it can generate a problem of indetermination. This problem was commonly solved by using optimization approaches, which were generally classified in static and dynamic optimization. Usually, they were defined as finite, restricted, and non-linear optimizations, and were commonly solved using quadratic sequential programming methods [154].

3.2.1 Static optimization

Static optimization, also known as inverse dynamics, takes non-invasive measures of body movements, such as their position, speed, acceleration, and external loads, as inputs to calculate muscle forces.

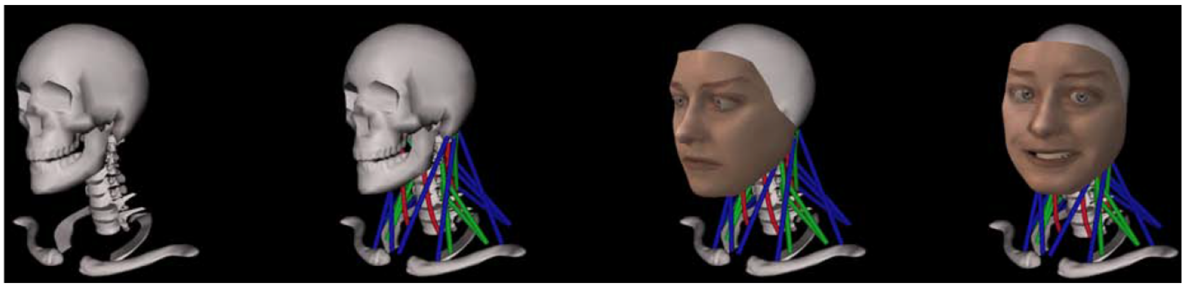


FIGURE 3.7: Neuromuscular simulation of head and neck. The biomechanical system consists of skeleton, muscles, neural control system, and a face with expressions.[3].

In static optimization, since there were no dynamic dependencies between the muscular forces at different instants of time, an integration of time was not required, which makes the problem computationally easier. However, it was difficult to integrate the physiology of the muscle (for example, excitation and activation dynamics) as well as the objective of the motor task (for example, maximum jump height). In addition, its validity was dependent on the precision of the experimental measures of movements.

Komura et al. [155, 156] calculated the activation of the muscles based on human postures of the lower extremities obtained using key-frames, and used an objective function that minimized muscle strength and the amplitude of the activation. In [156] their model was extended to consider physiological characteristics, such as fatigue and muscle injuries. Tsang et al. [1] presented a muscle-tendon model of a human hand and a forearm. Using data from MOCAP or key-frame animation, a set of optimal muscle activations was determined using static optimization, and then used as an input to simulate a model where the hand and forearm achieve a desired pose or movement.

Lee and Terzopoulos [3] developed a biomechanical model of the neck and head using a hierarchical structure to generate the simulations (see Figure 3.7). The system was controlled by two subsystems: a high level voluntary controller, and a low level reflex controller. The voluntary controller generates anticipated neural signals related to the desired poses, muscle tone, and feedback based on a monitoring of the movement in progress. When the reflex controller receives these signals, it determines how the muscles were activated, and modulates the tension levels of the muscles in relation to their current state. An artificial neural network was used to model the voluntary controller. It was trained off-line using precalculated signal functions of a target pose. Finally, to model the force actuators for each muscle, the Hill muscle model was used, defining each muscle-tendon unit as a uniaxial line segment on which the model was applied.

The previous work was extended to simulate the upper part of the human body, by integrating the torso, and the arms [8]. In addition to the muscle and skeletal dynamics model already described, a physics-based system was incorporated to simulate the soft tissue of the body, in

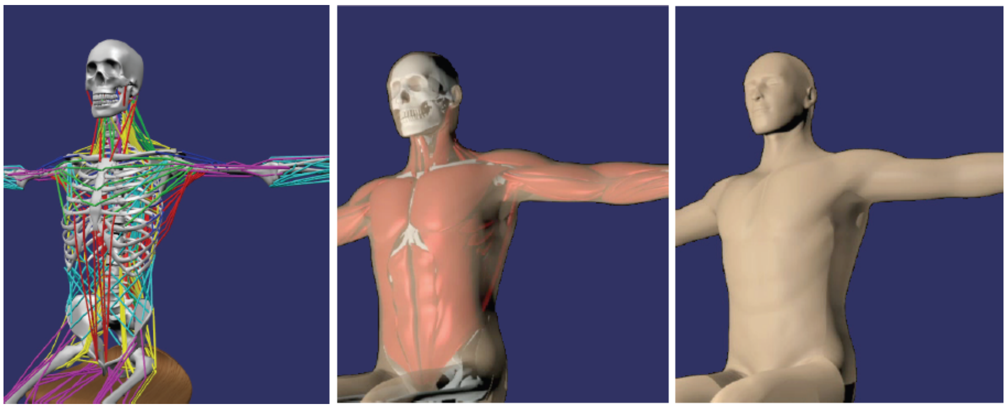


FIGURE 3.8: Thorough modeling of the relevant tissues to exert control over the body. A skeleton that moves with modeled muscular actuators with a uniaxial line segment of Hill (left) was presented. The generated movement deforms the soft tissue (center) and the skin (right). [8].

order to represent realistic deformations of the flesh and skin during body movement. Their body model was composed of 68 bones, with 147 degrees of freedom, as well as 814 muscles (both superficial, intermediate, and deep), each modeled as a line segment with a Hill uniaxial force actuator, and each of them exerts a force on the related bones. The complete model of the upper body can be seen in the Figure 3.8.

Sueda et al. [157] presented a simulation of the musculo-tendinous system of a hand, where the behavior of the muscles and tendons was directed by a dynamics of spline curves. This dynamic was formulated by joining the muscle contraction and the restrictions of forces that were applied to muscles and tendons.

Ruiz [158] addressed the actuation redundancy challenge of muscle-based virtual character control. Actuation redundancy results when the character has more actuators than needed to perform a specific task; this was an important control challenge due to the fact that the character's motion controller must be able to select an appropriate actuation solution among numerous possibilities to achieve a desired task. A control solution for muscle-based characters was proposed consisting of identifying and adapting low-dimensional control representations according to kinematic goals, and it was tested on overhead throwing motions. Surface EMG signal data coupled with MOCAP data of subjects throwing a ball were used in a motion analysis stage. Results showed that only 2 control variables were necessary to encode the important activation trends of sets containing 6-14 muscles. These variables were used as input to control 6 muscles and reproduce the motion of 3 Degrees of Freedom of the character's throwing arm.

3.2.2 Dynamic optimization

Dynamic optimization, also known as direct dynamics, was usually formulated by combining the total force generated by the muscle-tendon unit, the activation dynamics, and the skeletal dynamics. The excitation of the muscles, usually electromyography, was taken as input, in order to produce a movement of the body and then determine the optimal excitation trajectory. Unlike static optimization, which only takes into account a moment of time, dynamic optimization considers the complete duration of the movement, requiring a time integration. Therefore, dynamic optimization is much more computationally demanding than static. However, unlike static optimization, physiological or time-dependent properties can be included as jumps trying to reach the highest possible height [159], vertical jumps in three dimensions [160], or walking [161]. In the case of Anderson and Pandy [161, 162] the minimization of metabolic energy expenditure per unit distance during normal human walking was used.

In [162], Anderson and Pandy demonstrated that static and dynamic optimization generate virtually the same results in predicting muscle forces and contraction forces during normal human walking. They argue that similarity occurs since minimizing muscle fatigue at each instant of time was almost the same as minimizing the metabolic energy expended per unit of distance traveled to complete a walking cycle. In addition, they comment that certain physiological properties, such as strength, length, and speed of the muscles, as well as activation dynamics, have little influence on static optimization.

3.3 Fiber-based simulation

As can be seen in previous sections, the muscle models presented in the literature usually use phenomenological models, and focus on the muscles as a whole, without paying attention to internal structures. Normally, these represent the real anatomy of muscles with simplified geometries in order to minimize computational costs or to apply phenomenological models to their simulations. Simulations based on solid mechanics, such as those based on FEM or FVM, have to use varied techniques to detect collisions, and simulate the effect of contact between muscles. However, these techniques were computationally expensive, and do not work very well with deformable bodies. In addition, the basic primitives of these models, such as tetrahedra or hexahedra, were not deformed in the same way as muscle fibers. [163].

In addition, most of the papers presented use phenomenological models to simulate the forces produced by the muscles; specifically, Hill's muscle model was used. The muscle models based on Hill's work have certain limitations. One of the main ones was that the forces produced by these models were applied along a line of action represented as a line segment in a dimension that joins the bones of the body on which it acts. However, the three-dimensional shape of muscles and intermuscular forces were often overlooked, or simplified. Another problem with this type of models was the one Epstein et al found [164]. In their work they demonstrate that models where muscle fibers, aponeurosis, and tendons were simplified tend to generate erroneous transmission forces, and that it was necessary to consider all the relationships between muscle fibers, connective tissues, aponeurosis, and the tendons. In the same way, Herzog [165] found that Hill muscle models can generate errors in force production of up to 50%, compared to isometric human muscle reference forces, even in controlled scenarios.

The following works focus on making simulations of the muscles considering the different internal structures, focusing mainly on muscle fibers. In addition, they use different biophysical models to simulate the contraction of fibers and the generation of muscle strength.

Ng-Thow-Hing et al. [166] defined B-spline solids (extension of curves and B-spline surfaces to a volume domain) to create deformable models shaped like muscles. Muscle data was used to obtain the shape of the muscles through images of The Visible Human Project [167]. Although these images give an indication of the muscle perimeter, they do not provide information on the internal arrangement of the muscle fibers. To obtain the coordinates of the groups of muscle fibers, an optical triangulation was made of images of three chambers of specimens dissected in series from the soleus muscle of the leg. Using the fiber arrangement information, they generated a method to fit a B-spline solid to the fibers, and thus have a fiber-based solid that approximates the muscle. In order to deform the generated muscle, a viscoelastic network was applied to the control points of the solid.

In [168] previous work was extended to include: collisions between muscles using a method that finds the points closest to a solid; add mass points to the control points, to allow physical reactions of the muscle, and be able to apply muscular forces to the points of mass; use Lagrangian equations to preserve volume; and finally use the fibers as generators of force on bones and generate movement. Figure 3.9 shows the fiber arrangement and the resulting B-spline solid. In [80], previous work was continued by including the aponeurosis of the muscles, and a force model for each fiber was included. The aponeurosis was modeled as an elastic leaf that restricts the deformation of the soleus muscle along the surfaces where it was attached. Each fiber has a Hill model that allows it to contract and generates a non-uniform distribution of the contractile forces within the same muscle.

Blemker and Delp [141] created FEM meshes in three dimensions of the muscles shape based on MRI, and developed a method to prescribe the geometry of the muscle fibers within the mesh, depending on the architecture of each muscle. His method was based on cube-shaped templates of the geometry of the muscle fibers, which were subsequently adjusted to create the geometry of the fibers within the muscle. To achieve this, the cubic template was divided into several sections, which were then projected to similar sections of the target mesh. The base of each fiber geometry template was an interpolation between multiple B-spline curves, to simulate different muscle architectures. A disadvantage of this method was that it was very impractical to represent muscles in three dimensions, because the computational cost of simulating the final meshes was high. This made it not feasible to integrate them with simulations that were already expensive, such as controlling the movement dynamics of the muscles.

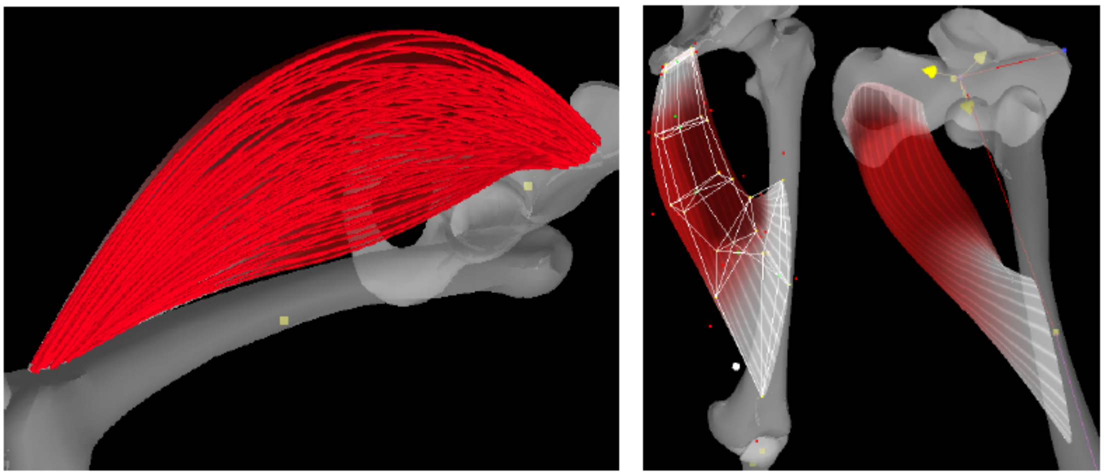


Figure 3.9: On the left side, the fibers that were inside the B-spline solid were shown. In the right part it shows how the muscles were attached to the bones and their deformation when generating a rotation movement. [168].

Kohout et al. [169] presented a method to represent the muscles by means of realistic groups of muscular fibers that were generated automatically in a volume defined by a mesh of a muscle. Its implementation could decompose the volume of a muscle into muscle fibers by adjusting a template of fibers to the volume of the muscle. However, their model had no biomechanical considerations, and can produce unrealistic fiber paths, and that were not close to their actual points of attachment to the bone.

Tang et al. [79] presented a model of FEM in three dimensions to simulate the mechanical behavior of the muscles during their lengthening or shortening. The entire muscle was modeled as a hyperelastic material with active muscle fibers. The model considered that the muscle fibers would join two central points of hexahedrons that form the final mesh of the muscle. The mechanical properties of muscle fibers were described based on the Hill muscle model. However, the model was restricted to applications where the muscle was modeled as ordered hexahedra, and could not be applied to models with more complex geometries.

Pai et al. [163] did an analysis of the deficiencies of simulations of previous muscles, and proposes a muscle model based on fibers to consider biophysical properties that had not been considered. They mention that in previous simulations, muscle mass was not considered correctly. A convenient way to consider it was to unite it with the mass of the bones and that of the soft tissue along a segment of the body. However, when the muscle was stretched or shortened, muscle mass also moves in the direction of stretching or contraction, and will contribute to the inertia of the system. Another problem they encounter in the previous simulations was that the musculoskeletal systems could not deal correctly with contact and with routing restrictions. Finally they mention that simulations based on FEM or FVM, were not ideal for simulations with elements together or tight, since it requires collision detection and resolution algorithms, which were computationally expensive and do not work well with deformable objects. Because

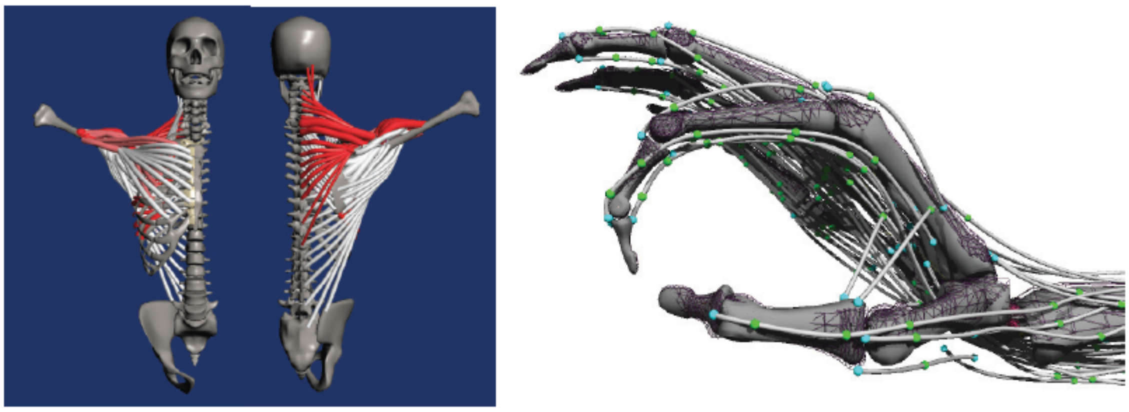


FIGURE 3.10: On the left a simulation of the shoulder is shown modeling the muscles with strands. On the right is a simulation of the movements of a hand using strands. [163].

of these limitations, they propose a strand simulator that was capable of simulating soft, thin, fiber-like fabrics with biophysical properties, and capable of having complex routing constraints (such as those of tendons, muscles, and ligaments). Using Pai's terminology [170], the term *strand* was used to indicate that the fibers were not only curves in a space, but that they have mass, elasticity, and other physical properties that influence their dynamics. In the figure Figure 3.10 you can see two simulations resulting from the proposed simulator.

3.4 Control with the Bidomain and monodomain models

Recently the fields of computational modeling and medical image processing have advanced significantly allowing the biomedical engineering community to focus on the development of patient-specific models. In the context of cardiac electrophysiology, such personalized models offer the ability to better understand the heart in pathological conditions, to develop and improve new therapies and also to optimize complicated procedures. However, before introducing these computer models for clinicians, efficient numerical and computational techniques for the fast solution of the mathematical models underlying the complex phenomena of electrical activity of the heart have to be pursued. One of the first simulations of cardiac electrophysiology was presented by Leon and Horáček [171]. Their model of propagated excitation and recovery in anisotropic cardiac tissue consisted of a large number of excitable elements whose interactions were governed by the anisotropic bidomain model. The authors' objective was to develop a model for simulating large and complex cardiac structures with different tissue types and anisotropic properties. This was accomplished by tesselating a tissue region into subregions characterized by two static parameters: a cell type T and a principal fiber direction. Associated with each T was a set of electrophysiologic properties, which allowed modeling of inhomogeneous tissue.

The bidomain equations are the most complete description of cardiac electrical activity. Their numerical solution is, however, computationally demanding, especially in three dimensions, because of the fine temporal and spatial sampling required. Vigmond et al. [172] proposed several techniques to speed up this computation. The first one was to recast the equations into a parabolic part and an elliptic part. The parabolic part was solved by either the FEM or the interconnected cable model (ICCM). Since the elliptic equation was more demanding, it was solved by FEM on a coarser grid and at a reduced frequency. The ICCM was found to be about twice as fast as the FEM for solving the parabolic portion of the bidomain problem.

Nickerson [89] developed a computational modelling and simulation framework for cardiac electromechanics which for the first time tightly coupled cellular, tissue, and whole heart modelling paradigms. Applications of the framework were demonstrated in simulations of the electrical activation and mechanical contraction of cardiac myocytes, myocardial tissue, and models of ventricular structure and function. The framework was implemented as part of the Continuum Mechanics, Image analysis, Signal processing, and System identification (CMISS) computational modelling environment. This allowed for a detailed specification of tissue microstructure and the variation of cellular models and their material parameters over a geometric domain of interest.

Keener and Bogar [173] presented a numerical scheme for efficient integration of the bidomain model. The scheme was a mixed implicit–explicit scheme with no stability time step restrictions and requires that only linear systems of equations be solved at each time step. Authors suggested that by numerically integrating the non-linear, time-dependent bidomain equations, using a Crank–Nicolson step for the spatial discretization, and a multigrid inversion of the linear system at each time step, the method was faster than a fully explicit method.

Austin et al. [174] followed the idea of using multigrid methods, and presented a more robust method, called Black Box Multigrid, as an alternative to conventional geometric multigrid. Additionally, the effect of discontinuities on solver performance for the elliptic and parabolic part was investigated. Results indicated that for certain discontinuous bidomain problems, Black Box Multigrid provides 60% faster simulations than using conventional geometric multigrid.

Since the solution of the bidomain equations was computationally expensive, due to the fine spatial and temporal discretization needed, the size and duration of the problem which can be modeled were limited. Vigmond et al. [175] presented a review of the methods that had been used to solve the equations, giving particular interest to multigrid methods. Authors noted that these methods offer the fastest solution time, as well as having only modest memory requirements. The monodomain model of electrophysiology was often used to approximate the more accurate bidomain model because of the huge computational requirements of the latter. It has been claimed that there was a difference of two orders of magnitude between these models. Sundnes et al [93] provided numerical and analytical arguments discussing this relation in the presence of both simple and complex cell models. Authors observed that for simple cell models, the PDE part of the solution procedure dominates the CPU efforts, and the bidomain model requires about six times the CPU efforts of the monodomain model. For the complex model, the ODE part requires much more CPU efforts, and since these computations were virtually the same for the monodomain and the bidomain model, the factor was reduced to below two. Authors also note other reasons than CPU considerations for approximating the bidomain solutions with solutions of monodomain-type models. The monodomain model was easier to understand from a mathematical point of view. Also, from a viewpoint of computation and programming, a scalar PDE was much easier to handle than a system of PDEs.

Most numerical schemes for solving the monodomain or bidomain equations use a forward approximation to some or all of the time derivatives. This approach, however, constrains the maximum timestep that may be used by stability considerations as well as accuracy considerations. Whiteley [96] proposed a semi-implicit algorithm that ensures stability. This method uses either a semi-implicit approximation or a Crank-Nicolson approximation to update the transmembrane and extracellular potentials, and requires only the solution of a linear system. The author reported a substantial reduction in computation time at the cost of only a slight reduction in accuracy. Whiteley [176] then extended the previous work to create an efficient numerical scheme which utilized the observation that the only component of the ionic current that had to be calculated on a fine spatial mesh and updated frequently was the fast sodium current. Other components of the ionic current could be calculated on a coarser mesh and updated less frequently, and then interpolated onto the finer mesh. The use of this technique induced very little error in the solutions.

Röhrle [84] developed an electromechanical model of the skeletal muscles. The model coupled the electrophysiological properties at the cellular level (the contractile unit of the muscle) with the biomechanical principles at the organ level (the whole muscle), focusing on the generation of muscle strength. To model the electrical activity of muscle fibers, without describing the electro-physiological properties of a single cell, the bidomain equations [34, 84, 172] were used. These equations provide a continuous modeling approach, where the extracellular and intracellular spaces were modeled as if they occupied the same space. Finally, force equations were generated to manage the muscle contraction, considering as input to the system an electric current, which supplies the fibers. Muscle fibers were modeled as one-dimensional objects, which were discretized using linear finite elements of a Lagrange dimension. These fibers were aligned in a three-dimensional space along the actual direction of the fibers of a muscle (these data were obtained from [167]). Also included was a level of detail for fiber graphing, where you could determine if a one-dimensional chain represents a fascicle, a fiber, or a group of fibers.

Röhrle et al. [34] extended the previous work by presenting a physiological model of skeletal muscles that was capable of representing geometric descriptions of muscle fibers and their grouping. In conjunction with a motor neuron activation model in muscles, the electrophysiological

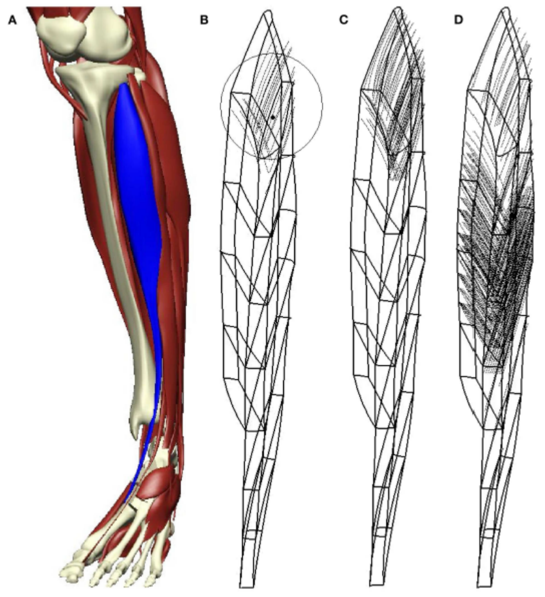


FIGURE 3.11: Anterior tibial muscle shown in blue (A). It shows a motor unit, and the muscle fibers related to that unit (B). Muscle fibers related to 5 (C) and 10 (D) motor units are shown.[34].

behavior of muscle fibers within a motor unit was calculated and applied to obtain the muscular forces that generate movements. In this case, it was considered that all fibers were innervated at their midpoint, so it was sufficient to model the activation of one muscle fiber per motor unit, and use its output for all other associated fibers. The fibers were inside a finite element, an anatomically realistic mesh generated from images of the visible human project [167], of the anterior tibial muscle. The presented model was also capable of simulating the fatigue generated in the muscles. In Figure 3.11 the arrangement of muscle fibers for different motor units can be seen.

Heidlauf and Röhrle [177] presented an extensible, flexible, multiscale, and multiphysics model for nonisometric skeletal muscle behavior. The skeletal muscle model was based on the entire excitation-contraction pathway by coupling a biophysical model to the propagation of action potentials along skeletal muscle fibers. Since the macroscopic electrical conductivity of muscle tissue perpendicular to the fiber direction is up to one magnitude lower than the conductivity along the fiber direction [178, 179], and electrical stimulation from one fiber to adjacent ones is not observed, the propagation of action potentials along the fibers was modeled as a 1D system. In such case, the bidomain equations reduce to the simpler monodomain equation, which was used to solve the bioelectrical field. Even though authors used parallel techniques to speed up their model, they only achieved a 2.185 speedup for a simulation of 200 time steps which took 81360.24 seconds to compute. Additionally, said performance was obtained on a cubic geometry with 2cm edge lengths, and discretized using eight finite elements.

For a detailed description of the numerical methods for the solution of the bidomain equations, refer to Linge et al. [180].

3.4.1 Using the Lattice Boltzmann method

One method that has recently been used to simulate cardiac electrical activity using the monodomain model, was the Lattice Boltzmann method (LBM). Campos et al. [97] proposed a GPU model based on the LBM that solved the reaction–diffusion model 2.4, that describes the electrical activity in cardiac tissue. Authors later validated and compared against a traditional FEM solution in a benchmark problem [181].

For simulations using the LBM, the domain was discretized by an equally spaced Cartesian grid. Every node of the grid has N velocity directions $\mathbf{e}_i$ ($i = 0, \ldots, N-1$) and N particle distribution functions f_i which describe the probability of a certain number of particles moving in the velocity direction $\mathbf{e}_i$. In the LBM framework, the distribution function of the scalar quantity v at time t, position $\mathbf{x}$ with velocity in the direction $\mathbf{e}_i$ was denoted by $f_i(\mathbf{x}, t)$. Here v was the transmembrane potential.

The lattice Boltzmann equation that describes the evolution of the particle distribution functions $f_i(\mathbf{x}, t)$ may be written as:

$$f_i(\mathbf{x} + \mathbf{e}_i \Delta\mathbf{x}, t + \Delta t) - f_i(\mathbf{x}, t) = \Omega(\mathbf{x}, t) \tag{3.1}$$

where $\Omega(\mathbf{x}, t)$ was the collision operator for v and depends on distribution functions $f_i(\mathbf{x}, t)$, and Δt and Δx were the time step and grid interval, respectively. The left-hand side represents free transport of the particles (streaming), while the right-hand side describes interactions of the particles through collisions. Authors proposed a modification to the collision term to that it may be written as the sum of a reactive term Ω^R and a non-reactive term denoted by Ω^{NR}. The non-reactive part of the collision operator was described by the traditional Bhatnagar–Gross–Krook (BGK) approximation [182], which models the collision as a single-time relaxation towards a local equilibrium. The reaction term R that appears in the reactive collision operator was determined by the cell model used to describe the kinetics of the cellular membrane; in this case, the monodomain model.

Corre and Belmiloudi [183] also proposed a LBM simulation of cardiac electrophysiology, using the bidomain model, in order to capture the detailed activities of macro-to micro-scale transport processes. Authors took into account domain anisotropical properties using intracellular and extracellular conductivity, such as in a pacemaker or an electrocardiogram, in both parallel and perpendicular directions to the fibers.

3.4.2 Parallel implementations

Various parallel implementations of solvers have been developed and applied for simulations of tissue electrophysiology. For example, some simulators [184] were based on OpenMP (Open Multi-Processing), an implementation of multithreading that supports shared memory architectures. The main advantage of OpenMP was that implementations can be easily adapted for parallel execution, since the serial implementation needs only to be slightly modified or can use OpenMP indirectly by interfacing with specific linear algebra packages able to deal with OpenMP. An alternative approach was based on Message Passing Interface (MPI), a message-passing application programming interface that allows codes running on many computers to communicate with one another. In this case, specific code has to be developed to deal with MPI, but high scalability can be achieved. MPI based code can run on both distributed and shared memory architectures. The principal advantage of MPI was that it allows usage of systems with distributed memory architectures, which have only a fraction of the cost of systems with shared memory. Examples of this programming strategy applied to tissue electrophysiology simulations can be found in [185, 186].

Bordas et al. [98] described the available procedures for numerical modelling of the bidomain equations and reviewed developments in adaptive numerical algorithms, the development of spectral element (SE) methods as a high-performance alternative to FE, and state-of-the-art parallel linear solvers for large-scale algebraic systems. Additionally, authors discussed the alterations that would be required in order to ensure that the code designed for sequential machines or clusters, would be able to run and scale appropriately con High Performance Computing (HPC) facilities. These, at the time, were necessary to achieve the speedups required for interactive applications.

3.4.2.1 Using Graphical Processing Units (GPU) for computation

The increased computational power and memory of graphics processing units (GPUs), combined with decreasing costs, has generated significant interest in utilizing graphics hardware for other applications. GPUs have been optimised for traditional computer graphics, which was focused on highly data-parallel operations on floating-point numbers, and provide less of an advantage for activities outside this range, such as intensive memory communication and integer and double precision calculations [187]. Nevertheless, many computationally demanding calculations may benefit from GPUs, which have already been applied to simulations outside graphics and visualisation applications, including cellular automata, dendritic growth, fluid and gas dynamics, signal and image processing, geometric computing, and reactione diffusion equations [187].

Vigmond et al. [188] considered the use of GTX 280 GPUs to accelerate cardiac models based on the bidomain equations, and analyzes results in the context of simulating a small mammalian heart in real-time. The ODEs associated with membrane ionic flow were computed on traditional CPU and compared to GPU performance, for one to four parallel processing units. The scalability of solving the PDE responsible for tissue coupling was examined on a cluster using up to 128 cores. Results indicate that the GPU implementation was between 9 and 17 times faster than the CPU implementation and scaled similarly.

Sato et al. [189] simulated the electrical wave propagation in cardiac tissue using Nvidia Geforce 9800 GPUs. The cardiac tissue was modeled using the monodomain equations. Authors found that the computational speed of two-dimensional (2D) tissue simulations with a single commercially available GPU was about 30 times faster than with a single 2.0 GHz Advanced Micro Devices (AMD) Opteron processor. GPU simulations of wave conduction in a three-dimensional (3D) anatomic heart were 1.6 times slower, when compared to a 32-central processing unit (CPU) Opteron cluster. However, a cluster with two or four GPUs was faster than the CPU-based cluster.

Bartocci et al. [190] presented GPU simulations of realistic, detailed cardiac-cell models whose compute times were close to real-time using a desktop computer equipped with a Tesla C1060 GPU. Authors considered five different models of cardiac electrophysiology that span a broad range of computational complexity. To achieve the maximum gains in computational efficiency, it was necessary to consider model-specific aspects of the implementation, including appropriate division of the model among multiple kernels and careful use of the available levels of memory.

Jararweh et al. [191] presented an evaluation study for porting a cardiac simulator of the bidomain equations to the high performance Tesla C1060 GPUs. Authors also conducted a comparative evaluation using conventional computing platforms: a single CPU and a CPU Cluster system. A speedup of up to 81.5 was achieved when compared to a single CPU core, while a speedup of up to 21.1 was achieved when compared to a CPU cluster.

Neic et al. [192] used Tesla c2070 GPUs instead of HPC hardware to simulate anatomically realistic and biophysically detailed multiscale models of the heart. Authors demonstrated the feasibility of multi-GPU bidomain simulations by running strong scalability benchmarks using a state-of-the-art model of rabbit ventricles. Results obtained revealed that there were considerable speedups, from 11.8 to 16.3, for all components of the computational scheme.

Xia et al. [193] tackled the issue of large-scale 3D simulations of heart models by developig a GPU-based simulation algorithm to simulate the conduction of electrical excitation waves in the 3D atria. In the GPU algorithm, a multicellular tissue model was split into two components: one was the single cell model (ordinary differential equation) and the other was the diffusion term of the monodomain model (partial differential equation). Furthermore, several optimization strategies were proposed based on the features of the virtual heart model, which enabled a 200-fold speedup as compared to a CPU implementation.

Gouvêa de Barros et al. [194] developed a cardiac electrophysiology model described by the monodomain equations, that captured the microstructure of cardiac tissue and used a cell model based on Markov chains for the characterization of ion channel's structure and dynamics. The model was parallelized using cluster computing and GPUs. The implemented model was able to reduce the execution times of the simulations from more than 6 days (on a single processor) to 21 minutes (on a small 8-node cluster equipped with 16 GPUs, i.e., 2 GPUs per node). Cordeiro et al. [195] extended the Multi-GPU solver by coalescing the data and GPU kernels executions in the multi-GPU environment. This new scheme was tested only for the solution of the systems ODEs. The results showed that the proposed scheme was very effective, having reduced the execution time to solve the systems of ODEs by half, when compared to a scheme that does not implemented the proposed data and kernel coalescing. The total execution time of cardiac simulations was 25% faster.

Nimmagadda et al. [196] evaluated the performance of single and multi-GPU implementations of the bidomain model, and simulated interactions between the cells. Authors evaluated the architecture specific fine grained parallelization and optimization strategies, identified the suitable threads per block configuration, and study the impact of memory organization and coalesced memory access on performance. Simulating one action potential duration (350 msec real-time) for a 256 x 256 x 256 tissue took 453 hours on a high-end general purpose processor, while it took 664 seconds on a four-GPU based system including the communication and data transfer overhead.

Amorim and dos Santos [197] presented a GPU cardiac simulator based on the bidomain equations. The use of a Geforce GTX260 GPU accelerates the cardiac simulator by about 6 times compared to the best performance obtained in CPU. In addition, the GPU implementation was compared to an efficient parallel implementation developed for cluster computing. A single desktop computer equipped with a GPU was shown to be 1.4 times faster than the parallel implementation of the bidomain equations running on a cluster composed of 16 processing cores. Authors also tested the monodomain model, and achieved speedups around 20 times faster than the CPU implementation.

Vigueras et al. [198] ported to the GPU a number of components of CHeart—a CPU-based finite element code developed for simulating multi-physics problems. Specifically, authors implemented on the GPU the ODE and PDE solution steps for the monodomain equations. Results show that for a human scale left ventricle mesh, GPU acceleration, using the Nvidia G80 GPU, provided speedups of 164 compared with single-core (SC) and 5.5 times compared with multi-core (MC) for the solution of the ODE model. Speedup of up to 72 compared with SC and 2.6 compared with MC was also observed for the PDE solve.

Zhang et al. [199] presented the G-Heart system where GPU-based acceleration technologies were adopted for both the simulation of cardiac electrophysiological activities and the online rendering of 3D anatomical and electrophysiological data. Authors used the model of human ventricle presented by Tusscher et al. [200] to represent the excitable dynamics of cardiac tissue. Authors computed the simulation of cardiac electrophysiology on a Tesla C1060 GPU, and rendered the visualizations on a Geforce 9600 GPU.

Wülfers et al. [201] used OpenCL to develop a cross-platform software to compute the macroscopic monodomain model for excitation conduction and an atrial myocyte model for ionic currents. On a CPU with 12 HyperThreading-enabled Intel Xeon 2.7 GHz cores, authors achieved a speed-up of simulations by a factor of 1.6 against existing software that uses OpenMP. On two high-end AMD FirePro D700 GPUs the OpenCL software ran 2.4 times faster than the OpenMP implementation. The more nodes the discretized simulation domain contained, the higher speed-ups were achieved.

Menta et al. [202] took advantage of newer GPU and CUDA versions available, and developed a novel electrophysiology simulation software entirely developed in Compute Unified Device Architecture (CUDA). The software implemented fully explicit and semi-implicit solvers for the monodomain model, using operator splitting. With the GPU based solver on a Tesla C2090 GPU using double precision arithmetic, a speedup of over 50 was obtained for three-dimensional problems.

Considering that the LBM was very suitable to parallel computing due to its high locality, it turns out that a parallel GPU implementation of the LBM for solving cardiac electrophysiology models was a promising approach for performing near real-time simulations.

Campos et al. [97] presented a LBM solver for computational simulations of the cardiac electrical activity using the monodomain equations. Authors validated and compared their solution against a traditional FEM solution in a benchmark problem [181]. The results showed speedups of up to 500 for the overall simulation and for the LBM a performance of 419 mega lattice updates per second was achieved. With near real-time simulations in a single computer equipped with a modern GPU these results demonstrated that the proposed framework was a promising approach for application in a clinical workflow.

To speed up cardiac simulations and to allow more precise and realistic ones, 2 different techniques have been traditionally exploited: parallel computing, with multi-code CPUs and GPus, and sophisticated numerical methods. To test which technique was more effective at accelerating cardiac simulations, and test whether the combination of either results in a greater or smaller speedup, Oliveira et al. [203] evaluated them to test the achieved speed up for cardiac electrophysiology simulations based on the monodomain equations. Authors used the benchmark proposed by Niederer et al. [181] to verify the different solvers and configurations. The obtained results suggested that by combining all the techniques, the speedups ranged between 165 and 498. The tested methods were able to reduce the execution time of a simulation by more than 498 for a complex cellular model in a slab geometry and by 165 in a realistic heart geometry simulating spiral waves.

3.5 Meshless methods for Biomechanical simulations

The FEM is one of the most popular discretization technique available for biomechanics simulations. However, its performance relies strongly on the model's mesh quality. Additionally, any mesh modification or mesh refinement during the analysis represent an extra computational cost, which is a significant drawback in biomechanics.

Meshless methods have several advantages over the FEM, and as such have been used to simulate a wide range of biomechanical phenomena. The work of Doweidar et al. [204] showed that meshless methods possess clear advantages over the FEM in biomechanical problems dealing with large strains, such as in the simulation of the human lateral collateral ligament and the human knee joint. Additionally, Zhang et al. [205] extended a meshless method to the nonlinear explicit dynamic analysis of the brain tissue response. The results confirmed the accuracy of meshless methods to deal with highly demanding nonlinear hyperelastic biomaterials.

Furthermore, meshless methods have been used to simulate hemodynamics. Tsubota et al. [206] used meshless methods to simulate the motion of a deformable red blood cell in flowing blood plasma, while Mori et al. [207] studied the effect of red blood cells on the primary thrombus formation. Another popular computational biomechanical field in which meshless methods proved to possess clear advantages is the in-silico prediction of bone tissue remodelling [208]. Lee et al. [209] applied meshless methods to simulate the bone tissue remodelling process with success. Recently, Belinha et al. [210] presented a new bone tissue remodelling algorithm relying on the meshless method accuracy. The methodology was capable to obtain numerical solutions very close with the clinical X-ray images of natural bones, and natural bones with implants.

Soft tissue simulations have also benefited from the meshless formulation. Horton et al. [211] presented a meshless method for computing the deformation of soft tissue. The model ran at half the speed of a hexahedral-based finite element simulation but three times faster than a similar tetrahedral-based simulation. Additionally, authors developed a finite element simulation for comparisson, and obtained less than 5% differences in forces and displacement of elements between the simulations. Banihani et al. [212] introduced the point collocation method of finite spheres to model hyper-realistic responses of soft biological tissue for virtual surgery simulations. The method is a physics-based meshless technique that models the behavior of organs in real-time. Similarly, Zhang et al. [205] used a meshless algorithm for the simulation of hyper-realistic soft tissue mechanical responses of brain indentation. Their approximation uses a large number of nodes which significantly delays mesh distortion when large deformations are present. Authors verified the accuracy of their approach against a FEM simulation.

Specifically, for skeletal muscle simulations, Basava [213] presented the meshless Reproducing Kernel Collocation Method (RKCM) in context of nonlinear hyperelasticity. The method achieved both computational efficiency and controllable accuracy for large scale problems. Chen et al. [214] introduced the meshfree Reproducing Kernel Particle Method (RKPM) for 3D image-based modeling of skeletal muscles. This approach allows for construction of simulation model based on pixel data obtained from medical images. The pixel points from these images can be directly used as nodes for domain discretization in the meshless modeling. Valizadeh et al. [215] implemented a 3D patient specific leg-muscle pixel-based model using isogeometric analysis (IGA) and the RKPM. Authors preserved geometric exactness by using IGA for the representation of the exact geometry of the problem domain boundary, while the RKPM discretization is employed in the interior of the domain.

3.5.1 SPH for Biomedical simulations

The extension of SPH to simulate biological tissue was relatively sparse, with a few examples of fluid SPH confined by meshes [216–219], and simulations of a virtual liver [220], lips [62], cartilage [45], and generic biological tissues [60, 64]. In this section, a brief description of said simulations was presented, focusing on the way in which SPH was used in soft tissue simulations.

3.5.1.1 Virtual surgery

SPH has been used in virtual surgery to simulate biological fluids, such as blood, which were contained by tissues modeled by different methods, such as FEM or mass-spring.

Müller et al. [216] presented an interactive method based on SPH to simulate blood as a fluid and with deformable solids represented by a Finite Element approach. The finite element mesh (simulating tissue) was able to fracture due to pressure forces in the blood stream.

Qin et al. [217] proposed a particle-based solution to simulate the interactions between blood flow and vessel wall for virtual surgery. By coupling two particle-based techniques, SPH and mass-spring model (MSM), authors simulated the blood flow and deformation of vessel seamlessly. At the vessel wall, particles were considered as both boundary particles for SPH solver and mass points for the MSM solver.

Farazi et al. [219] presented a 3D biomechanical swallowing model of the oral, pharyngeal and laryngeal (OPAL) muscles and structures. The model consists of a mixture of rigid bodies with 6-degrees of freedom (DOF) frames and finite element models (FEMs) with 3-DOF for each node within the volumetric body. The bony structures (jaw and teeth) were modeled as rigid bodies and the soft structures (tongue and soft palate), which exhibit large deformations during swallowing, were simulated as FEMs with tetrahedral elements. Authors used the SPH formulation described in [221] to model water-like and nectar-like fluid boluses, simulated within an airway-skin mesh that encompasses the modeled 3D structures and follows the model's dynamics. This airway skin-mesh acted as the deforming boundary for the boluses: as the airway-mesh deformed due to the change in the OPAL geometry during a swallow, it created forces on the fluid particles which then move to represent the bolus movement.

Chui et al. [218] introduced a particle-based rheologic modeling framework for simulating thrombus formation in medical simulation applications. The effect of blood rheology was simulated with SPH, and by modeling vessel wall and embolizing coil as virtual particles, a pure Lagrange particle formulation for fluid-structure interaction was proposed for modeling the blood-vessel or blood-coil interaction. To simulate the blood vessel, authors used constitutive equations for modeling rubber-like solids as proposed by Chui and Heng [222].

3.5.1.2 SPH for Biological soft tissue simulations

In computer aided surgery the accurate simulation of the mechanical behavior of human organs was essential for the development of surgical simulators. One of the first works that simulated biological systems with said objective was presented by Heiber et al. [220], who introduced the remeshed Smoothed Particle Hydrodynamics (rSPH) method to simulate viscoelastic solids, and their interaction with fluids. A key aspect of this approach was its flexibility which allowed the simulation of complex time varying topologies with large deformations. Authors presented a reconstruction of the liver topology and its strain distribution under a small local load.

Gastélum et al. [62] presented a user-specific 3D mechanical lips model where lips were modeled as a set of particles whose dynamic behaviour was governed by SPH with an elastic material smoothing kernel introduced by Solenthaler et al. [55]. Internal forces were defined by the elastic forces presented between the particles, obtained using the strain energy (potential energy) stored in each particle. An ellipsoid force field encircling the lips simulated the muscles controlling the lips' motion. The muscles were also represented by particles so that they could interact with the lips using the same scheme. Using similar techniques for the simulation of tissue, in [63] Gastélum et al. modelled the esophagus and the stomach using SPH. Authors considered a multilayer model of particles related to a single triangle mesh, where each layer represents distinct biological tissues of the esophagus and the stomach. The model was able to simulate esophagus changes due to internal muscular changes and user input forces, and presented the advantage of avoiding to redefine the system after morphological modifications of the mesh, such as in cases of large deformations.

Preoperative simulations of real patients that could be manipulated during the planning of procedures would be of great benefit to surgeons. However, the methods that were used to simulate biological structures, such as FEM, were not suited to perform interactively in such simulations. To address that problem for the case of Femoral Acetabular Impingement (FAI), Boyer and Joslin [45] used SPH to simulate articular cartilage as a biphasic fibril-reinforced poroviscoelastic material. The objectives were to implement such a model and validate it using indentation and unconfined compression tests while considering the necessity of computational efficiency as applicable to a future real-time hip simulation. To simulate the fibers, authors followed the method proposed by Gupta et al. [223], where the placement of fibres within the material was simplified with the assumption that a fibre was created between every particle and each of its neighbours, which in this case were arranged in a regular lattice.

Biological entities often have physical features that were not found in standard mechanical devices such as elastic matter for an outer shell (skin or cell membranes) and internal reservoirs of liquids or gels (blood, brain fluid, cytoplasm). Parts responsible for active movement (such as muscles) also require elastic matter that can contract on demand. These often interact with liquids or gels that were incompressible in an external environment as well, which means that the surfaces of the elastic matter interact with the surfaces of the liquids.

Palyanov et al. [64] presented an open source software built using OpenCL called Sibernetic, designed for the physical simulation of biomechanical matter (membranes, elastic matter, contractile matter) and environments (liquids, solids and elastic matter with variable physical properties). It was built as an extension to Predictive–Corrective Incompressible Smoothed Particle Hydrodynamics (PCISPH) [224]. The authors also introduced elastic connections that can be used to create contractile matter which can act in a muscle-like manner. Contractile matter was implemented by adding the ability to connect particles together with a special kind of connection refered to as a “contractile fiber”. A contractile fiber connects two particles and can exert an equal and opposite force on them. A contractile fiber was a special type of elastic interaction, and therefore has a spring stiffness co-efficient associated with it.

A contractile fiber can contract in response to an incoming signal outside of the physics of the system. Therefore the force applied by the contractile fiber, in addition to having a spring component, has a second component that was driven by a time-varying parameter that can be modulated as an input to the simulation. As this parameter changes, a force proportional to its value was applied to the pair. This force acts in the direction of the midpoint of the straight line connecting the ith and jth particles. With the ability to chain particles together in this manner, it was possible to organize contractile fibers into larger lattices of contractile matter. Lattices can also include elastic matter relationships, enabling the construction of simple muscle-like tissue.

Rausch et al. [60] tackled the issue of biological soft tissue damage and failure by modeling it using a particle/continuum approach; specifically, authors combined continuum damage theory with the Normalized Total Lagrangian SPH. The total lagrangian SPH consists in establishing a reference configuration of particles neighbors, instead of frequently updating the list of neighbors for each particle. The proposed method lend itself well to the implementation of standard constitutive relations of finite strain anisotropic elasticity and thus applied to soft tissue modeling. Authors demonstrated that, for simple cases of uniaxial and biaxial extension as well as clamped uniaxial extension, the SPH method predicted elastic responses under large deformations.

3.6 Deformable solid simulations with the SPH method

SPH has more sparingly been applied to solids, beginning with the work of Desbrun and Cani [106], and extended to bending, fracturing, and high impact tests [107, 109]. Solenthaler et al. [55], provided an approach to derive the deformation field of an elastic solid, which was improved upon by Becker et al. [57] by a corotation method to prevent unrealistic forces in rotations. Here, a summary of the SPH based solid solvers was presented.

Desbrun and Cani [106] where the first to present a simulation of highly deformable and separable bodies with the SPH method. Authors extended the method to the animation of inelastic bodies with a wide range of stiffness and viscosity, and concluded that the smoothed particles paradigm leads to a coherent definition of the object's surface as an iso-surface of the mass density function. The choice of the smoothing kernel h was very important: it gives the radius of influence of interaction forces created by a particle. Different behaviors can be obtained by tuning h: A small value will create very local interactions so the body will separate more easily into pieces. Authors concluded that particle systems provided an easy approach for modeling large changes in shape and in topology.

Bonet et al. [107] described a variational formulation of SPH for both fluids and solids applications. The resulting equations treated the continuum as a Hamiltonian system of particles where the constitutive equation of the continuum was represented via an internal energy term. For solids this internal energy was derived from the deformation gradient of the mapping in terms of a hyperelastic strain energy function. Since the energy terms were independent of rigid body rotations and translations, this formulation ensured the preservation of physical constants of the motion such as linear and angular momentum.

Paiva et al. [225] presented a visually realistic animation technique for objects that melt and flow. It simulated viscoplastic properties of materials such as metal, plastic, wax, polymer and lava. The technique consists in modeling the object by the transition of a non–Newtonian fluid with high viscosity to a liquid of low viscosity. During the melting, the viscosity was formulated using the General Newtonian fluids model, whose properties depend on the local temperature. The phase transition was then driven by the heat equation. The fluid simulation framework uses a variation of SPH.

Solenthaler and Pajarola [55] presented a method for the simulation of melting and solidification in a unified particle model based on the SPH method for the simulation of liquids, deformable as well as rigid objects, which eliminates the need to define an interface for coupling different models. With said approach, it was possible to simulate fluids and solids by only changing the attribute values of the underlying particles. Melting and solidification were also introduced, where arbitrary sets of particles can solidify into deformable bodies. Since previous kernels turned out to be unsuitable for those situations, authors designed the following new kernel function:

$$W(\mathbf{r}, h) = \begin{cases} c\frac{2h}{\pi}\cos(\frac{(r+h)\pi}{2h}) + c\frac{2h}{h} & 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

$$c = \frac{\pi}{8h^4(\frac{\pi}{3} - \frac{8}{\pi} + \frac{16}{\pi^2})} \tag{3.3}$$

Cleary and Das [109] described the advantages of using mesh free simulation methods such as SPH for elastic and elastoplastic deformation and for brittle fracture. Using only particles with no prescribed geometric linkages (such as in a mesh or a grid) allowed high deformations to be dealt with easily in cases where finite element methods would either fail and/or require expensive and diffusive re-meshing. SPH was able to solve elastic deformation problems with accuracy comparable to finite elements and demonstrated excellent stability and adequate convergence. The method had powerful abilities to model material free surfaces, extreme deformation including self-collision and automatically carries stress and strain history and material dependent information. Brittle fracture of rocks during impact and compression was demonstrated and with good predictions for fragmentation patterns.

Becker et al. [57] extended the work of Solenthaler and Pajarola [55] to include a novel corotational SPH formulation for elastic deformable solids, which allowed to use the linear Cauchy-Green tensor to calculate elastic forces for a wide range of scenes. Authors adopted the original corotational idea for the FEM [226, 227] to SPH. The rotations in the deformation field were computed using an SPH variant of the shape matching method [105].

3.7 GPU-based SPH simulations

SPH has traditionally been an expensive computational method, mainly due to two factors [110]. First and foremost, compared to many mesh-based schemes, the number of interactions with neighbouring particles for each particle can be up to 100 in 2-D simulations and 200–500 in 3-D simulations, which was far greater than the number of neighbouring cells for FVM and FEM stencils. Secondly, the main options for simulating (almost) incompressible fluids were using the Weakly Compressible SPH (WCSPH) [228] which has a very small time step (typically on the order of 10^{-6} - 10^{-5} s) due to the use of explicit time integrators, or Incompressible SPH (ISPH) [229] which uses time steps an order of magnitude larger, but requires the solution of a pressure Poisson equation (not a trivial task for simulations with many millions of particles).

Until recently, the only viable option was to use high-performance computing (HPC) using many thousands of cores with a standard message passing interface (MPI) to handle communication between processors on distributed memory systems [230]. Multi-core processors have become more popular in recent years, where the CPU on each processor can have up to dozens of cores (at the time of writing) with shared memory requiring programming frameworks such as OpenMP. Obtaining the maximum performance from multiple nodes with multiple cores with a mixture of distributed and shared memory has required a combination of OpenMP-MPI approaches working across heterogeneous architectures [231]. This was technically difficult, expensive in terms of hardware investment and maintenance, and highly restrictive in terms of code portability across other architectures – an important consideration for disseminating use throughout industry.

More recently, GPUs have been used to take the place of HPC clusters. Originating from computer games and the computer graphics industry, GPUs were highly portable devices designed for high throughput data processing. With SPH ideally suited to the streaming multi-processor parallel architecture of GPUs, several works have been developed for said architecture instead of depending on expensive HPC systems. Here, a brief survey of such work was presented.

3.7.1 Using GPU texture memory

The first implementation of the SPH method totally on GPU was realized by Kolb and Cuntz [232] in 2005, Hegeman et al. [233] in 2006, and Harada et al. [234] in 2007. These approaches used GPU texture memory, because there was no dedicated GPU compute platform available at the time.

Hegeman et al. [233] proposed a dynamic quadtree structure to find the neighbors for a given particle; the quadtree was efficiently rebuilt at each time step of the simulation. The GeForce

FX 7800 GPU implementation ran nearly an order of magnitude faster than CPU versions for 2D experiments with up to 65k particles.

By 2007, Harada et al. [234] also tackled the problem of neighbor search on the GPU by proposing the use of a bucket data structure: For each particle, the texel in texture memory to which each particle belongs to was computed, and the particle index was stored in it; neighbor search operations were performed via shaders. Authors also used a 3D texture in which a 3D array was divided into a set of two-dimensional arrays and was then placed in a large texture, in order to simulate 3D fluids. With a Geforce 8800GTX GPU, 3D simulations of fluids with 65k particles ran 17.3 times faster than the CPU versions. In 2008, Zhang et al. [235] proposed a similar method, but included adaptive sampling of fluids to improve the accuracy of the simulation in geometrically complex regions. Authors also visualized the fluid via metaballs and isosurfaces. A similar approach was presented by Yan et al. [236] who, additionally, proposed a generalized distance field function which considers not only geometrical complexity but also physical complexity of fluid body, and new sampling rules for splitting and merging of particles to greatly reduce the computation time of the dynamic fluid simulation. Authors simulated 16000 particles with an average frame rate of 66 fps.

3.7.2 Using specialized compute frameworks

It was until 2010 when Hérault et al. [237] implemented the SPH method on a GPU using the Compute Unified Device Architecture (CUDA) developed by Nvidia. For the neighbor search, authors used the algorithm described in the Particles example of the CUDA toolkit [238]. The computational domain was divided into a grid of linearly indexed cells; each particle was assigned the cell index it lies in as a hash value, and then the particles were reordered by their hash value. The neighbor list for each particle was then built by looking for neighbors only in nearby cells. The list was updated only each 10 iterations, using more memory, but resulting in faster executions. The speedup achieved using a GTX 280 GPU was up to two orders of magnitude faster than the equivalent CPU code for a simulation with 677,360 particles. A similar approach was presented by Gao et al. [239], who reported speedups of up to 140 on a Geforce GTX 480 GPU. Authors tested different block and thread configurations, and found that the correct selection of a a configuration can lead to better GPU utilization. Krog and Elster [240] added support for Newtonian and Non-Newtonian fluids, and achieved speedups of 6 when compared to the work by Yan et al. [236], and up to 91 when compared to a CPU version, simulating 16k particles with a Geforce GTX 470 GPU.

The main disadvantage of the previous grid based approaches was that there was an excess of memory consumption per grid cell for the neighbor search part of the algorithm. Goswami et al. [241] based the neighbor search on Z-indexing and parallel sorting which eliminates GPU memory overhead due to grid or hierarchical data structures, such as buckets. This approach still used texture memory for the indexing, and authors reported faster times than Zhang [235] and Harada [234]. Results presented by Krog and Elster [240] were still faster (around 400 fps for a 16k particle simulation), but the GPU used was a deciding factor for the speedup: Goswami et al. used a Geforce GTX 280, while Krog and Elster used a Geforce GTX 470. The GTX 280 had almost half the compute cores, and around 25% less memory than the GTX 470.

In 2011, Crespo et al. [242] developed a SPH GPU solver named DualSPHysics to deal with free-surface flow problems requiring high computational cost. The code was validated using a dam break impacting with an obstacle. Authors reported results for simulations with one million particles, achieving speedups of up to 64 with a GTX 480 GPU. Test results from a TESLA M1060 GPU, which presented some of the highest computational specifications in terms of memory (4 GB), were still not as efficient as the ones from the GTX 480.

By 2013, Huang et al. [243, 244] presented an efficient GPU-based simulation and rendering framework for large scale SPH fluids. A robust particle classification algorithm was introduced to classify particles into either active or inactive, and reduces the computational burden in inactive areas where many particles with stable properties and low local pressure cluster together. Even though authors used a mobile GPU (Geforce 9500M), real-time processing was achieved for simulations with 30k particles.

Domínguez et al. [245] described different strategies for CPU and GPU optimizations applied to the GPU SPH solver DualSPHysics [242]. Some of the GPU optimizations were described in the CUDA programming guide [246], such as maximizing occupancy and reducing global memory accesses. However authors mentioned other GPU optimizations intrinsic to the SPH method. One of them was a method to simplify the neighbor search by defining ranges of particles for searching decreases the memory accesses and the number of divergent warps. Another was a division of the domain into smaller cells. The procedure consisted in dividing the domain into cells of size $r/2$ instead of size r in order to increase the percentage of real neighbors. The disadvantage was the increase in memory requirements. The optimized GPU version of the code outperformed the GPU implementation without optimizations by a factor on the order of 1.65 using a GTX 480 and 2.15 using a Tesla 1060.

Xiong et al. [247] presented a GPU implementation of Adaptive Particle Splitting and Merging (APS) in the framework of SPH. Particle splitting and merging process were carried out based on a prescribed criterion. Results showed that APS actually achieves comparable accuracy but reduces computational effort considerably. In addition, a single-GPU implementation can give up to a 35 speedup when compared to CPU.

In 2015 Nie et al. [248] presented a parallel framework for simulating incompressible fluids with the Predictive-Corrective Incompressible Smoothed Particle Hydrodynamics (PCISPH) on the

GPU in real-time. Authors proposed an efficient GPU streaming pipeline to map the entire computational task onto the GPU. Specifically, an efficient parallel sorting method for the neighbor search step, which combined the benefits of index sorting [238] and z-index [241], was introduced. Additionally, a Structure of Arrays (SoA) instead of an Array of Structures (AoS) was used to achieve memory coalescing. Simulations achieved a speedup of up to 23 on a GTX 780 GPU in comparison to single-threaded CPU-based implementation.

Joselli et al. [249] introduced a novel and efficient data structure, called neighborhood grid, capable of supporting large number of particle based elements on GPUs, and was used for optimizing SPH fluid animations. The neighborhood grid method implemented with a GTX 580 GPU achieved speedups of up to 9 times when compared to traditional GPU approaches, and up to 100 times when compared against CPU implementations.

In 2016, Xia et al. [250] designed a new GPU-based SPH model for solving the two-dimensional Shallow Water Equations (SWEs) with variable smoothing lengths. Authors implemented a quad-tree neighbour searching method to further optimize the model performance. Compared with the commonly used uniform grid searching method, quad-tree neighbor searching could reduce redundant computation when searching neighbor particles. Because this, the particle interaction, i.e. calculation of water depth and forces, was also accelerated due to better memory coalescing. The combination of these two elements allowed simulations to achieve speedups of up to 2 times against the uniform grid searching based model implemented on GPU. Tests were performed on a Testla M2075 GPU.

3.7.3 Multi-GPU solvers

In 2012, Rustico et al. [251] presented a multi-GPU, CUDA version of the SPH method. Authors extended the work of Hérault et al. [237] to run simulations on multiple GPUs and obtained a gain in speed and overcame the memory limitations of using a single device. The computational domain was spatially split with minimal overlapping and shared volume slices were updated at every iteration of the simulation. The obtained speedup factor differed from the ideal one by a small cost function linear in the number of devices, and it was possible to run simulations with a higher number of particles than would fit on a single device. Authors were able to simulate up to ten million particles using 6 GTX 480 GPUs. The simulation scaled almost linearly with the number of GPUs used.

In 2013, Valdez-Balderas et al. [252] extended the work of DualSPHysics [242, 245] to include computation on multiple GPUs. The approach was based on a spatial decomposition technique, whereby different portions (sub-domains) of the physical system under study were assigned to different GPUs. Communication between devices was achieved with the use of the MPI API. Authors also introduced a radix sort algorithm for inter-GPU particle migration and sub-domain "halo" building, which enabled interaction between SPH particles of different sub-domains. Accelerated simulations were performed with up to 32 million particles on six compute nodes, each hosting two NVIDIA Tesla M2050 GPUs.

Rustico et al. [253] extended, in 2014, their previous work [251] to include several minor optimizations that improve single-GPU performance. The optimizations led to a speed-up of over 2 in single-GPU execution that also reflected on multi-GPU runs. First, an analysis of cache utilization was done on devices belonging to the Fermi architecture, and direct access to the underlying arrays was performed instead of using texture cache, resulting in a more effective use of both texture cache and the L2 cache available on Fermi devices. Second, the layout of the neighbor list was changed: instead of storing the neighbors of the first particle followed by the neighbors of the second particle, authors interleaved neighbors, so that the first neighbor of all particles come first, followed by the second neighbor of all particles, and so on. On GPU, this ensured memory coalescence when particles handled by threads in the same warp traverse the neighbor list. Finally, the neighbor list construction now relied on the thrust library shipped with CUDA.

Chapter 4

Real-time meshless simulation of skeletal muscle

Considering the challenges and issues presented in section 1.2, and the need for efficient and accurate biological tissue simulations, in this work a meshfree method that allows the real-time simulation of biological tissue, specifically, skeletal muscle belly, is presented.

In order to simulate skeletal muscle belly, the use of SPH to simulate tissue as a viscous fluid is proposed. The idea to simulate skeletal muscles as fluids arose because they are composed of 70% water [254], and because most methods consider them as solids and focus only on simulating the external deformations of the material, and not necessarily the entire tissue. For this work, the activation of the tissue itself will dictate how it deforms. Meshfree methods are preferred over FEM because of their flexibility, efficiency, and ease of use.

To get the fluid to conserve its volume, and allow it to behave as a deformable solid, the integration of the fluid with Shape Matching, through a velocity correcting scheme, similar to Takahashi et al. [255], will be used. Furthermore, to activate and deform the tissue, the use of a biophysical model of electrophysiology, particularly, the monodomain model, is proposed. This method was selected because it is computationally less expensive than the bidomain model [93]. Additionally, if we consider the muscle fibers in the simulation, the bidomain equations can be reduced to the monodomain equations [177–179].

Since most works use the FEM to simulate the tissue, the method is also used to solve the electrophysiology needed to deform the tissue. For this work, the use of SPH to solve the monodomain model is proposed. Not only has it not been previously used, to our knowledge, to solve the monodomain model, but its use will also allow for a more efficient use of computational resources, since the calculations will be performed at the same step as the deformable solid computations.

By using the monodomain model, an electric stimulation that innvervates the tissue is used to calculate an electric transmembrane potential. To determine how the stimulation propagates throughout the tissue, the Fitzhugh-Nagumo cell model will be used. Since the electric potential can be considered as pressure in hydraulic systems, the resulting electric potential will be considered in the pressure force calculation of the fluid simulation, and the changes in pressure of the fluid will in turn deform the tissue. Fiber orientation of the tissue will be considered for the deformation of the tissue due to this pressure. With this proposed method, the particles will move from a high pressure area to a low pressure area, and effects such as the bulging of the muscles should be present because of this movement.

In general, the proposed meshfree method consists on the integration of Shape matching for shape definition, deformations, and volume conservation; SPH for the simulation of a viscous fluid that resembles biological tissue; and the solution of the monodomain model using SPH for the activation and control of the tissue. All the mentioned steps will be developed with GPGPU to achieve real-time interactive simulations.

Even though the use of these methods for the simulation of skeletal muscles consider the shape, and the internal functioning of the muscle, they had not been previously integrated together for this specific purpose. Additionally, the simplicity and flexibility of the method is ideal for real-time simulations, which is something that other methods lack. Furthermore, the flexibility of the method could allow it to be used to simulate other biological tissues just by modifying its properties and behaviour defining equations (which in turn could be solved using SPH).

4.1 Proposed architecture

The proposed architecture can be seen in Figure 4.1. In the following sections, each of the blocks will be described in detail. Here, a brief description of each is presented:

Velocity Correction. So that the method achieves the incompressibility and volume preservation of the fluid, a correction of the particle velocities to approximate the dynamics of viscoelastic fluids is calculated first. The velocity correction is based on the Shape Matching scheme [255]. This step considers the position and velocity of the particles, as well as external forces, such as gravity, to project the positions based on their original positions. With those goal positions, a corrected velocity can be calculated, and using the unknown factor (X) SPH method, XSPH [53], an intermediate velocity is obtained.

Calculate Cell Model. A specific cell model which defines how the current propagates in the tissue is calculated. For each of the particles, the Fitzhugh-Nagumo model is solved to obtain the ionic current and the recovery variable. This step does not require the SPH method since the cell model only depends on the transmembrane potential and the recovery variable, and is

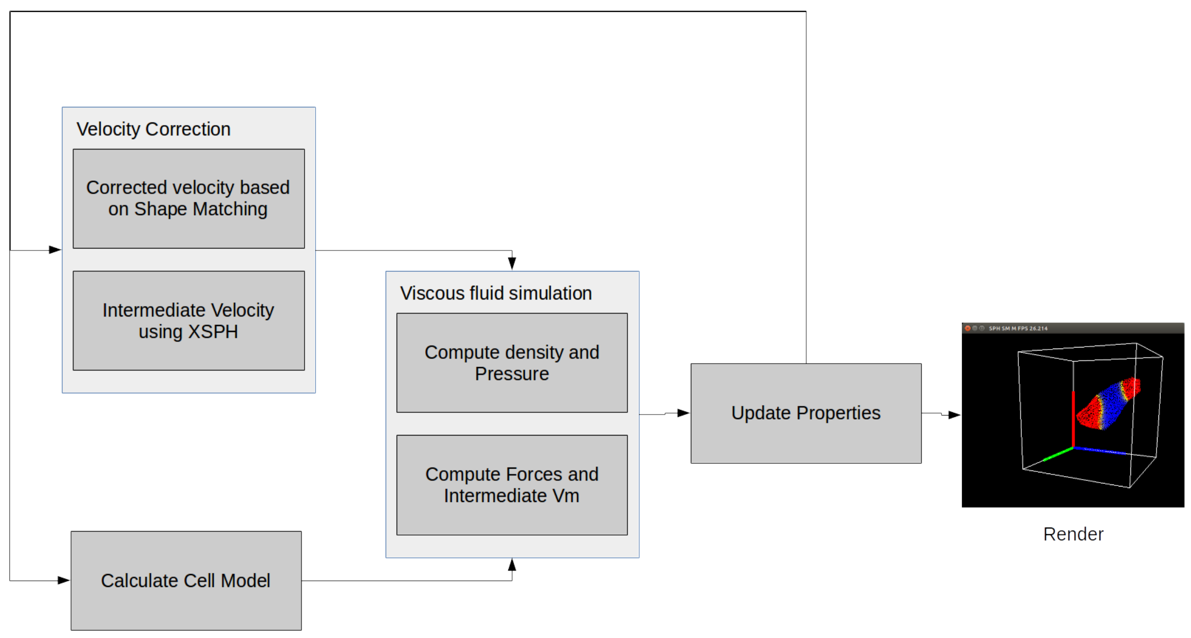

Velocity Correction
Corrected velocity based on Shape Matching
Intermediate Velocity using XSPH
Viscous fluid simulation
Compute density and Pressure
Compute Forces and Intermediate Vm
Update Properties
Calculate Cell Model
Render

FIGURE 4.1: Proposed architecture.

integrated in time with a forward Euler time integration. However, since the proposed solution is a meshfree method, the calculation of the variables and the time integration had to receive a special treatment, which is covered in section 4.3.

Viscous Fluid Simulation. This block is the main SPH viscous fluid simulation. Here, both the fluid and the monodomain properties are calculated. First, the density and pressure of the fluid are calculated, then the acceleration of the particles is obtained, and finally, an intermediate transmembrane potential is calculated. In this step, the pressure is also modified with the transmembrane potential, so that the fluid moves from areas of high pressure to areas of low pressure.

Update Properties. The velocity, position, acceleration, and transmembrane potential are updated using a Forward Euler integration. The velocity is updated using the intermediate velocity and the acceleration. The position is updated using the recently calculated velocity. And, finally, the transmembrane potential is updated using the intermediate transmembrane potential.

Render. Once the position and transmembrane potential of the particles are updated, the values are applied to the particles and a muscle is simulated using OpenGL.

The use of intermediate values throughout these steps is essential because the properties of a given particle, at a determined time step, may still be used by another neighboring particle.

This consideration is even more important because it will prevent potential race conditions since the method will be parallelized.

4.2 Viscoelastic fluid simulation

To simulate volume preserving viscoelastic fluids, it is necessary to separately deal with the volume preservation and with the viscoelasticity. While preserving the fluid volumes by enforcing the incompressibility of fluid using SPH, a velocity correction for viscoelastic effects based on Shape Matching is used.

4.2.1 SPH applied to fluid simulations

The first step to simulate a viscoelastic fluid is to use SPH to solve fluid flow. Most fluid flow is governed by the *incompressible Navier-Stokes* equations [256], a set of partial differential equations that are to hold throughout the fluid. The equations are usually written as:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} + \frac{1}{\rho}\nabla p = \mathbf{g} + v\nabla \cdot \mathbf{u} \tag{4.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{4.2}$$

where $\mathbf{u}$ is the velocity of the fluid, ρ is the density, p stands for pressure, $\mathbf{g}$ is the acceleration due to gravity, v is the kinematic viscosity (how viscous is the fluid). The first equation 4.1 is called the *momentum equation.* It describes how the fluid accelerates due to the forces acting on it. The first of the fluid forces is pressure: high-pressure regions push on lower-pressure regions. The other fluid force is due to viscosity. A viscous fluid tries to resist deforming: a force that tires to minimize differences in velocity between nearby sections of fluid.

When using SPH to solve the Navier-Stokes equations [112], each point in the fluid is labeled as a separate particle, with a position $\mathbf{x}$ and a velocity $\mathbf{u}$. Each particle could be thought of as being a molecule of the fluid. Furthermore, fluids are described by a velocity field $\mathbf{v}$, a density field ρ and a pressure field p. The evolution of these quantities over time is given by two equations. The first equation assures conservation of mass

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0, \tag{4.3}$$

while the Navier-Stokes equation formulates conservation of momentum

$$\rho(\frac{\partial v}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \rho \mathbf{g} + \mu \nabla^2 \mathbf{v} \tag{4.4}$$

where $\mathbf{g}$ is an external force density field and μ the viscosity of the fluid.

The use of particles simplifies these two equations substantially. First, because the number of particles is constant and each particle has a constant mass, mass conservation is guaranteed and Equation 4.3 can be omitted completely. Second, the expression $\partial \mathbf{v}/\partial t + \mathbf{v} \cdot \nabla \mathbf{v}$ on the left hand side of Equation 4.4 can be replaced by the substantial derivative $D\mathbf{v}/Dt$. Since the particles move with the fluid, the substantial derivative of the velocity field is simply the time derivative of the velocity of the particles meaning that the convective term $\mathbf{v} \cdot \nabla \mathbf{v}$ is not needed for particle systems. Using particles, Equation 4.4 can be expressed as

$$\rho_i \mathbf{a}_i = \mathbf{f}_i^{pressure} + \mathbf{f}_i^{external} + \mathbf{f}_i^{viscosity} \tag{4.5}$$

where $\mathbf{a}_i$ corresponds to the acceleration of particle i.

Substituting Equations 2.14 and 2.15 into the pressure and viscosity terms of the Navier-Stokes equation and symettrizing according to Müller et al. [112] yields:

$$\mathbf{f}_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{4.6}$$

$$\mathbf{f}_i^{viscosity} = \sum_j \frac{\mu_i + \mu_j}{2} m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{4.7}$$

where p is the pressure, $\mathbf{v}$ the velocity, m is the mass of the particle, r is the position of the particle, h is the smoothing distance, and μ the viscosity coefficient. Subscripts i and j, represent the current particle, and the neighbor particles, respectively. External forces, such as gravity, have to be considered as follows:

$$\mathbf{f}_i^{external} = \frac{\mathbf{f}}{m_i} \tag{4.8}$$

The pressure p_i of particle i is computed via the modified gas state equation suggested by [106]:

$$p_i = k(\rho_i - \rho_0) \tag{4.9}$$

where ρ_0 is the rest density, and k is a stiffness constant that scales the pressure, and, thus, the pressure gradient and the respective pressure forces [56].

The equations are integrated in time using a forward Euler method:

$$y_{n+1} \approx y_n + \Delta t f(y_n, tn) \tag{4.10}$$

where y_n is an approximation of the function f at time n, and Δt is the difference in time. Applying equation 4.10 the velocity $\mathbf{v}$ and position $\mathbf{r}$ of the particles can be calculated:

$$\begin{aligned} \mathbf{v}_{n+1} &= \mathbf{v}_i + \Delta t \, \frac{\mathbf{a}_n}{m_i} \\ \mathbf{r}_{n+1} &= \mathbf{r}_i + \Delta t \, \mathbf{v}_{n+1} \end{aligned} \tag{4.11}$$

The time step size Δt that will be used must be adapted by the Courant-Freidrich-Levy (CFL) condition [53, 257]:

$$\Delta t \leq 0.4 \frac{d}{||v_{max}||} \tag{4.12}$$

where d is the particle diameter, and v_{max} is the maximum particle velocity.

4.2.1.1 SPH smoothing kernels for fluid simulations

Stability, accuracy and speed of the SPH method highly depend on the choice of the smoothing kernels. For fluid simulations Müller et al. [112] designed the following kernel

$$W_{poly6}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

and use it in all but two cases. An important feature of this simple kernel is that r only appears squared which means that it can be evaluated without computing square roots in distance computations. However, if this kernel is used for the computation of the pressure forces, particles tend to build clusters under high pressure. As particles get very close to each other, the repulsion force vanishes because the gradient of the kernel approaches zero at the center. Desbrun [106] solves this problem by using a spiky kernel with a non vanishing gradient near the center. To generate necessary repulsion forces in pressure computations, Debrun's spiky kernel is used:

$$W_{spiky}(\mathbf{r},h) = \frac{45}{\pi h^6}\begin{cases}(h-r)^2 & 0 \le r \le h \\ 0 & \text{otherwise}\end{cases} \tag{4.14}$$

Viscosity is a phenomenon that is caused by friction and, thus, decreases the fluid's kinetic energy by converting it into heat. Therefore, viscosity should only have a smoothing effect on the velocity field. However, if a standard kernel is used for viscosity, the resulting viscosity forces do not always have this property. For two particles that get close to each other, the Laplacian of the smoothed velocity field (on which viscosity forces depend) can become negative resulting in forces that increase their relative velocity. The artifact appears in coarsely sampled velocity fields. In real-time applications where the number of particles is relatively low, this effect can cause stability problems. For the computation of viscosity forces, Müller et al. [112] proposed a third kernel:

$$W_{viscosity}(\mathbf{r},h) = \frac{15}{2\pi h^3}\begin{cases}-\frac{r^3}{2h^3}+\frac{r^2}{h^2}+\frac{h}{2r}-1 & 0 \le r \le h \\ 0 & \text{otherwise}\end{cases} \tag{4.15}$$

4.2.2 Velocity Correction

The velocity correction step is included before the iterations of SPH. The algorithm used is the following:

Algorithm 1 Viscous Fluid with Velocity Correction

```
1: Find neighboring particles
2: Correct particle velocity
      2.1 Compute predicted velocity $\mathbf{v}^{adv}$.
      2.2 Get corrected velocity $\mathbf{v}^*$.
3: Obtain intermediate velocity $\tilde{\mathbf{v}}$ using XSPH, which modifies particle velocities without affecting the convergence of the fluid solver.
4: Calculate pressure and viscosity forces using $\tilde{\mathbf{v}}$
5: Update acceleration, velocity, and position
```

Particle velocities are corrected to describe viscoelastic effects without changing particle positions. In order to obtain intermediate velocity $\tilde{\mathbf{v}}_i$ for particle i as an input for SPH, the predicted velocity $\mathbf{v}^{adv}$ is computed first with all forces $\mathbf{F}_i^{adv}$ (external and gravity) excluding viscoelastic and pressure ones:

$$\mathbf{v}_i^{adv} = \mathbf{v}_i + \Delta t \frac{\mathbf{F}_i^{adv}}{m_i} \tag{4.16}$$

where m_i is the mass of each particle. Then, the particle's velocities are corrected with the velocity correction vector $\Delta\mathbf{v}_i$ (which is obtained via a Shape Matching Scheme):

$$\mathbf{v}_i^* = \mathbf{v}_i^{adv} + \Delta\mathbf{v}_i \tag{4.17}$$

Finally, XSPH is used to reduce particle oscillations with $\mathbf{v}_{ij}^* = \mathbf{v}_i^* - \mathbf{v}_j^*$, and a velocity mixing parameter $\varepsilon(0 \leq \varepsilon \leq 1)$:

$$\tilde{\mathbf{v}}_i = \mathbf{v}_i^* + \varepsilon \sum_j \frac{m_j}{\rho_j} \mathbf{v}_{ij}^* W_{ij} \tag{4.18}$$

4.2.2.1 Shape Matching Scheme

In SM, particle i is pulled toward its goal position $\mathbf{g}_i$ to restore the original configuration of the particles, and individual goal positions are computed to match the original configuration of the particles defined by $\mathbf{x}_i^0$ with current particle distributions denoted as $\mathbf{x}_i$ after the particles are transferred.

In this case, a rotational matrix $\mathbf{R}_i$, and the translation vectors $\mathbf{t}$ and $\mathbf{t}_0$ have to be calculated. These should minimize:

$$\sum_i w_i(\mathbf{R}(\mathbf{x}_i^0 - \mathbf{t}_0) + \mathbf{t} - \mathbf{x}_i)^2 \tag{4.19}$$

where the w_i are weights of individual points (the natural choice for the weights is $w_i = m_i$. The optimal translation vectors turn out to be the center of mass of the initial shape and the center of mass of the actual shape, i.e.

$$\begin{aligned} \mathbf{t}_0 = \mathbf{x}_{cm}^0 = \frac{\sum_i m_i \mathbf{x}_i^0}{\sum_i m_i} \\ \mathbf{t} = \mathbf{x}_c m = \frac{\sum_i m_i \mathbf{x}_i}{\sum_i m_i} \end{aligned} \tag{4.20}$$

which is physically plausible. To find the relative locations $\mathbf{q}_i = \mathbf{x}_i^0 - \mathbf{x}_{cm}^0$ and $\mathbf{p}_i = \mathbf{x}_i - \mathbf{x}_{cm}$ of points with respect to their center of mass are defined, and the problem of finding the optimal rotation matrix R is relaxed to finding the optimal linear transformation $\mathbf{A}$. Now, the term to be minimized is $\sum_i m_i(\mathbf{Aq}i - \mathbf{p}_i)^2$. Setting the derivatives with respect to all coefficients of $\mathbf{A}$ to zero yields the optimal transformation

$$\mathbf{A} = (\sum_i m_i \mathbf{p}_i \mathbf{q}_i^T)(\sum_i m_i \mathbf{q}_i \mathbf{q}_i^T)^{-1} = \mathbf{A}_{pq}\mathbf{A}qq \tag{4.21}$$

The second term $\mathbf{A}_{qq}$ is a symmetric matrix and, thus, contains only scaling but no rotation. Therefore, the optimal rotation $\mathbf{R}$ is the rotational part of $\mathbf{A}_{pq}$ which can be found via a polar decomposition $\mathbf{A}_{pq} = \mathbf{RS}$, where the symmetric part is $\mathbf{S} = \sqrt{\mathbf{A}_{pq}^T \mathbf{A}_{pq}}$ and the rotational part is $\mathbf{R} = \mathbf{A}_{pq}\mathbf{S}^{-1}$. Finally, the goal positions can be computed as

$$\mathbf{g}_i = \mathbf{R}(\mathbf{x}_i^0 - \mathbf{x}_{cm}^0) + \mathbf{x}_{cm} \tag{4.22}$$

With the goal positions, the velocity correction vector $\Delta \mathbf{v}_i$ can be constructed:

$$\Delta \mathbf{v}_i = \alpha \frac{\mathbf{g}_i - \mathbf{x}_i}{\Delta t} \tag{4.23}$$

where $\alpha = [0...1]$ is a parameter which simulates stiffness.

4.3 Activating and deforming the muscle

The monodomain model describes the propagation of electrical activity in biological tissue, while the cell model describes the interaction of ionic concentrations at cellular levels. For the case of the monodomain model, the term I_{ion} has to be calculated with an specific cell model that is able to reproduce the tissues' behavior. In order to control the activation of the muscles, the use of the Fitzhugh-Nagumo model is proposed; not only due to its mathematical simplicity and its richness from a point of view of system dynamics, but also because of its correlation to the Hugh-Huxley model.

4.3.1 Meshfree Cell model

The FitzHugh-Nagumo model uses a cubic polynomial to model excitation, but also a recovery variable so both depolarization and repolarization can be modelled [258]. The model's potential values are normalized to lie between zero and one. The normalized transmembrane potential is denoted by v and is calculated by:

$$v = \frac{V_m - V_r}{V_p - V_r} \tag{4.24}$$

where V_p is the plateau potential, V_r is the resting potential, and V_m is the transmembrane potential. A cubic polinomial is used to describe the course excitation:

$$\begin{aligned} I_{ion} &= C_1\ v\left(v - \frac{V_{th} - V_r}{V_p - V_r}\right)(v-1) + C_2\ w \\ \frac{dw}{dt} &= b(v - d\ w) \end{aligned} \tag{4.25}$$

where I_{ion} is the Ionic current, C_1 is an excitation rate constant, C_2 is an excitation decay constant, and V_{th} is the threshold potential. The variable w is a dimensionless time-dependent recovery variable that represents the sodium gating variable. Here b is a recovery rate constant, and d is a recovery decay constant.

When using a meshfree approach to calculate the cell model, each of the particles of the system has an independent property, and to update them, the forward Euler method and the CFL condition were used:

$$v_{n+1} = \frac{V_{m_n} - V_r}{V_p - V_r} \tag{4.26}$$

$$I_{ion_{n+1}} \approx I_{ion_n} + \left(\frac{\Delta t}{m_i} C1\ v_{n+1}\left(v_{n+1} - \frac{V_{th} - V_r}{V_p - V_r}\right)(v_{n+1} - 1) + C_2\ w\right) \tag{4.27}$$

$$w_{n+1} \approx w_n + \frac{\Delta t}{m_i}\, b\ (v_{n+1} - d\ w_n) \tag{4.28}$$

4.3.2 Monodomain solved with SPH

The numerical solution of the bidomain equations is usually calculated using FEM. However, various meshless methods have demonstrated the ability to provide a computational feasible model for cardiac electrophysiology simulations, without the burden of mesh generation [97, 183, 259]. In this work, SPH is proposed to numerically solve the monodomain model of electrophysiology.

The monodomain equation 2.7 can be written as:

$$\frac{\partial V_m}{\partial t} = \frac{1}{C_m}\left(\frac{1}{\chi}\left(\nabla \cdot \sigma \nabla V_m\right) - I_{ion} + I_{ext}\right) \tag{4.29}$$

where I_{ext} represents the stimulus current that is applied to the tissue, and I_{ion} represents the cell model current specific to the biological tissue being simulated. Since the macroscopic electrical conductivity of muscle tissue perpendicular to the fiber direction is up to one magnitude lower than the conductivity along the fiber direction [178, 179], and electrical stimulation from one fiber to adjacent ones is not observed, the propagation of a potential along a skeletal muscle fiber is modeled as a 1D system. In this case, the intracellular and extracellular conductivity tensors are scalars when solving the monodomain equations. Therefore,

$$\nabla \cdot \sigma \nabla V_m = \sigma \nabla^2 V_m \tag{4.30}$$

Applying the SPH formulation of Equation 2.12 to the rewritten monodomain equation 4.29, using XSPH to reduce particle oscillations with $\varepsilon = 1$, and considering the simplification of Equation 4.30, yields:

$$\frac{\partial V_m}{\partial t} = \frac{1}{C_m}\left(\frac{\sigma}{\chi}\left(\sum_j m_j \frac{V_{m,j} - V_{m,i}}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h)\right) - I_{ion} + I_{ext}\right) \tag{4.31}$$

The cubic B-spline kernel 2.20 was used here since it has compact support, and the second derivative is continous [53]. The compact support means that interactions are exactly zero for $r > 2h$, and the continuity of the second derivative means that the kernel is not sensitive to disorder. The second derivative of the kernel, which is needed for to solve Equation 4.31 is the following:

$$W''(\mathbf{r} - \mathbf{r}_j, h) = \frac{\alpha_d}{h^n}\begin{cases} -3 + \frac{9}{2}q & 0 \leq q < 1 \\ \frac{3}{2}(2-q) & 1 \leq q < 2 \\ 0 & otherwise \end{cases} \tag{4.32}$$

Regarding the time integration scheme, a forward Euler method and the CFL condition were used.

To apply time integration to equation 4.31, it is first needed to calculate an intermediate potential, V_m^* as the sum of the contribution of the potentials from the neighboring particles:

$$V_{m,i}^* = \sum_j m_j \frac{V_{m,j} - V_{m,i}}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{4.33}$$

and then multiply time dependent variable, in this case I_{ext} by the Δt over mass, before using it to update the final potential:

$$V^*_{m,i_{n+1}} \approx V^*_{m,i_n} + \frac{1}{C_m}\left(\frac{\sigma}{\chi}\, V^*_{m,i_n} - \left(I_{ion} - I_{ext}\,\frac{\Delta t}{m_i}\right)\right) \tag{4.34}$$

$$V_{m_{n+1}} \approx V_{m_n} + \frac{\Delta t}{m_i}\, V^*_{m_{n+1}} \tag{4.35}$$

4.3.3 Muscle deformation

Once the muscle is activated, and a transmembrane potential is calculated, a way to deform the muscle is needed. Muscle cells resemble nerve cells in their ability to conduct action potentials along their membrane surfaces. In addition, muscle cells have the ability to translate the electrical signal into a mechanical contraction, which enables the muscle cell to perform work [94].

Since the muscle bellies are going to be simulated using highly viscous fluids, and considering that transmembrane potentials can be considered as electrical pressure, in order to deform the muscles, the calculated transmembrane potentials are added to the pressure force term of the fluid simulation. Since the fluid moves from areas of high pressure to areas of low pressure, by decreasing the pressure in specific areas, the tissue is forced to move in a given direction, essentially creating contractions in the muscle. After calculating the transmembrane potential, it is integrated into the pressure term as follows:

$$p_i = k(\rho_i - \rho_0) - V_{m_i} \tag{4.36}$$

This effect is caused when a stimulus current I_{ext} is applied to specific sections of the tissue: when the current is applied, the transmembrane potential increases, and thus the pressure is decreased, creating a contraction in a given point of the tissue. When the stimulus current is removed, the transmembrane potential decreases and the pressure begins to increase, relaxing the muscle and allowing it to return to its original, uncontracted, shape.

To consider muscle fiber orientation for the direction of the tissue deformation, the force of pressure had to be adjusted. Equation 4.6 was adjusted to consider fiber orientation as follows:

$$\mathbf{f}_i^{pressure} = -\tau_i \sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{4.37}$$

where τ_i is a vector that represents the normalized direction of the contraction or expansion at particle i. This vector is calculated by considering the unit vector of the relative velocity and the direction vectors of the fibers, which are added, and then weighted by the magnitude of the relative velocity. By considering this vector at the pressure calculation, the areas of higher pressure will be determined by the fiber orientation.

4.3.4 Muscle tissue properties

Incorporating the skeletal muscle's properties is fundamental for the correct functioning of the model. For the case of the proposed cell model used, the values need to correctly simulate the current propagation throughout the tissue. The values proposed for the monodomain model are based on the work of Röhrle et al. [34]; for the cell model used, the parameters are based on Nickerson [89]. These values can be seen in Table 4.1.

Variable	**Value**	**Description**
χ	500 cm^{-1}	Surface-to-volume ratio
C_m	1.0 $\mu F/cm^2$	Membrane capacitance
σ_i	0.893 mS/mm^{-1}	Internal fiber conductivity
σ_e	0.67 mS/mm^{-1}	External fiber conductivity
I_{ext}	8000 $\mu\text{A/mm}^2$	External stimulus current
V_r	-85.0 mV	Resting Potential
V_p	15.0 mV	Plateau Potential
V_{th}	-75.0 mV	Threshold Potential
C_1	0.175 $\mu\text{A / mm}^{-2}$	Excitation rate constant
C_2	0.03 $\mu\text{A / mm}^{-2}$	Excitation decay constant
b	0.011 ms^{-1}	Recovery rate constant
d	0.55 ms^{-1}	Recovery decay constant

TABLE 4.1: Values for the cell model, and the monodomain equations.

For the density of the muscle belly, muscle architecture reports typically do not directly measure muscle density [260]. Instead, several studies [74, 78, 261, 262] use the value 1.0597 g/cm^3, which was derived from unfixed rabbit and canine muscle tissue [263]. Given the fact that human muscle architecture is often characterized in formaldehyde-fixed tissue, the previous value was inaccurate for several reasons. First, a species effect may exist so that rabbit or canine muscle density may differ from human muscle density. Second, the method and duration of fixation may cause shrinkage and thus dehydration, which may alter muscle density. In this work, the value of 1.112 g/cm^3 for muscle density is used, following the recommendations of Ward et al. [260] who determined, through various experiments, that this was the correct value. As for the viscosity of the fluid, a coefficient of 15 Nm^{-1}s [264] was selected.

4.4 Integration of SM, SPH, and Monodomain

With the viscous fluid simulation, and with the monodomain solver using SPH, the next step is to integrate them into a single simulation. With the transmembrane potential calculated using the monodomain model, the pressure force of the fluid was altered in order to simulate a contraction of the muscles. Since several steps of the model rely on SPH to calculate different properties, special care was taken to use the cycles of the SPH algorithm as best as possible. For this work, three SPH cycles were needed: to calculate the intermediate velocity; to calculate the pressure and density; and to calculate the pressure and viscosity forces, as well as the transmembrane potential.

The proposed algorithm to integrate these models into one that simulates tissue, is the following:

Algorithm 2 Simulate Tissue using CPU

```
1: Load the geometry of interest
2: while Simulate do
3:   Find neighboring particles
4:   Correct particle velocity
       4.1  Compute predicted velocity $\mathbf{v}^{adv}$.
       4.2  Get corrected velocity $\mathbf{v}^{*}$.
5:   Obtain intermediate velocity $\tilde{\mathbf{v}}$
6:   Calculate the cell model.
7:   Compute the density and pressure of the fluid. The transmembrane potential is added to
     the pressure in this step.
8:   Calculate pressure and viscosity forces; was well as the intermediate transmembrane po-
     tential.
9:   Update acceleration, velocity, position, and transmembrane potential.
10: end while
```

4.4.1 GPU considerations

While the CPU implementation of the proposed algorithm was straightforward and had no additional requirements, the GPGPU version of the SPH algorithm did have additional constraints regarding how data was managed and processed. The Shape Matching section of the algorithm had no special considerations, and was “Embarrassingly Parallel”, ie. the same tasks were performed for each particle, and no special considerations or algorithms had to be considered besides thread synchronization.

Since the SPH method is going to be used to solve both the viscous fluid dynamics, as well as the monodomain model, special attention will be given to its implementation on a GPU. The main bottleneck for the parallelization of the method is the neighbor search step. For this work, the approach taken by several authors [238, 239, 265] will be used.

4.4.1.1 Particle Interactions

It is relatively simple to implement a particle system where particles do not interact with each other. In this case each particle is independent and they can be simulated trivially in parallel. However, for local interactions, such as modifying the properties of the particles using SPH, performance can be improved by using spatial subdivision. The potentially interacting partners of a particle i only need to be searched in i's own cell and all the neighboring cells within a certain radius. This technique reduces the time complexity of the force computation step from $O(n^2)$ to $O(nm)$, m being the average number of particles per grid cell. This technique divide the simulation space so that it is easier to find the neighbors of a given particle.

For this work, the use of a uniform grid, which is the simplest possible spatial subdivision, is used. A uniform grid subdivides the simulation space into a grid of uniformly sized cells. For simplicity, a grid where the cell size is the same as double the size of the particle radius is defined. This means that each particle can cover only a limited number of grid cells (8 in 3 dimensions).

A so-called "loose" grid is used, where each particle is assigned to only one grid cell based on its center point. This method allows the particles to be stored into the grid cells simply by sorting them by their grid index. The grid data structure is generated from scratch each time step. It is possible to perform incremental updates to the grid structure on the GPU, but this approach is simple and the performance is constant regardless of the movement of the particles.

To build the grid parallel sorting is used. The algorithm consists of several GPU kernels. The first kernel "calcHash" calculates a hash value for each particle based on its cell id. For this implementation, the linear cell id is used as the hash, but it may be beneficial to use other functions such the Z-order curve [241] to improve the coherence of memory accesses. The kernel stores the results to the "particleHash" array in global memory as a pair (cell hash, particle id).

Then the particles are sorted based on their hash values. The sorting is performed using the CUDA Thrust library. This creates a list of particle ids in cell order. In order for this sorted list to useful, the start of any given cell in the sorted list has to be found. This is achieved by running another kernel "findCellStart", which uses a thread per particle and compares the cell index of the current particle with the cell index of the previous particle in the sorted list. If the index is different, this indicates the start of a new cell, and the start address is written to another array using a scattered write. The end of each cell is found in a similar way.

Once the grid structure is built, it is used to accelerate particle-particle interactions, and interpolate the different properties of the model with SPH.

4.4.1.2 Data arrangement

A simple requirement to produce a better performing solution is to arrange data in a SoA manner to achieve memory coalescing. Each property of the simulation, from the position of the particles to their transmembrane current, were assigned to a one-dimensional array (the arrays were *device arrays*, ie., arrays whose data is only available on the graphics card itself), and the simulation data was stored sequentially.

In order to implement the sort and reordering algorithm of Section 4.4.1.1, additional device arrays were required to store the sorted properties, and the arrays that store the indices, hashes, and the start and end hashes of the neighboring particles. However, the only additional arrays needed were those whose properties were needed for SPH steps, such as the position or the corrected velocity.

GPGPU simulations require a launch configuration of a specific number of blocks and threads, and have to be considered when implementing the proposed algorithm. Since all the steps of the algorithm have to be applied to each particle of the system, and since the particle's data were stored in one dimensional arrays, a configuration of one dimensional blocks of one dimensional threads was used. Using two, or even three, dimensional blocks or threads does not necessarily expose more parallelism for this simulation, and the additional management needed hindered the performance. A fixed number of threads for each block of 512 was proposed, and the number of blocks needed to process all the particles was calculated at runtime. The 512 number of threads was selected from a series of power of two values after determining that it yielded the best occupancy and execution times for this solution.

4.4.1.3 GPGPU algorithm

Finally, some adjustments to algorithm 2 had to be made to consider the GPU changes, specifically to determine the neighbors of the particles. The final algorithm, which was used to simulate the skeletal muscle, is the following:

Algorithm 3 Simulate Tissue using GPU

```
1: Load the geometry of interest to device arrays
2: while Simulate do
3:     Correct particle velocity.
         3.1  Compute predicted velocity $\mathbf{v}^{adv}$
         3.2  Get corrected velocity $\mathbf{v}^{*}$
4:     Calculate the cell model
5:     Find neighboring particles
         5.1  Calculate the hash of each particle
         5.2  Sort the particle's index based on their hash
         5.3  Reorder the system's data based on the sorting
6:     Obtain intermediate velocity $\tilde{\mathbf{v}}$
7:     Compute the density and pressure of the fluid; the transmembrane potential is added to
       the pressure in this step
8:     Calculate pressure and viscosity forces; was well as the intermediate transmembrane po-
       tential
9:     Update acceleration, velocity, position, and transmembrane potential
10: end while
```

Chapter 5

Implementation and experimental results

The methods proposed in Chapter 4 were applied to develop a simulation of the long head of the triceps brachii and the vastus lateralis. This chapter will describe all the different components of the simulation in detail, as well as the experiments that were conducted to test the accuracy and performance of the model. The focus of the simulations were the contraction and expansion of the muscle activated by the biophysical model, giving special attention to the displacement, pressure, and processing time. Unless explicitly stated, all the simulations presented here were developed using C++, and rendered using OpenGL. All the developed code will be open source, and available at [266] for the GPGPU version, and at [267] for the CPU version.

5.1 Experimental setup

For the SPH method, 2 types of simulations were developed: one using only the CPU for processing, without any parallel processing; another using GPGPU. For the FEM simulation, only a CPU version was developed. Each of the simulations consisted on two phases: one where specific parts of the tissue were innervated with a stimulus current, and one where the stimulus current was removed. The purpose of the first phase was to test the effects of the stimulation current on the tissue, and try to simulate a contraction on the tissue. The second phase would allow the tissue to return to its original shape. Each phase ran for 250 time-steps. The simulations were tested using the following setup, as shown in Table 5.1:

Component	**Specification**
Processor	12x Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz
Memory	16 GB
Operating System	Ubuntu 16.04.3 LTS
GPU	GeForce GTX TITANX 3072 CUDA Cores @1.08 GHz, 12GB Memory
GCC version	5.3.1
OpenGL version	4.5

TABLE 5.1: Specification for the computer where the experiments were conducted.

5.1.1 Model evaluation and validation

Although there are several benchmark tests [268–271] that are widely accepted by the community for computational fluid dynamics solutions, such as the 3D dam break [272] or 3D flow around a cylinder [273], most focus only on fluid or gaseous problems. Even for biological or medical problems, available benchmarks [274, 275] only focus on problems such as blood flow. Even though there are some benchmarks [276, 277] for soft solid simulations, currently, there are no widely available or recognized benchmarks for skeletal muscle simulations. Models such as Röhrle et. al. [34] or Millard et. al. [278] are validated by comparing specific muscles and movements to other models, or to whatever tissue data is available: in some cases, human data, but most commonly, it is data obtained from animal tissue, such as from rats.

To evaluate the accuracy of the proposed SPH-based model, another model that uses FEM was simulated for the same geometry, and results were compared for the displacement of the geometry, and the pressure generated by the tissue. The root mean squared error (RMSE) was used to compare the results of both methods. The error is defined by

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2} \tag{5.1}$$

where $\hat{y}$ is the value predicted by the SPH method, and y is the value predicted by the FEM method. For the displacement, the resulting deformed mesh by both methods is compared. For the pressure, the total average pressure of the tissue by each method was calculated and then compared.

5.1.2 Muscle geometry

In order to simulate a muscle, the need for an anatomically correct representation of it is needed. The muscle geometry for the proposed simulation was obtained from the BodyParts3D database [279], a dictionary-type database for anatomy in which anatomical concepts are represented by 3D structure data that specify corresponding segments of a three-dimensional whole-body model for an adult human male.

5.2 Geometry preparation

The geometric model that was used only represents the outer shell of the tissue: there are no internal particles. Those are needed because the viscous fluid simulated with SPH depends on a set of particles of the whole tissue. To create the additional points needed for the simulation, the following algorithm was proposed:

Algorithm 4 Create inner particles

```
1: Load the muscle geometry, and orient it along the X axis.
2: Create empty clusters of a given width along the X axis.
3: for all particles do
4:     Assign the particle to a specific cluster based on the particle's $x$ position.
5: end for
6: for all clusters do
7:     Calculate its centroid. (This creates a "central skeleton" for the tissue.)
8: end for
9: for all particles, all centroids do
10:    Add new particles between the particle and the centroid. Each new particle is separated
       by a specified distance.
11: end for
```

This algorithm adds additional particles which are aligned to the "central skeleton". By adding more particles, more detail and accuracy can be achieved at the cost of more computational time. Figure 5.1 shows a 3D model with the additional generated particles. The red particles represent the original data, the green points are the ones that were added, and the yellow points represent the "central skeleton" that was calculated. The spacing between each of the particles was 0.024 for the muscle geometry. To facilitate the use of this data, the geometry was scaled so that the range of its positions are between 0 and 1.

For particle numbers, the values 2231, 4944, 9888, and 18475 were selected based on the number of particles of the muscle model. For the core radius, the values of 0.02, 0.04, and 0.08 were selected based on the separation of the particles of the models. For the cell sizes, the values of 0.02, 0.04, 0.08, and 0.16 were selected in order to test the effect of having more or less particles in each cell, and thus the increase or decrease in their approximation for a given property of a particle.

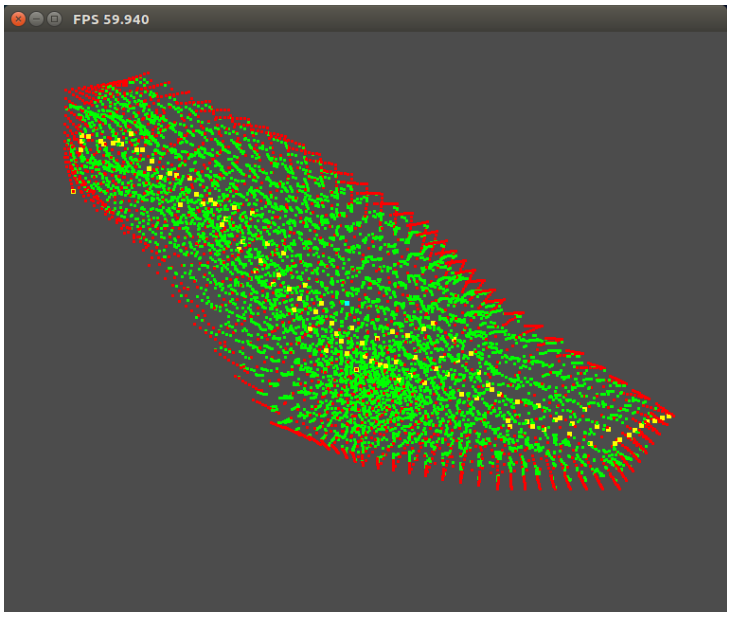


FIGURE 5.1: The muscle geometry with the added points.

5.2.1 Fiber generation

Fiber directions for the muscle were calculated as follows: each fiber was uniformly aligned and parallel to direction vectors from the "central skeleton". To get the direction for each point of the skeleton, the points were processed from left to right, and a direction vector was calculated by subtracting one from the other, and normalizing the result. The vectors were oriented as to follow a line of action from the origin to the insertion of the muscle.

5.3 Viscous fluid with a velocity correction scheme

In order to simulate the tissue, two steps were needed: a viscous fluid had to be simulated, and the shape of the muscle had to be considered.

For the viscous fluid part of the simulation, a SPH fluid simulation with $k = 0.8$, $\rho_0 = 1{,}112$ kg/m^3, and $\mu = 15$ Nm^{-1}s was developed. The mass of each particle was computed as the product of the density times the volume of the cubic cell defined between the particle of interest and the neighboring particles. To optimize the neighbor search step of the algorithm, a grid of size $2h$ was selected. The core radius h value of the smoothing kernel functions was set to 0.04, while the cell size was set 0.08. The choice of a cell size double the size of the core radius was because of stability concerns: for the complex geometry of the muscle tested, if the cell size was the same as the radius, the particles would begin to vibrate and eventually collapse (by breaking the geometry and moving erratically throughout the simulation space). While the added size for each cell increases the amount of particles that contribute to the approximation, and thus a more accurate value, of a property, the computational time is also increased.

The velocity correction step mentioned in Section 4.2 was implemented to integrate these two simulations. In this case, the goal positions for the object were obtained first, so that a corrected velocity could be calculated and later used in the SPH viscous fluid simulation.

5.4 Solving the monodomain equation with SPH

The monodomain model is usually solved with the FEM or with FDM, with their respective advantages and disadvantages. Here, a solution with FD, using central difference approximation, was implemented in 2D to serve as a benchmark for the solution of the model with SPH. The simulation space was a square of side 1, divided into cells of size 0.01. The FitzHugh-Nagumo cell model was used for the simulation. A forward Euler method was used for time integration, with Δt set to 0.05, and a total of 100 time-steps were calculated. The conductivity tensor σ was set to 1 for simplicity. The simulation steps were calculated using C++ and plotted with Python using Matplotlib.

To solve the monodomain model with SPH, several considerations had to be taken:

- Solutions of the monodomain model with methods such as FEM do not consider the mass or the density of the material, only the cell model. In this case, a mass of 0.2 was arbitrarily selected; different mass values produced similar results.
- The size of the core radius h was set experimentally as 0.02, with the cell size set as 0.04. The selection of double the core radius for the cell size was because of stability concerns, similar to what happened for the viscous tissue simulation, if a cell was the same size as the radius, the simulation would produce incorrect values and eventually collapse (by calculating Not a Numbers (NaN)). The size of the radius considerably impacts the performance and accuracy of the solution: the larger the radius and the cell size, the lower the accuracy of the solution; but if the radius was lower than 0.02, the performance was considerably lower: simulations took double the time to compute, in average.
- The selection of a smoothing kernel function was a determinant factor to obtaining correct results. Several smoothing kernels were tested, including the kernels for fluid simulations 4.2.1.1, and the Wendland kernel, since it was also used for electronic calculations. However, the kernel that yielded the most similar results to benchmark simulation was the cubic B-spline kernel 2.4.4.1.

- When approximating the transmembrane potential using SPH, it was also needed to store the approximation in a separate variable. This intermediate value was later used to update the transmembrane potential.
- When using SPH to solve a model, time integration is usually applied after the properties of the model have been approximated. However, in this case the time integration also had to be applied when calculating the cell model, and when calculating the intermediate transmembrane potential. Additionally, the time step had to be normalized by the mass of each particle to ensure a stable simulation.

Figure 5.2 shows a comparison of the monodomain model solved with both FD and with SPH. The solution with SPH presented an average difference of 10% when comparing the values of each particle with the values of the cells for the FD solution. Further tests, with different configurations are needed before a conclusion regarding the accuracy of the proposed method can be achieved. However, considering that the FD solution with central differences only considers the cells at the top, right, left, and bottom, of a given cell for the approximation of the property, and that with SPH several more particles are considered for the same approximation, it is likely that the model with SPH yields a better approximation than with the FD method. This assumption is also based on the fact that other studies have already found that SPH is an adequate alternative solution method for different problems [55, 117, 280–282]. An added benefit of using SPH is that it can be easily used to solve 3D models by also considering the Z axis when looking for neighbor particles, instead of just the X and Y axis.

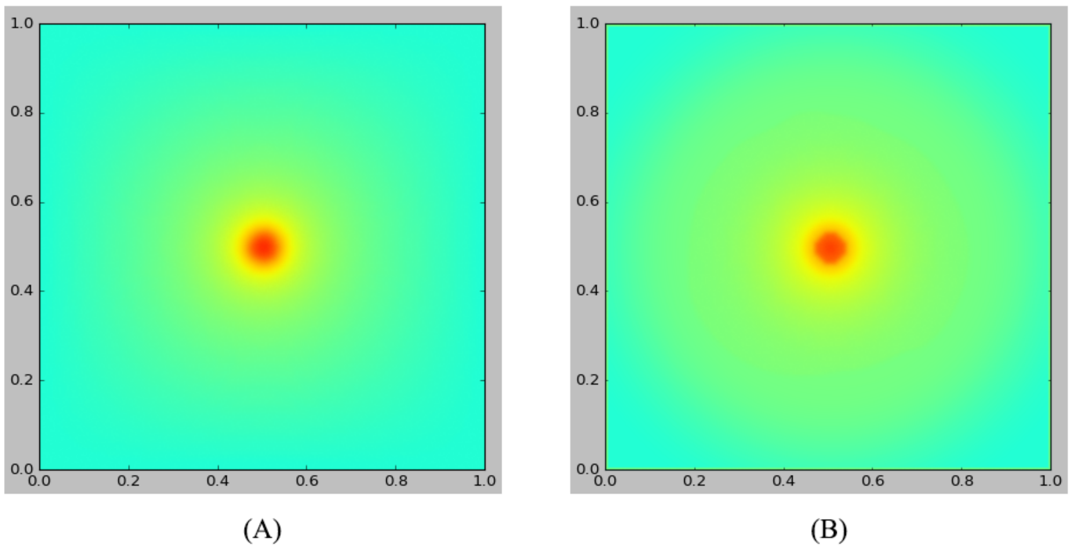


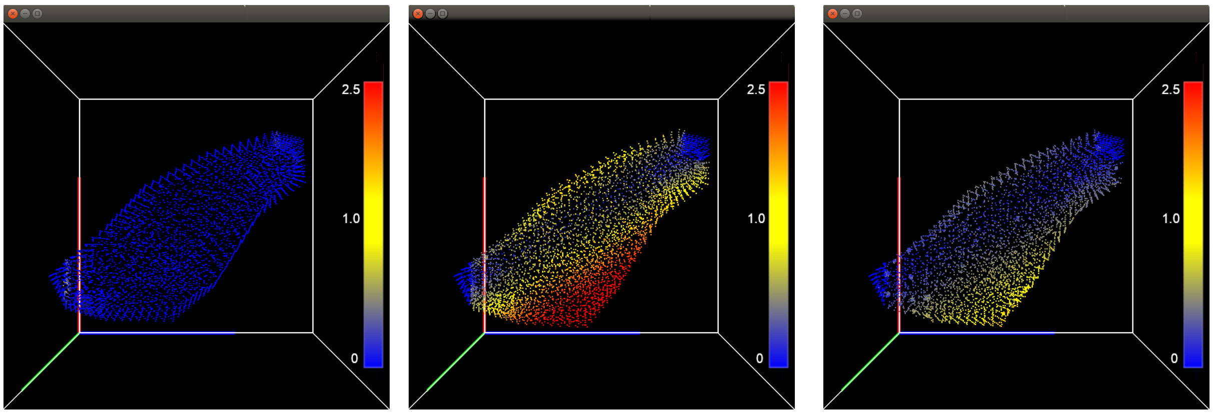
Figure 5.2: Monodomain model solved with FD (A), and solved with SPH (B).

5.5 Tissue simulation on muscle geometry

The ends of the muscle geometry were fixed in space to emulate the origin and insertion of the muscle belly.

For the first phase, the stimulation current was applied to all the particles for 250 time-steps. The transmembrane potential of each particle gradually increased as the Ionic current was propagated throughout the tissue. The particles moved towards regions of lower pressure, and the contraction and bulging of the tissue was visible. For second phase, the particles that had been innervated had the stimulation current removed and the tissue was allowed to return to its initial configuration. Figure 5.3 shows the simulation using the 18475 particle resolution.

Figure 5.3 shows the displacement in millimeters of the triceps using the 18475 particle resolution, while Figure 5.4 shows the displacement for the vastus lateralis with the same particle resolution. A video with the simulation of the triceps with particles can be seen in [283].

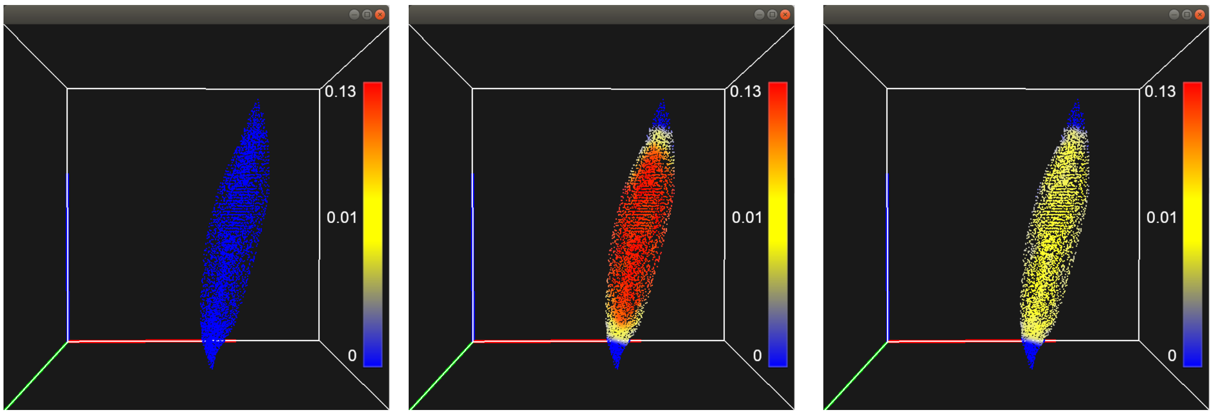


(A) (B) (C)

FIGURE 5.3: Integrated point-based tissue model for the triceps. (A) shows the initial state of the tissue, (B) shows the muscle after being innervated with a stimulus current, and (C) shows the muscle returning to its initial shape after the current was removed. The color of the particles represents its displacement with respect to its original position.

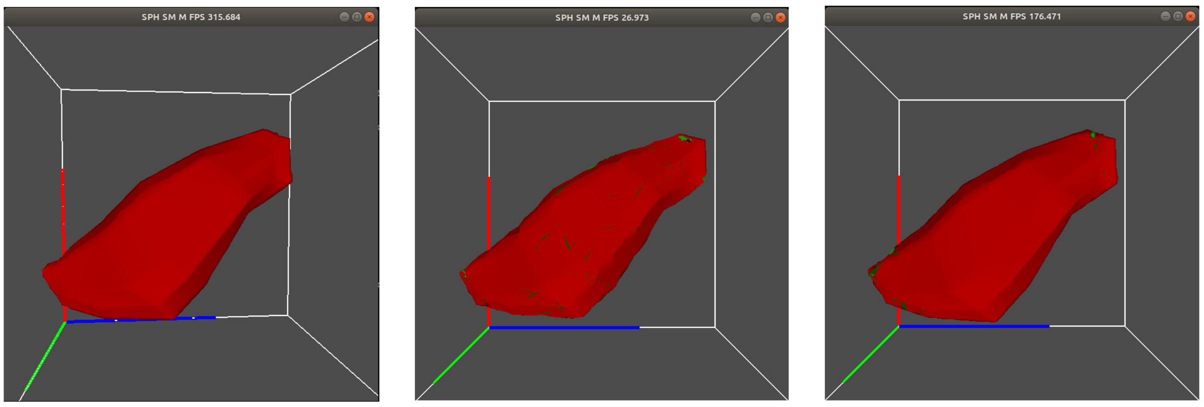
5.6 Meshed tissue simulation

To be able to see a clearer effect of the deformation of the muscle geometry, a meshed rendering of the tissue particles is presented. The triangle mesh that originally defined the muscle geometry was used for rendering. The simulation follows the same guidelines as the particle-based simulation. The resulting simulation can be seen in Figure 5.5 for the triceps, and in Figure 5.6 for the vastus lateralis. A video with the simulation of the meshed triceps can be seen in [284].



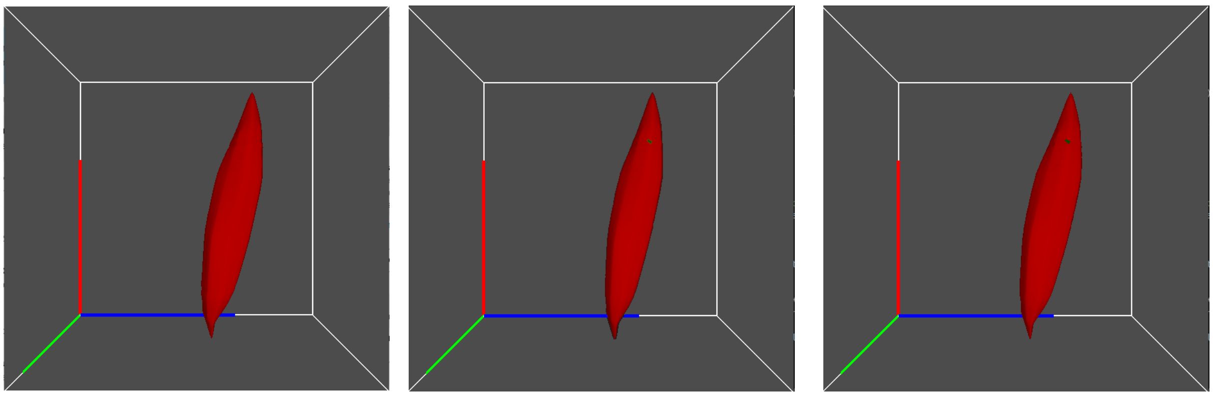
(A) (B) (C)

FIGURE 5.4: Integrated point-based tissue model for the vastus lateralis. (A) shows the initial state of the tissue, (B) shows the muscle after being innervated with a stimulus current, and (C) shows the muscle returning to its initial shape after the current was removed. The color of the particles represents its displacement with respect to its original position.



(A) (B) (C)

FIGURE 5.5: Integrated meshed triceps model. (A) shows the initial state of the triceps, (B) shows the muscle after being innervated, and (C) shows the muscle returning to its initial shape.



(A) (B) (C)

FIGURE 5.6: Integrated meshed vastus lateralis model. (A) shows the initial state of the muscle, (B) shows the muscle after being innervated, and (C) shows the muscle returning to its initial shape.

5.7 Simulation of tissue with the FEM

To evaluate the performance of the proposed method, a model solved with FEM was developed, and results were compared for the deformation and the pressure within the tissue. Particularly, the model was the one introduced by Blemker and Delp [141]. This model was selected because it is capable of representing complex muscle geometry and architecture from MR images, it considers muscle fiber orientation and arrangement, and the predicted muscle shape was compared to MR images of the same movement, obtaining less than 5mm of distance error for large muscles.

5.7.1 Software framework for FEM solutions

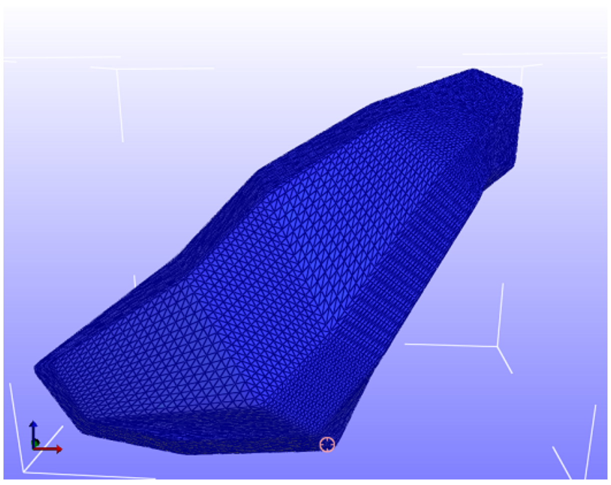
The FEM simulation was solved using the software framework FEBio [285]. The principal goal of the FEBio project is to provide an advanced finite element tool for the biomechanics and biophysics communities that allows researchers to model mechanics, transport, and electrokinetic phenomena for biological systems accurately and efficiently. In addition, since FEBio is geared towards the research community, the code is designed such that new features can be added easily, thus making it an ideal tool for testing diverse computational methods.

FEBio, however, is a nonlinear implicit FE solver and does not have mesh generation capabilities. Therefore the input files need to be generated by preprocessing software. The preferred preprocessor for FEBio is called PreView, which is the software that was used for the preprocessing of the geometry. PreView has been designed specifically to set up FE problems for FEBio. It allows the user to create or import meshes, specify the boundary conditions and material properties, and set the analysis options, all in a graphical environment.

To visualize and analyze the results from a FEBio analysis, the software Postview was used. It can import the FEBio extendible plot file format (XPLT), and it also offers several ways to add additional data to an already loaded model. PostView shows a graphical rendering of the model and, in the case the model has time-dependent data, can show an animation of the model. The rendering of the model can be augmented by adding additional plots, such as surface plots, isosurface plots, vector plots, plane cuts and several other.

5.7.2 Model pre-processing

As was previously stated, one of the drawbacks of the FEM is the preprocessing steps needed before any computation can be performed. In this case, a mesh with finite elements had to be created, and muscle material properties, including fiber distribution, were integrated. Boundary conditions were similar to the meshless simulation: the ends of the tissue were fixed in space. A tetrahedral mesh, with 20 nodes per element, was computed. Six resolutions were selected: 1 thousand, 2 thousand, 5 thousand, 10 thousand, 20 thousand, and 40 thousand finite elements. Figure 5.7 shows the mesh with around 40 thousand tetrahedral elements for the FEM simulation.


(A)

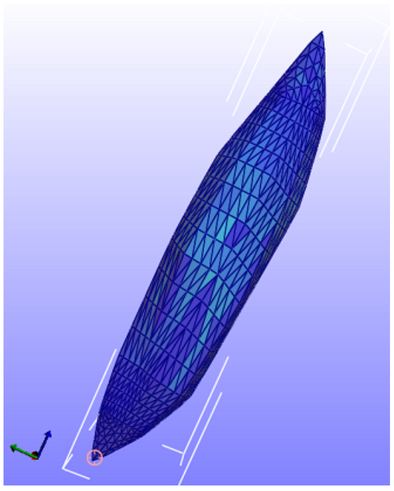

(B)

Figure 5.7: Tetrahedral elements for the triceps and the vastus lateralis. (**A**) shows the tetrahedral elements for the triceps, while (**B**) shows the tetrahedral elements for the vastus lateralis.

The material model used was the constitutive model developed by Blemker and Delp [141]. This model was selected because it is capable of representing complex muscle geometry and architecture from MR images, it considers muscle fiber orientation and arrangement, and the predicted muscle shape was compared to MR images of the same movement, obtaining less than 5mm of distance error for large muscles. The parameters of the model can be seen in Table 5.2.

Property	Value	Description
Density	1112 kg/cm^3	Density of the tissue
G_1	500 Pa	Fiber shear modulus
G_2	500 Pa	Cross shear modulus
K	1e5 Pa	Bulk modulus
P_1	0.05	Exponential stress coefficients
Lofl	10.7 cm	Optimal fiber length
σmax	3e5 Pa	Maximum isometric stress
α	8000	Activation level

Table 5.2: Properties used for the material model solved with FEM

5.7.3 FEM tissue simulation

FEM simulations consist on the geometry being activated for 250 time steps, and then removing the activation and running for 250 time steps. A step size of 0.001 was selected to avoid convergence errors. As for boundary conditions, we selected 10% of the nodes around the origin and insertion of the muscles, respectively, and fixed their displacement in X, Y, and Z. Fiber direction was created similarly to the meshless solution: instead of each particle having a direction, each node had a direction vector which represented the fiber direction. Figure 5.8 shows the deformation in millimeters of the triceps using the 40 thousand node resolution, while Figure 5.9 shows the deformation of the vastus lateralis for the 40 thousand node resolution.

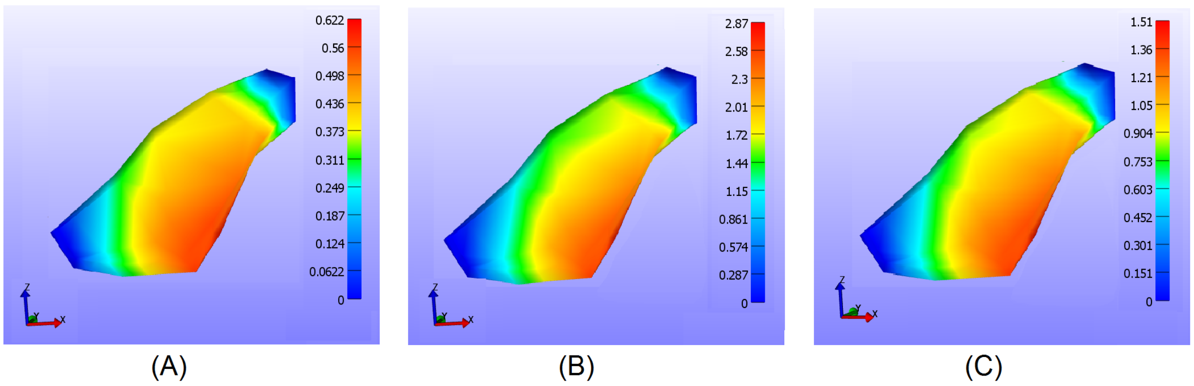
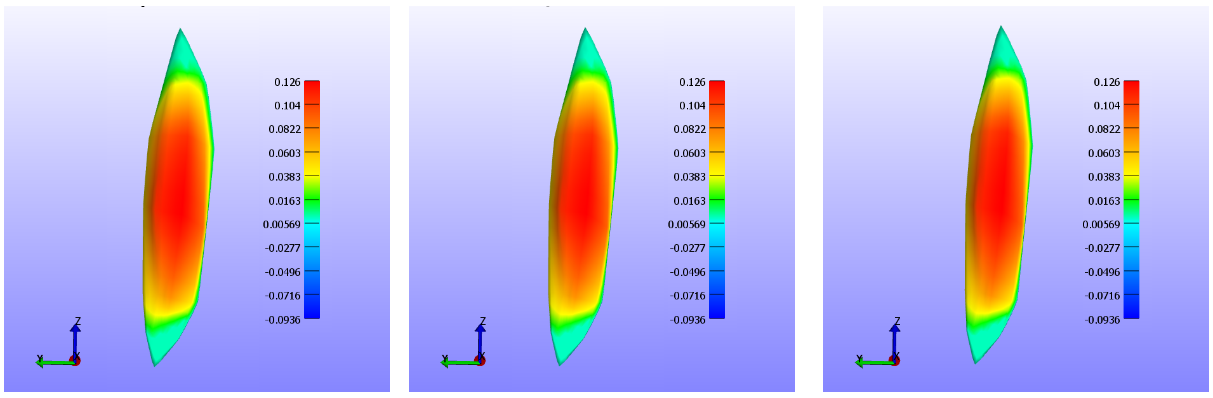


Figure 5.8: FEM based triceps model. (A) shows the initial state of the tissue, (B) shows the muscle after being innervated, and (C) shows the muscle after the current is removed. The color of the particles represents its displacement with respect to its original position.



(A) (B) (C)

Figure 5.9: FEM based tissue model for the vastus lateralis. (A) shows the initial state of the tissue, (B) shows the muscle after being innervated, and (C) shows the muscle after the current is removed. The color of the particles represents its displacement with respect to its original position.

5.8 Experimental results

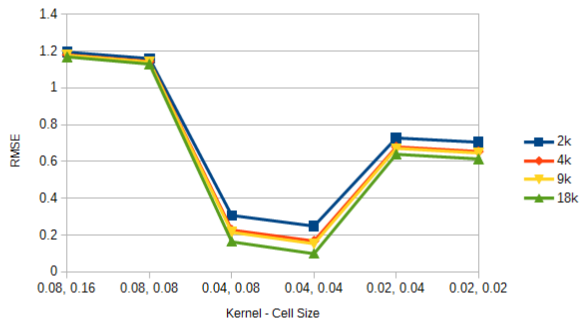
Results are presented as follows: first, a quantitative comparison against the FEM solution for the displacement and the pressure is presented; then the stability of the SPH method is discussed; finally, the computation times for the different simulations are reported. The results are from the GPGPU simulation, even though the CPU simulation also yielded similar results but with a larger processing time per step. To get accurate results, each simulation was executed 100 times, without rendering, and the displacement, pressure, and computation time were recorded for each.

5.8.1 Sensitivity analysis

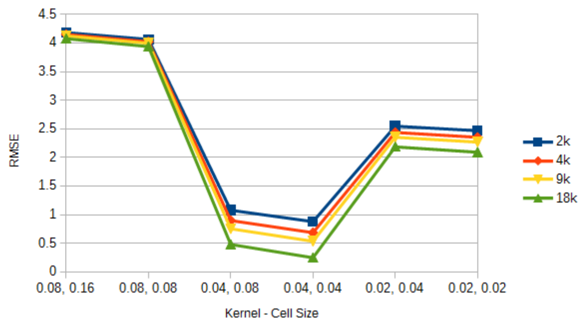
5.8.1.1 Error analysis

For the SPH method, a simulation for each of the proposed particle numbers, kernel, and cell sizes was developed. Similarly for the FEM, a simulation for each of the node resolutions was developed. Each of the simulations were paired against each other to calculate the RMSE. Figure 5.10 shows the average error for all the FEM resolutions against the different particle resolutions by kernel and cell size for the triceps, while Figure 5.11 shows the average error for the vastus lateralis. It can be seen that the kernel and cell size selection is critical when considering the accuracy of the model. In this case, when those value were set to 0.08 and 0.16, and to 0.08 and 0.08, the model presented the greatest error. This could be attributed to the fact that too many particles were present at a given neighborhood, and particles that should not have contributed to updating a property were updating it. This also explains the results for the 0.02 and 0.04, and the 0.02 and 0.04 configurations, where too few particles were contributing to updating the properties. Since the 18k particle distribution with 0.04 kernel and 0.04 cell size produced the least error with respect to the FEM, for the rest of the results these choice of parameters was used. Additionally, all the meshless simulations were compared to the 40k node resolution FEM solution.

To get a better understanding of the deformation and the pressure of the tissue, additional considerations were taken. The muscle belly was divided into three regions: one around the origin of the muscle, another around the insertion, and the last comprised the rest of the muscle. From the origin and insertion, respectively, 20% of the tissue was selected to form each of the respective regions. The region at center of the muscle belly comprised the remaining 60%. Additionally, for the displacement, different coordinate planes were considered to analyze the deformation of the tissue along those planes. Figure 5.12 shows the average error for each of the regions, and their respective planes for the deformation of the triceps, while Figure 5.13 shows the average error for the vastus lateralis. In this case, when more particles are used for the

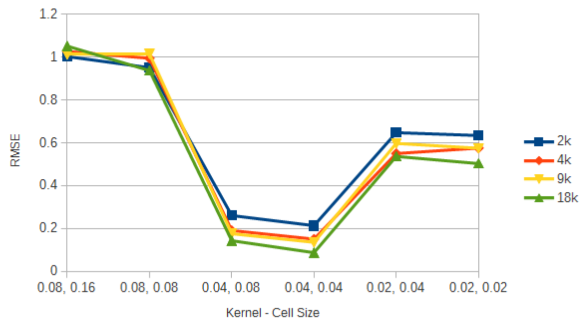


(A)

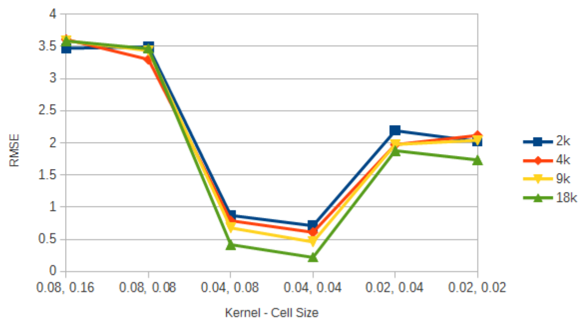


(B)

FIGURE 5.10: Average error for all the FEM resolutions for the triceps. (A) shows the average displacement error for all particle resolutions. (B) shows the average pressure error for all particle resolutions.



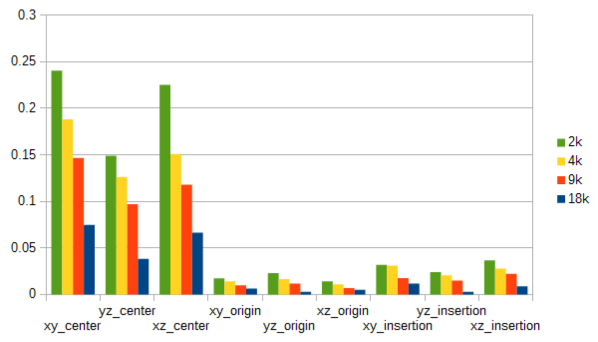
(A)



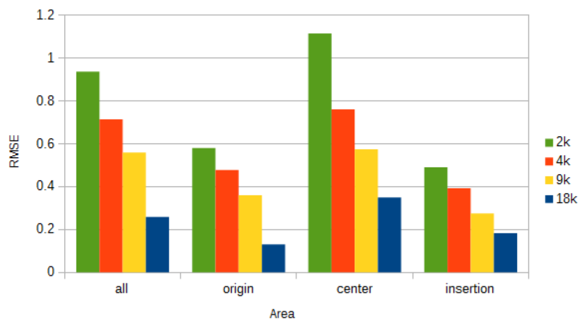
(B)

FIGURE 5.11: Average error for all the FEM resolutions for the vastus lateralis. (A) shows the average displacement error for all particle resolutions. (B) shows the average pressure error for all particle resolutions.

simulation, the error is reduced considerably at the expense of additional processing time. For example, when comparing the error produced by the xy coordinate plane on the center region for the 2k and 18k particle resolutions, the error difference is more than 80%.

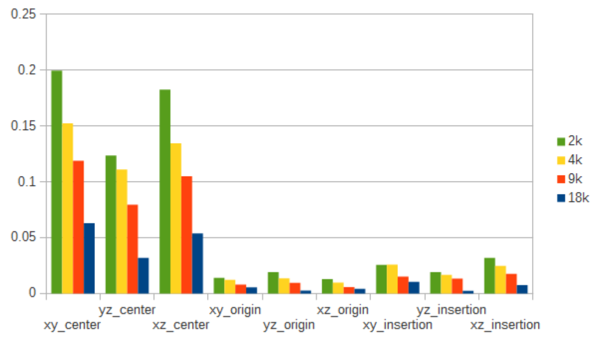


(A)

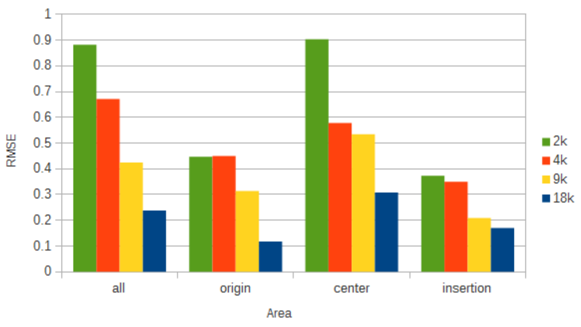


(B)

FIGURE 5.12: Average error for all the FEM resolutions for different regions for the triceps. (A) shows the average displacement error for all particle resolutions. (B) shows the average pressure error for all particle resolutions.



(A)



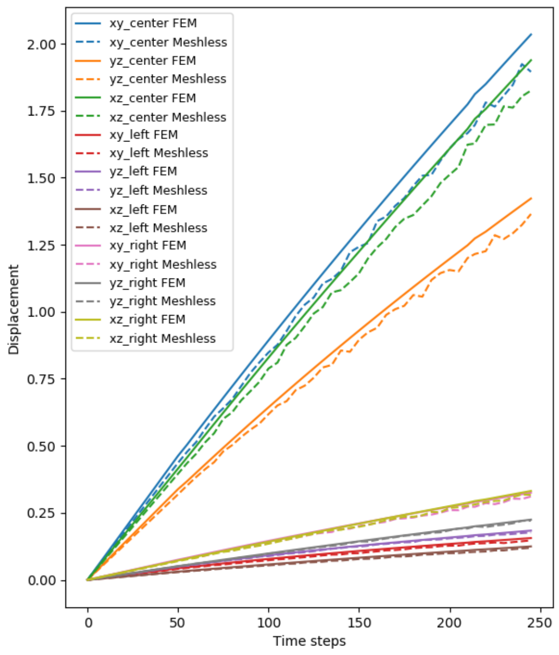
(B)

Figure 5.13: Average error for all the FEM resolutions for different regions for the vastus lateralis. (A) shows the average displacement error for all particle resolutions. (B) shows the average pressure error for all particle resolutions.

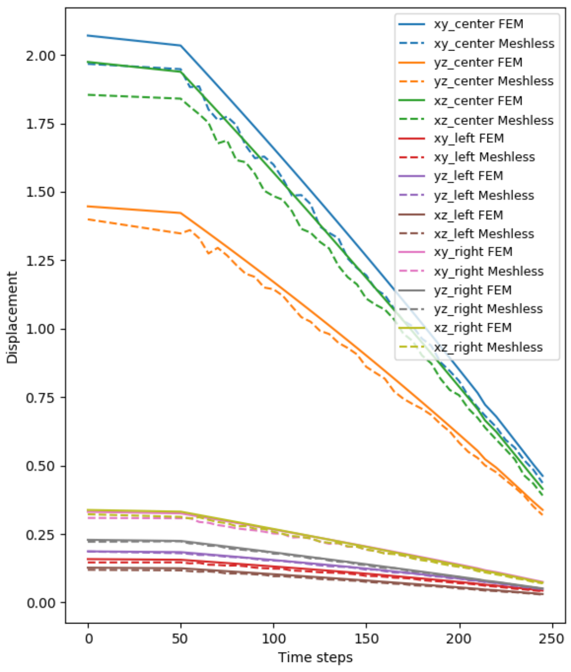
5.8.1.2 Displacement and pressure analysis

For the analysis of the displacement and the pressure, only one node configuration for FEM, and one particle configuration for the meshless method are presented. Several configurations were tested, but, for reporting, the chosen FEM configuration was with 40k nodes, while the meshless was 18k particles, with 0.04 cell size and kernel. These were chosen since more FEM nodes lead to better accuracy, and because with 18k nodes the error was the lower from all the tested configurations. It is worth noting that the results were similar for all the configurations, but with a larger error.

The mean displacement of the meshless and FEM simulations of the triceps can be seen in Figure 5.14, while Figure 5.15 shows the mean displacement for the vastus lateralis. Figure 5.14A shows the displacement for the contraction part of the simulation for the triceps (Figure 5.15A shows the same information for the vastus lateralis), while the displacement for the expansion part can be seen in Figure 5.14B (Figure 5.15B for the vastus lateralis). The area of the tissue that presented the most change was around the center, not only because it was the largest section, but also because the pressure exerted by the model made it so that the particles moved towards that direction. When both models were compared, a difference of around 10% was present throughout the simulations. Another point that became apparent was that the meshless simulation was not entirely stable when compared to the FEM simulation: the displacement of the model was not smooth and the data shows slight noise throughout. This was also apparent, if ever so slightly, in the rendered simulation: particles would oscillate while moving, creating visual artifacts. Contrary to the contraction of the tissue, where the displacement increased constantly, the expansion had a few moments, from time step 0 until around 50, where it experienced almost no change and then the displacement began to reduce constantly.

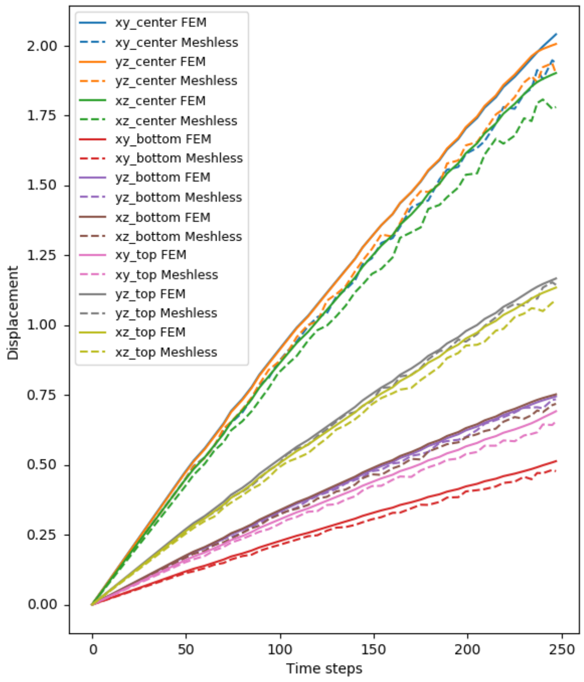


(A)

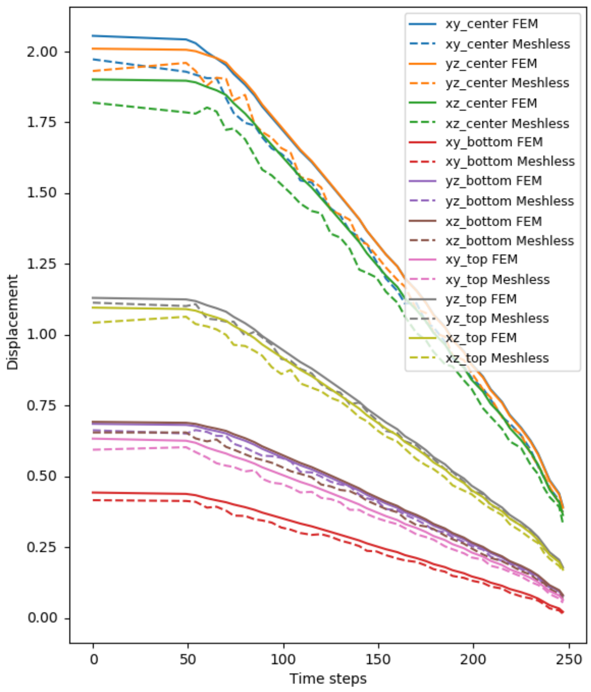


(B)

FIGURE 5.14: Mean displacement of the triceps. (**A**) shows the mean displacement for the contraction step, (**B**) shows the mean displacement for the expansion step.



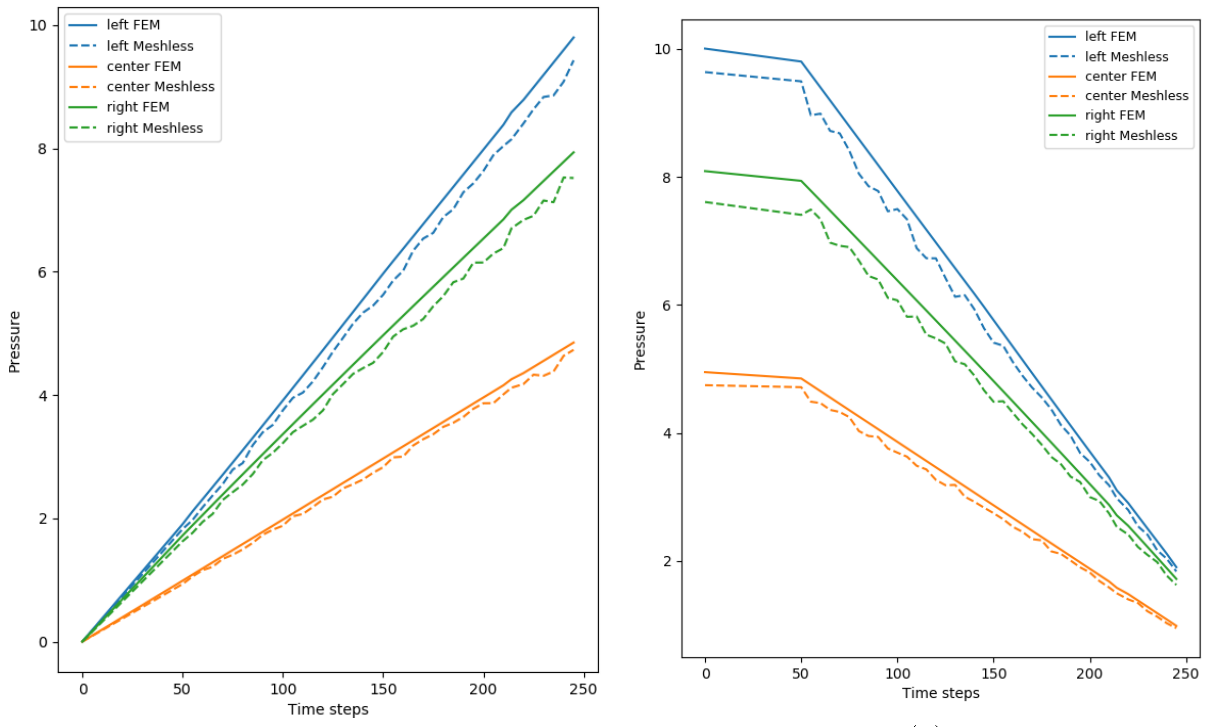
(A)



(B)

FIGURE 5.15: Mean displacement of the vastus lateralis. (**A**) shows the mean displacement for the contraction step, (**B**) shows the mean displacement for the expansion step.

The mean pressure of the particles for the triceps simulation can be seen in Figure 5.16, while Figure 5.17 shows the pressure for the vastus lateralis simulation. Figure 5.16a and Figure 5.17a show the pressure for the contraction part of the simulation, while the pressure for the expansion part can be seen in Figure 5.16b and Figure 5.17b. Even though the pressure for the center of the tissue increases, the pressure at the left and right sides is still larger; this allows the particles to move from areas of high pressure to areas of lower pressure. This behaviour is consistent with the pressure present in the FEM simulations. Similarly to the displacement, the pressure in the meshless simulation is around 10% lower than the FEM simulation, and is still not increasing smoothly: all throughout the simulation there is noise.



(A) (B)

FIGURE 5.16: Mean pressure of the triceps. (A) shows the mean pressure for the contraction step, (B) shows the mean pressure for the expansion step.

5.8.2 Stability and deformation of the model

The stability and deformations of the model depended mainly on the cell size, the core radius, and the number of particles. Deformation ranges were also similar. When the cell size and the core radius were set to 0.02 and 0.02, and to 0.02 and 0.04, the particles moved an average of 7% from their original positions. This behavior is due to an insufficient number of neighbor particles in each cell. Deformations became noticeable when the cell size and the core radius were set to 0.04 and 0.04, with a deformation of an average of 23%. This value ranged from around 20% to 26% depending on the number of particles; the simulation with 18475 particles yielded the 26% average deformation. This result is the closest to the 28% of optimal length during

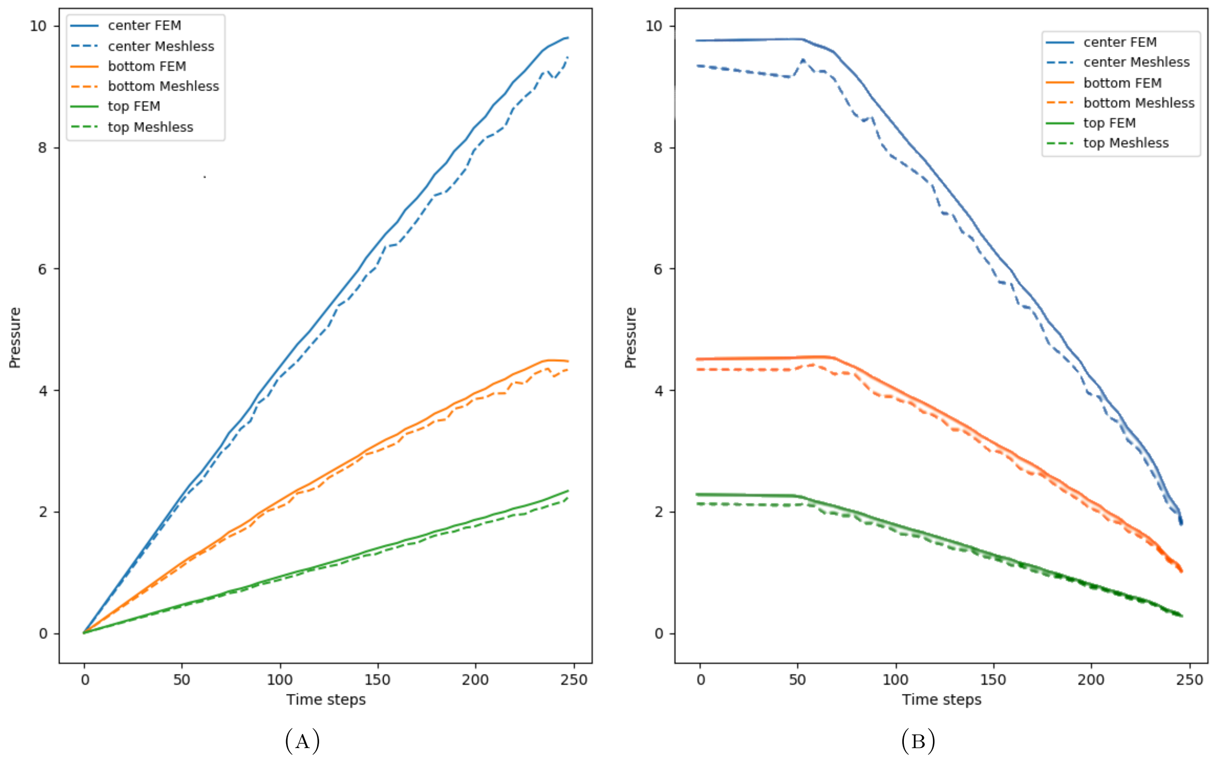


FIGURE 5.17: Mean pressure of the vastus lateralis. (A) shows the mean pressure for the contraction step, (B) shows the mean pressure for the expansion step.

contraction that was reported by Murray et.al. [286]. When the cell size and core radius were set to 0.08 and 0.04, there was an average deformation of 37%. When the cell size and the core radius were set to 0.08, the particles deformed more than 70% from their original configuration, also presenting visual artifacts. Finally, the simulation became unstable in less than 100 time steps when the cell size was increased to 0.16 and the core radius was set to 0.08. The geometry deformed more than 70%, with several artifact forming before losing the shape completely. The FEM simulation, in contrast, did not present any instabilities, and also got closer to the 28% average deformation when more nodes were considered.

Deformation of the particles had an additional effect perceivable when the stimulation was removed from the mesh-based simulation of the muscle. At some sections of the mesh, specially the origin and insertion of the muscle, the mesh would show holes or triangles that were not stable. This can be seen in the green sections of the triceps muscle in Figure 5.18 and of the vastus in Figure 5.19.

5.8.3 Computation time

Using the GPU, the computation time was considerably sped up when compared to the CPU version. The results for the average computation times for each of the algorithm steps, for the 18k particle set are presented in Figure 5.20. Both the triceps and the vastus simulation

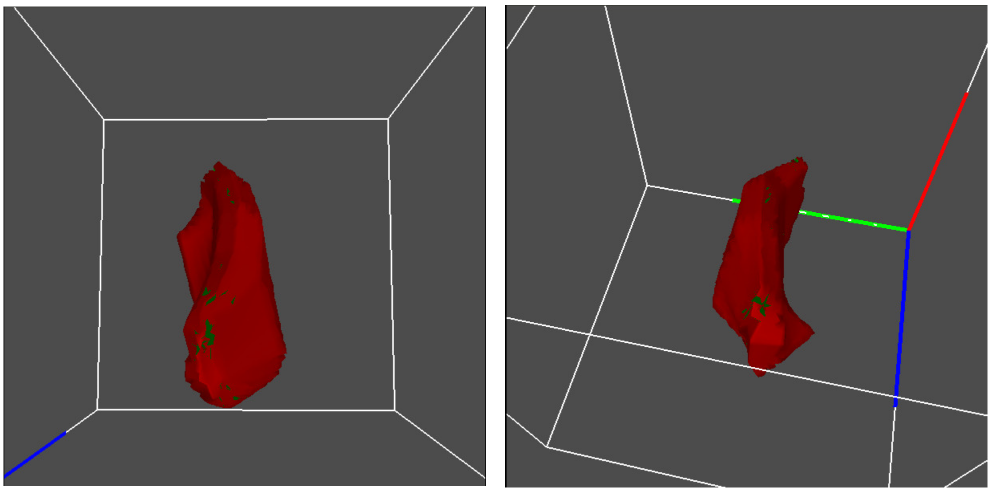

(A) (B)

Figure 5.18: Deformed triceps mesh at the points of origin and insertion. (A) shows the deformation at the origin, while (B) shows the deformation at the insertion.

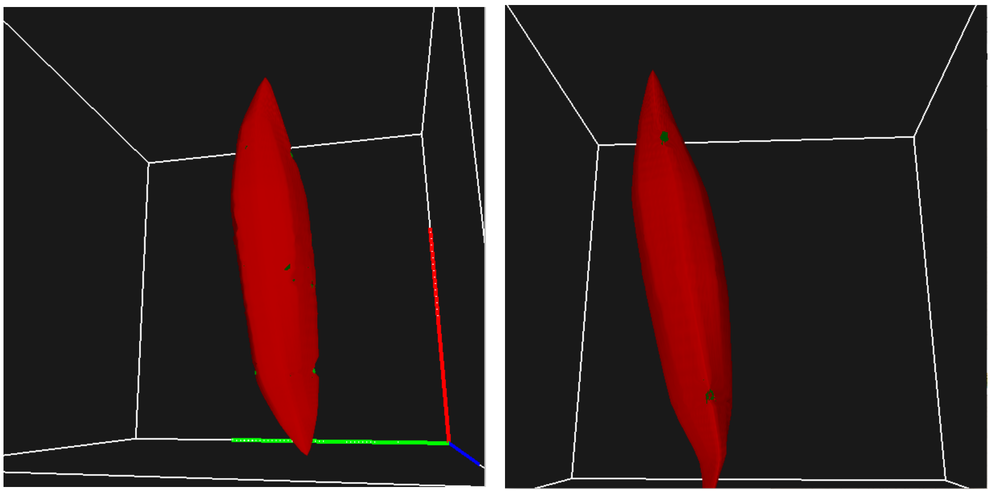
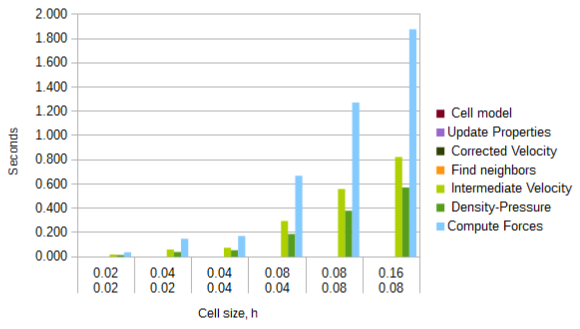

(A) (B)

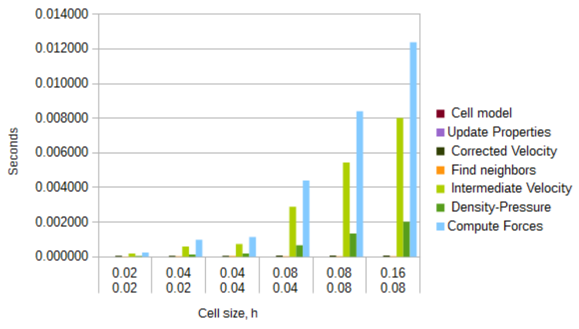
Figure 5.19: Deformed vastus lateralis mesh at several points of the mesh. (A) shows the deformation at several parts of the mesh, while (B) shows the deformation at the origin and insertion.

yielded similar execution times and speedups, with a difference of less than 1%. The presented results are the ones obtained for the triceps geometry. The average times and speedups were reported for each of the main methods of the algorithm in order to showcase the differences in their performance. For the average times in Figure 5.20b, the functions that calculated the corrected velocity, the cell model, and updated the properties performed the best since simple calculations on each element of the data sets were executed; they did not have constructs such as *if-then* blocks which lead to thread branching, and did not involve the more complex SPH method.

These simpler methods, however, did not gain much from being parallelized using GPGPU,



(A)



(B)

FIGURE 5.20: Average times for 18475 particles simulation. (A) shows the average times for cpu, while (B) shows the gpu times.

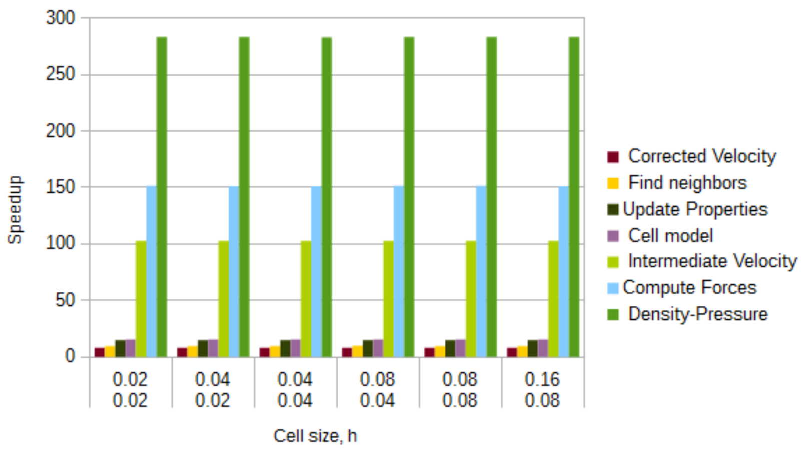


FIGURE 5.21: Average speedups for the muscle geometry with 18475 particles.

as can bee seen from the obtained speedups in Figure 5.21. The function that gained the least speedup was the calculation of the corrected velocity, with an average speedup of 8. The calculation of the cell model had an average speedup of 15.473, while the updating of the properties achieved an average of 14.887. Even though the search for neighbors for the GPGPU version was more elaborate when compared to the CPU version, it also was not sped up by much, having an average speedup of 9.715.

The functions that took the longest to compute, even while using GPGPU, where the ones that implemented the SPH method. These functions, specially the one that computes the forces, took, in some cases, more than two orders of magnitude more than the previously discussed simpler functions. However, since these functions are more elaborate, the contribution of using GPGPU was more noticeable. The intermediate velocity calculation was sped up by an average of 102.667, the computation of forces was speed up by an average of 151.551, while the calculation of the density and pressure by an average of 283.354. These results are dependent on the types of calculations and blocks that are involved in each function. The calculation of density and pressure had the most speedup since the executed operations were limited to additions and multiplications.

The average speedups for all cell and radius sizes of the kernels that implemented the SPH method can be seen in Figure 5.22. As more particles were involved in the calculations, the speedup for each kernel was larger, indicating that the SPH method benefited more with the use of GPGPU when a larger number of particles was involved in the calculations.

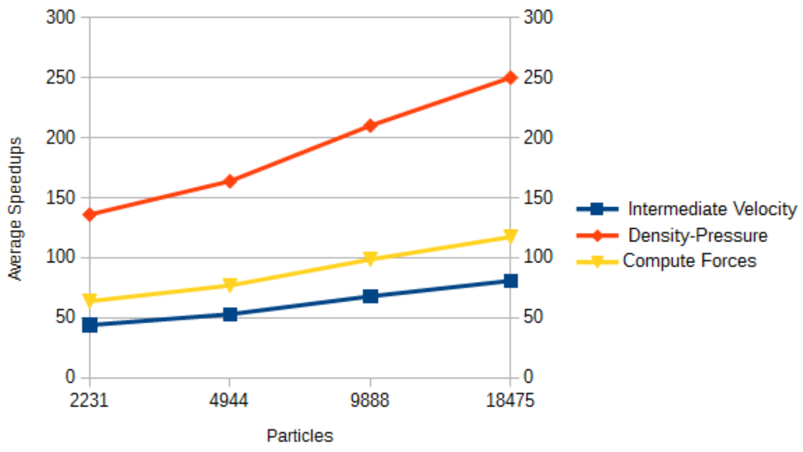


FIGURE 5.22: Average speedups obtained for each kernel and each particle set.

The decrease in execution time can also be seen in the increase in FPS of the simulations using GPGPU. Figure 5.23 shows the average FPS obtained for the muscle geometry. When compared to the CPU version of the muscle geometry simulation, the obtained FPS speedup was an average of 10.72 for the 2231 particle set, 12.303 for 4944 particles, 15.201 for 9888 particles, and 18.97 for 18475 particles. When more particles were simulated, a larger speedup was obtained. Real time simulations of one muscle could still be achieved when simulating 18475 particles for a cell size of 0.04, and a core radius of 0.04, because the average FPS obtained was 70.125 FPS.

Finally, it is worth noting that the computation times for the FEM were much higher than those for the SPH method. In particular for the 40 thousand node resolution, it took around 2 hours to compute 500 time steps. Figure 5.24 shows the average speedup when comparing the CPU and GPU versions of the SPH method to the FEM simulation. The FEM times could be improved if another platform that solved the equations in parallel was used, such as OpenCMISS [287].

5.8.4 Nvidia visual profiler results

The Nvidia visual profiler gave additional details on the performance of the GPU implementation. Specifically, it gave information regarding what could be limiting performance, and how

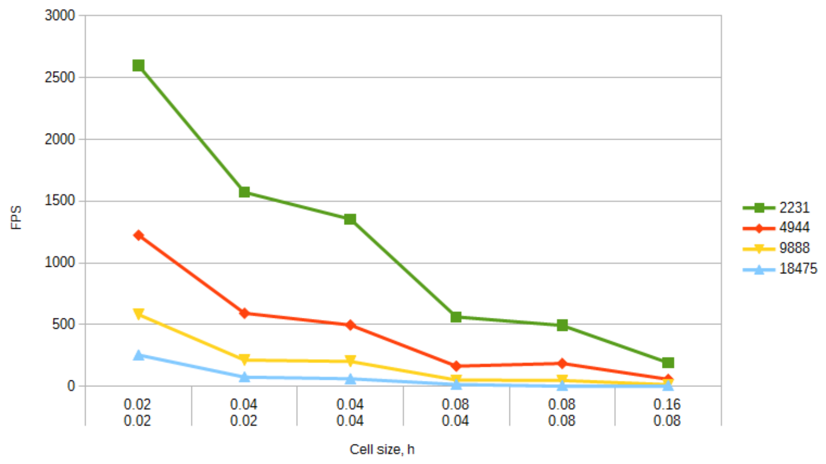
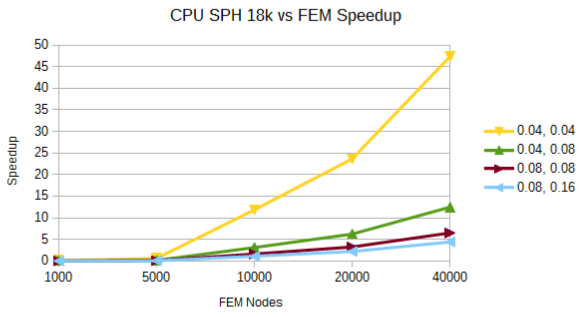
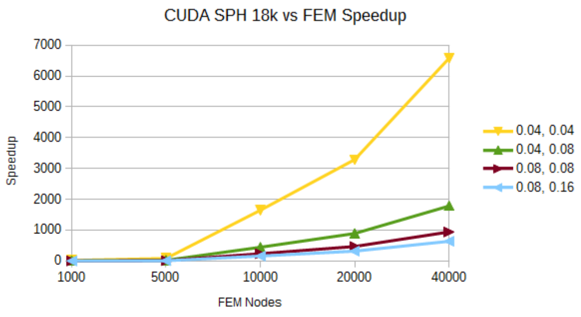


Figure 5.23: Average FPS for the different particle sets for the muscle geometry using CUDA.



(A)



(B)

Figure 5.24: Average speedups for SPH kernels when compared to FEM. (A) shows the average speedups obtained when comparing the SPH solution to the FEM solution using CPU (B) shows the speedup when comparing the SPH solution to the FEM solution using GPU.

it could be worked around. Tests for different block and thread configurations were performed for the functions using the SPH method (calculation of density and pressure, calculation of intermediate velocity, and calculation of force) because those were the ones that could benefit the most from performance gains. The tests were conducted on the 18475 particle set, where the cell size was set to 0.04, and the kernel size to 0.04.

5.8.4.1 Calculation of density and pressure

First, results for the kernel that calculates the density and pressure can be seen in Table 5.3.

The first step in analyzing an individual kernel is to determine if the performance of the kernel is bounded by computation, memory bandwidth, or instruction/memory latency. The results indicate that the performance of kernel was most likely limited by instruction and memory latency; specifically by the latency of arithmetic or memory operations. As can be seen in Figure 5.25, achieved compute throughput and/or memory bandwidth below 60% of peak typically

Grid Size	Block Size	Duration	Occupancy	Utilization
[289, 1, 1]	[64, 1, 1]	162.37 μs	25.2%	28%
[145, 1, 1]	[128, 1, 1]	163.107 μs	25.8%	28%
[73, 1, 1]	[256, 1, 1]	166.947 μs	25.5%	28%
[37, 1, 1]	[512, 1, 1]	164.514 μs	26.7%	28%

TABLE 5.3: Results for the different configurations of the compute density-pressure kernel.

indicates latency issues. The use of divisions within the kernels was the principal reason why there were instruction latency issues, and the memory latency was caused mainly because of the many necessary accesses to global memory in order to calculate each of the properties. Similar results were obtained for different configurations of blocks and threads.

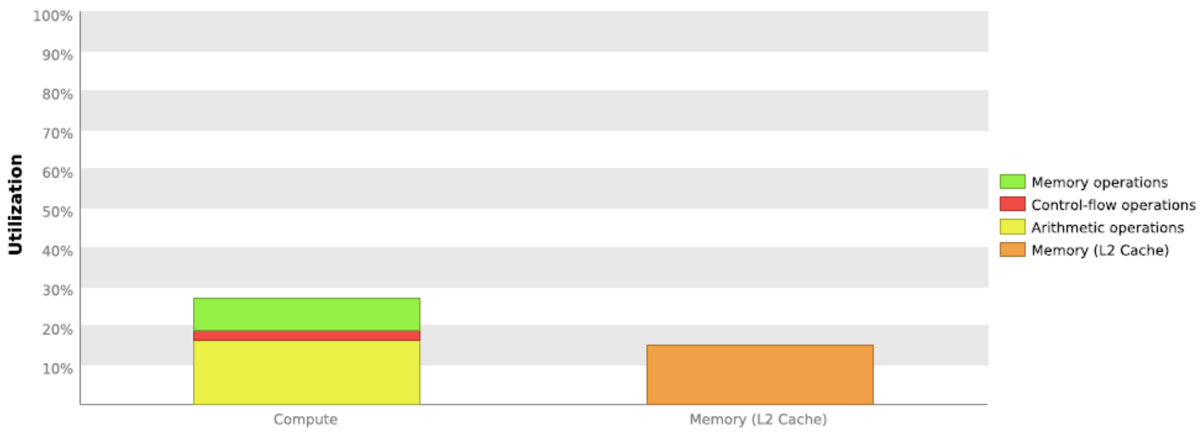
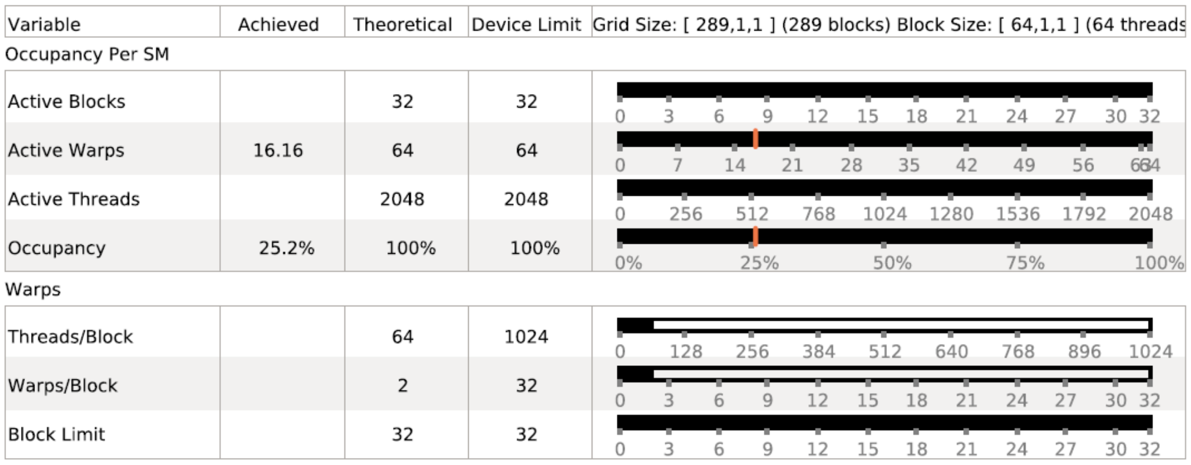


FIGURE 5.25: Utilization for the kernel that calculates density and pressure with 64 threads per block.

Instruction and memory latency limit the performance of a kernel when the GPU does not have enough work to keep busy. The performance of latency-limited kernels can often be improved by increasing occupancy. Occupancy is a measure of how many warps the kernel has active on the GPU, relative to the maximum number of warps supported by the GPU. Theoretical occupancy provides an upper bound while achieved occupancy indicates the kernel's actual occupancy. The results in Figure 5.29 indicate that the occupancy of 25.2% can be improved by executing a number of blocks that is multiple of 96. This allows more blocks and warps to be executed per SM. Other configurations also had this issue: the average occupancy across all configurations was 25.8%.

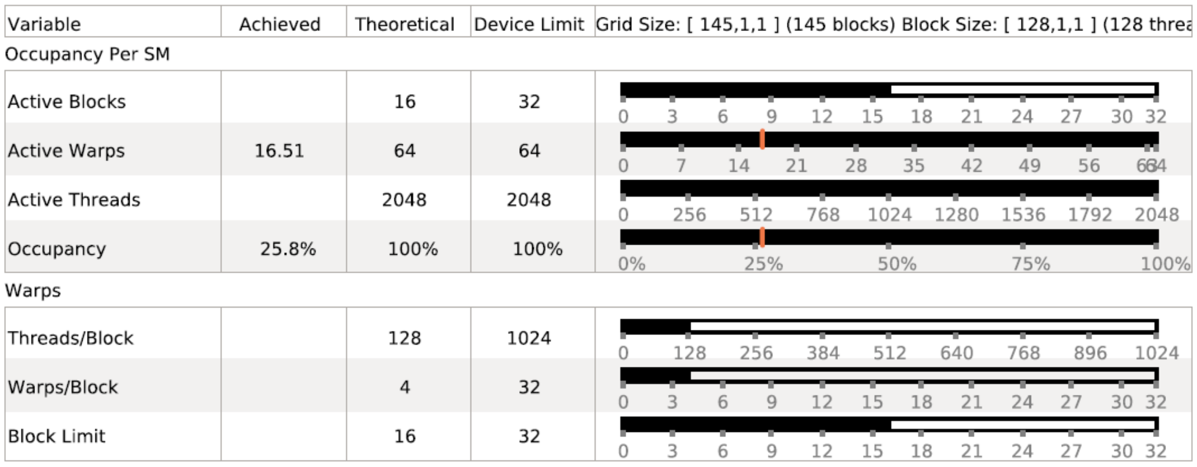
Increasing the number of threads did not do much to increase occupancy and block usage per SM, and reduce the execution time. Using 128 threads, and 145 blocks, the achieved occupancy was increased to 25.8%, but the blocks and warps were not used better, as can be seen in Figure 5.27. Additionally, the computation time was almost the same, increasing by 0.5%. It is also worth noting that more threads does not necessarily improve performance. For the case of 256 and 512 threads, the average duration of the kernels was increased by 2.8% and 1.3% respectively. The



Variable	Achieved	Theoretical	Device Limit	Grid Size: [289,1,1] (289 blocks) Block Size: [64,1,1] (64 threads
Occupancy Per SM				
Active Blocks		32	32	
Active Warps	16.16	64	64	
Active Threads		2048	2048	
Occupancy	25.2%	100%	100%	
Warps				
Threads/Block		64	1024	
Warps/Block		2	32	
Block Limit		32	32	

FIGURE 5.26: Occupancy for the kernel that calculates density and pressure with 64 threads per block.

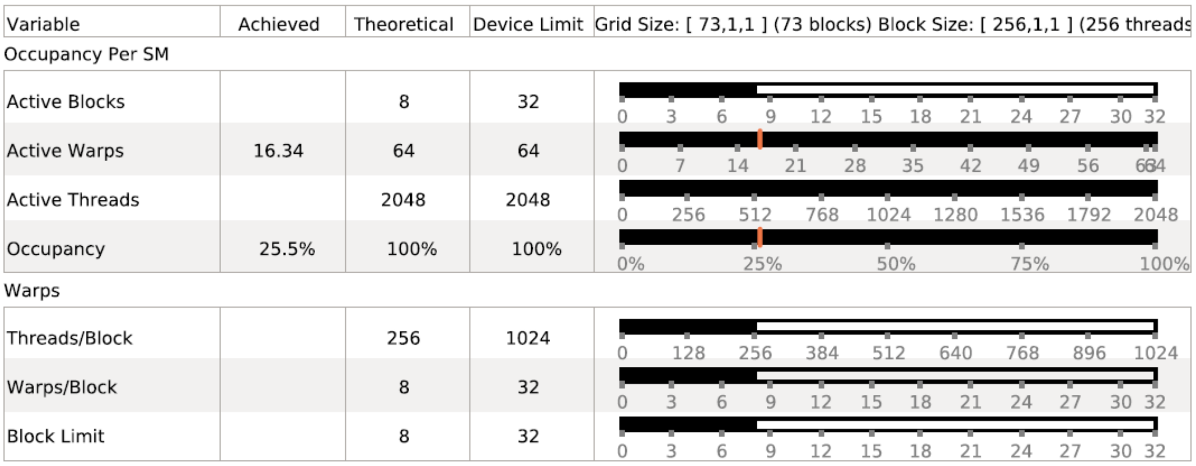
occupancy was 25.5% for 256 threads, and 26.7% for 512 threads. In these cases, the warps and registers were not used effectively, and there were blocks that were not active for the duration of the simulation. This can be seen in Figures 5.28 and 5.29.



Variable	Achieved	Theoretical	Device Limit	Grid Size: [145,1,1] (145 blocks) Block Size: [128,1,1] (128 threa
Occupancy Per SM				
Active Blocks		16	32	
Active Warps	16.51	64	64	
Active Threads		2048	2048	
Occupancy	25.8%	100%	100%	
Warps				
Threads/Block		128	1024	
Warps/Block		4	32	
Block Limit		16	32	

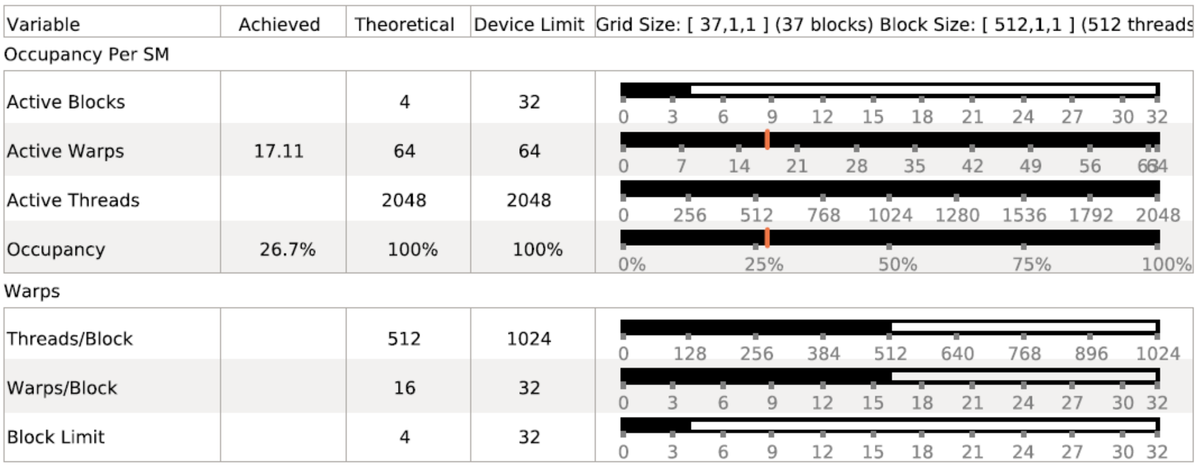
FIGURE 5.27: Occupancy for the kernel that calculates density and pressure with 128 threads per block.

Compute resource are used most efficiently when all threads in a warp have the same branching behavior. When this does not happen the branch is said to be divergent. Divergent branches lower warp execution efficiency which leads to inefficient use of the GPU's compute resources. For all tests, a divergence of about 60% was present. In the case of the SPH method, there are *if-then* and *for* blocks of in each of the kernels, and improving performance by decreasing divergence is not a trivial task, since it implies reformulating the algorithms themselves. Divergence can be seen in the *Inactive* result from Figure 5.30, which shows the thread executions that did not execute any instructions because the thread was inactive due to divergence.



Variable	Achieved	Theoretical	Device Limit	Grid Size: [73,1,1] (73 blocks) Block Size: [256,1,1] (256 threads
Occupancy Per SM				
Active Blocks		8	32	
Active Warps	16.34	64	64	
Active Threads		2048	2048	
Occupancy	25.5%	100%	100%	
Warps				
Threads/Block		256	1024	
Warps/Block		8	32	
Block Limit		8	32	

FIGURE 5.28: Occupancy for the kernel that calculates density and pressure with 256 threads per block.



Variable	Achieved	Theoretical	Device Limit	Grid Size: [37,1,1] (37 blocks) Block Size: [512,1,1] (512 threads
Occupancy Per SM				
Active Blocks		4	32	
Active Warps	17.11	64	64	
Active Threads		2048	2048	
Occupancy	26.7%	100%	100%	
Warps				
Threads/Block		512	1024	
Warps/Block		16	32	
Block Limit		4	32	

FIGURE 5.29: Occupancy for the kernel that calculates density and pressure with 512 threads per block.

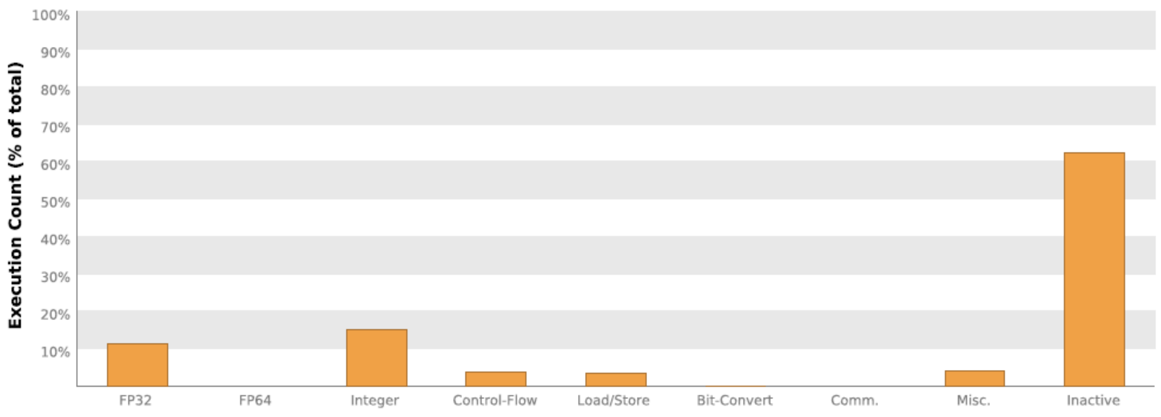


FIGURE 5.30: Instruction execution counts for density and pressure calculations.

Finally, this kernel was also restricted by the memory dependency: A load/store cannot be made because the required resources are not available or are fully utilized, or too many requests of a given type are outstanding. Data request stalls can potentially be reduced by optimizing memory alignment and access patterns. Figure 5.31 shows samples for each source and assembly line with various stall reasons, with memory management being the one causing more latency.

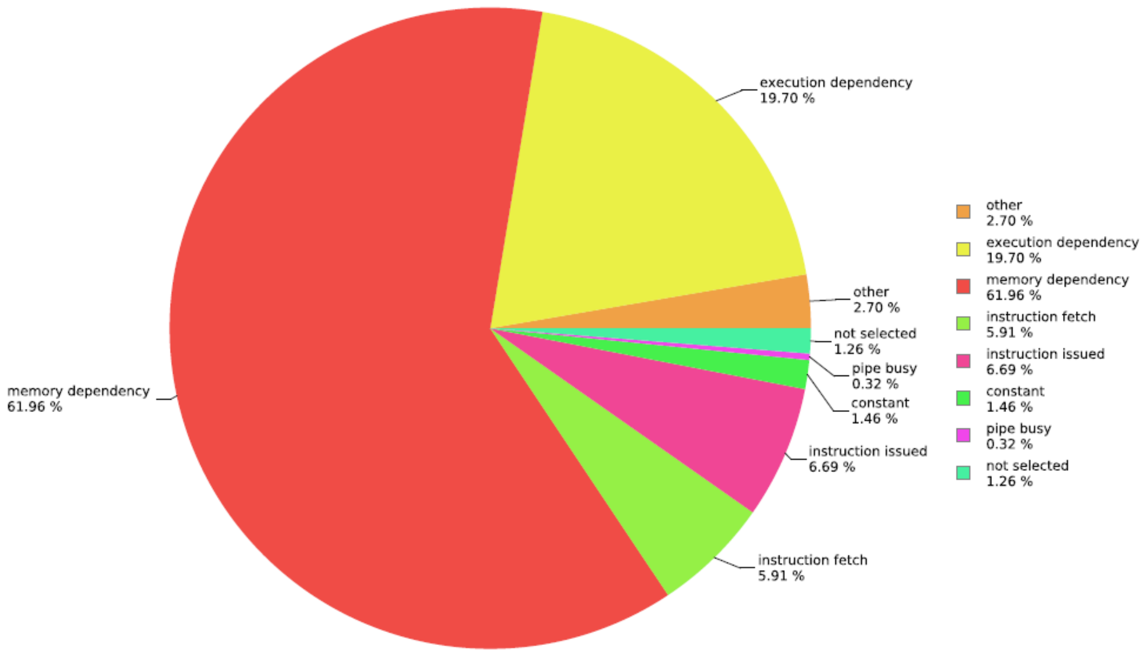


FIGURE 5.31: Memory dependency for density and pressure calculations.

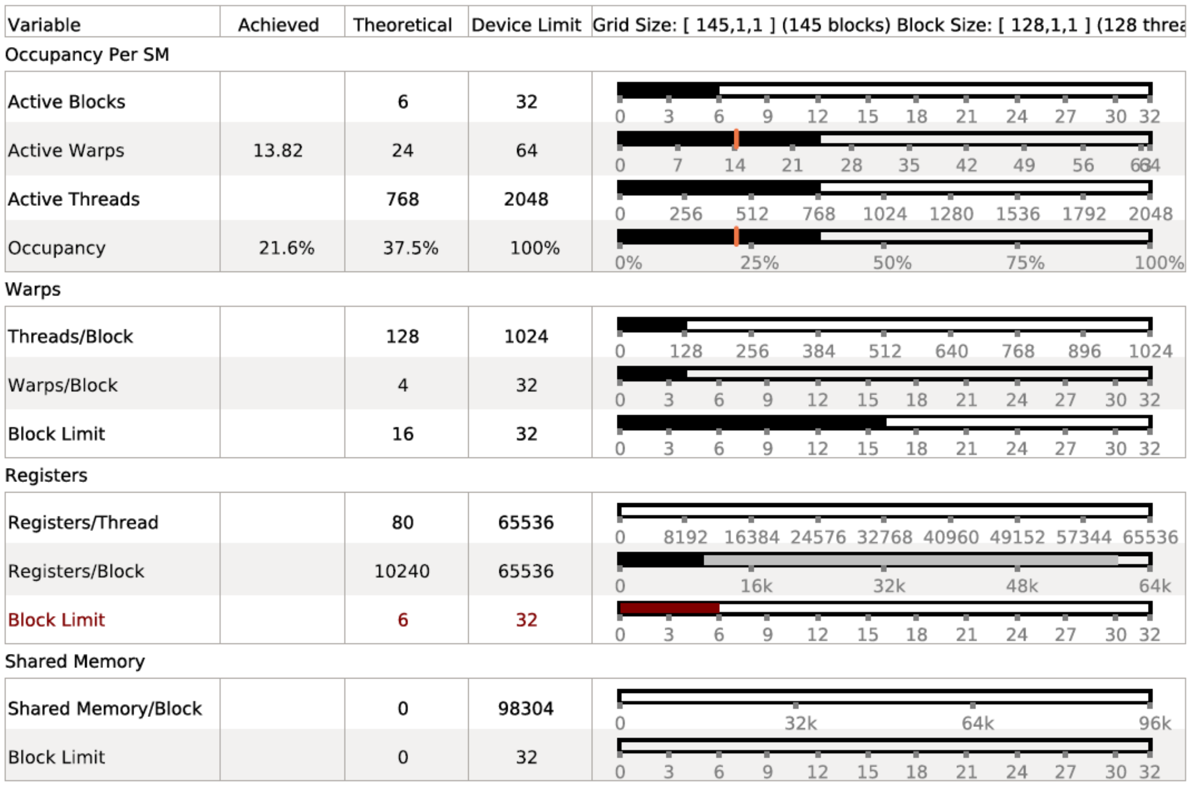
5.8.4.2 Calculation of intermediate velocities

Results for the kernel that calculates the intermediate velocity can be seen in Table 5.4.

Grid Size	Block Size	Duration	Occupancy	Utilization
[289, 1, 1]	[64, 1, 1]	817.42 μs	21.8%	32%
[145, 1, 1]	[128, 1, 1]	819.787 μs	21.6%	31%
[73, 1, 1]	[256, 1, 1]	808.906 μs	22%	31%
[37, 1, 1]	[512, 1, 1]	792.009 μs	16%	26%

TABLE 5.4: Results for the different configurations of the intermediate velocity calculation kernel.

The intermediate velocity kernel has the same issue with utilization as previous kernel. However, a potential bottleneck, which can be seen in Figure 5.32 is the register usage. Registers are the fastest memory space on a GPU. An automatic variable declared in a kernel without any other type qualifiers is generally stored in a register. Arrays declared in a kernel may also be stored in registers, but only if the indices used to reference the array are constant and can be determined at compile time. Register variables are private to each thread. A kernel typically uses registers to hold frequently accessed thread-private variables. Register variables share their lifetime with the kernel. Once a kernel completes execution, a register variable cannot be accessed again. Registers are scarce resources that are partitioned among active warps in an SM.



Variable	Achieved	Theoretical	Device Limit	Grid Size: [145,1,1] (145 blocks) Block Size: [128,1,1] (128 threa
Occupancy Per SM				
Active Blocks		6	32	
Active Warps	13.82	24	64	
Active Threads		768	2048	
Occupancy	21.6%	37.5%	100%	
Warps				
Threads/Block		128	1024	
Warps/Block		4	32	
Block Limit		16	32	
Registers				
Registers/Thread		80	65536	
Registers/Block		10240	65536	
Block Limit		6	32	
Shared Memory				
Shared Memory/Block		0	98304	
Block Limit		0	32	

FIGURE 5.32: Occupancy for the kernel that calculates the intermediate velocity with 128 threads per block.

For the case of 128 threads, the kernel uses 80 registers for each thread; 10240 registers for each block. This register usage is likely preventing the kernel from fully utilizing the GPU. The GPU used, GeForce GTX TITAN X, provides up to 65536 registers for each block. Because the kernel uses 10240 registers for each block, each SM is limited to simultaneously executing 6 blocks in 24 warps. To increase performance, the registers used by each thread have to be decreased. By using the *__launch_bound__* qualifier or the *maxregcount* compiler flag, the compiler is instructed to minimize register usage while keeping register spilling and instruction count to a minimum.

After testing with these options, register count was limited to 32 registers per thread. However, this was really not limiting kernel performance because occupancy was still at 21.7%, which is significantly lower that its theoretical occupancy of 100%. The kernel even took longer to compute: 1.074 ms instead of 819.787 μs, an increase of 31%. Figure 5.33 shows that there is an imbalance in how the kernel's blocks are executing on the SMs so that all SMs are not equally busy over the execution of the kernel. A possible solution would be to either increase the number of blocks executed by the kernel, or to make sure that all the blocks are doing roughly the same amount of work. Divergence is the most likely issue present in this kernel.

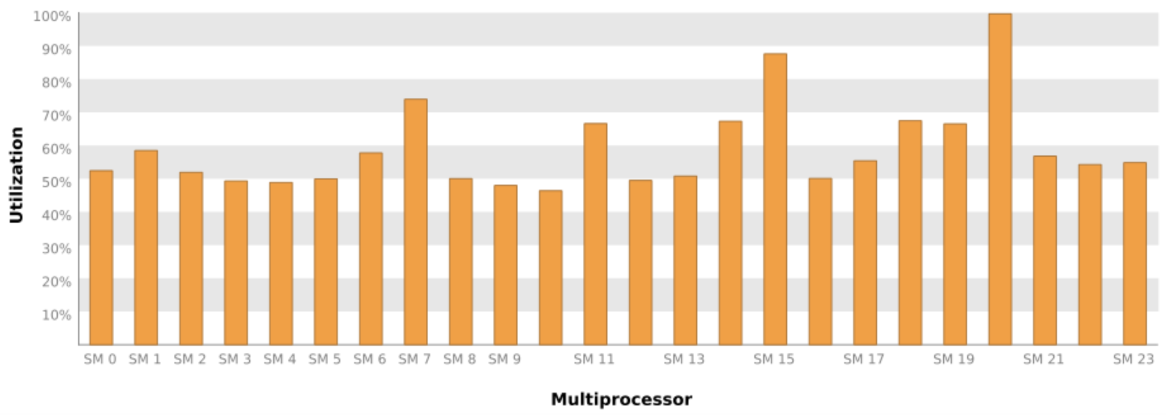


FIGURE 5.33: Utilization of SMs for the intermediate velocity kernel with 128 threads per block, with fixed register usage.

Figure 5.34 shows that this kernel was additionally restricted by execution dependency: An input required by the instruction is not yet available, and can potentially be reduced by increasing instruction-level parallelism. In this case, the kernel depends on the execution of several functions, in particular the calculation of the hash for the current cell, and to get the data on each of the neighbors.

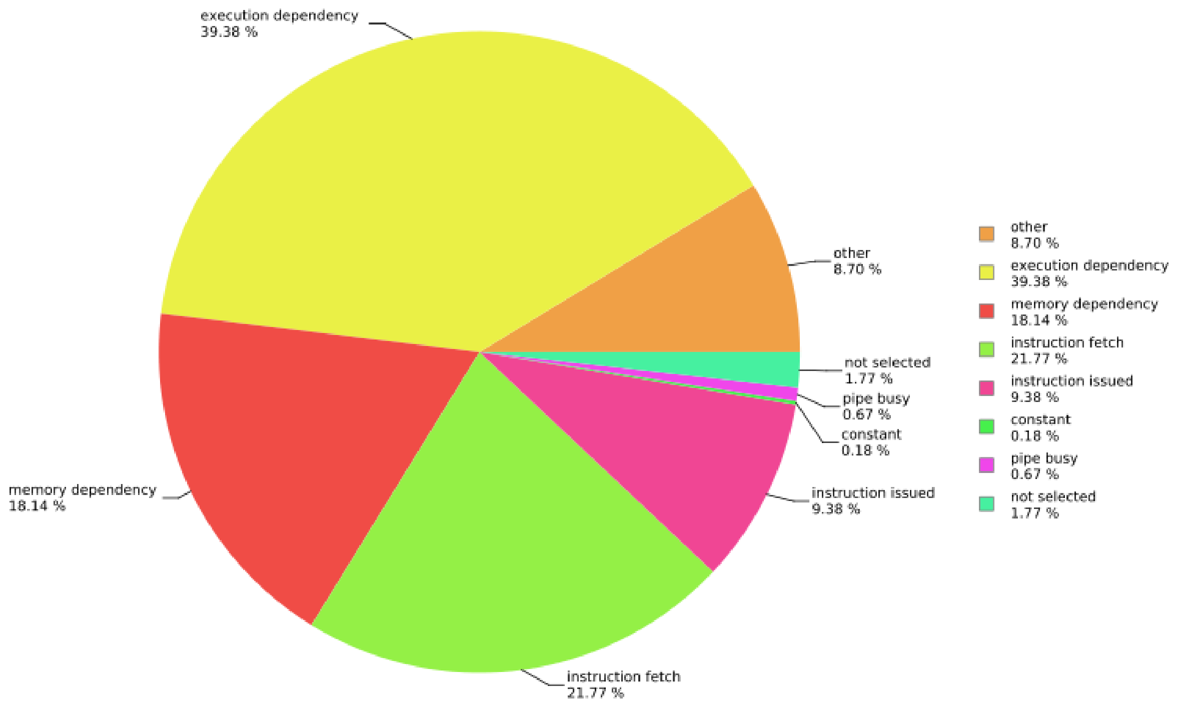


FIGURE 5.34: Execution dependency for the kernel that calculates the intermediate velocity.

This dependency also explains the results from Figure 5.35, where divergence is shown to be almost 80%: threads are stalled because they are waiting for the execution of those functions. A possible solution would be to include the calculation of those functions in one call, reducing both divergence and increasing the work made by a each thread.

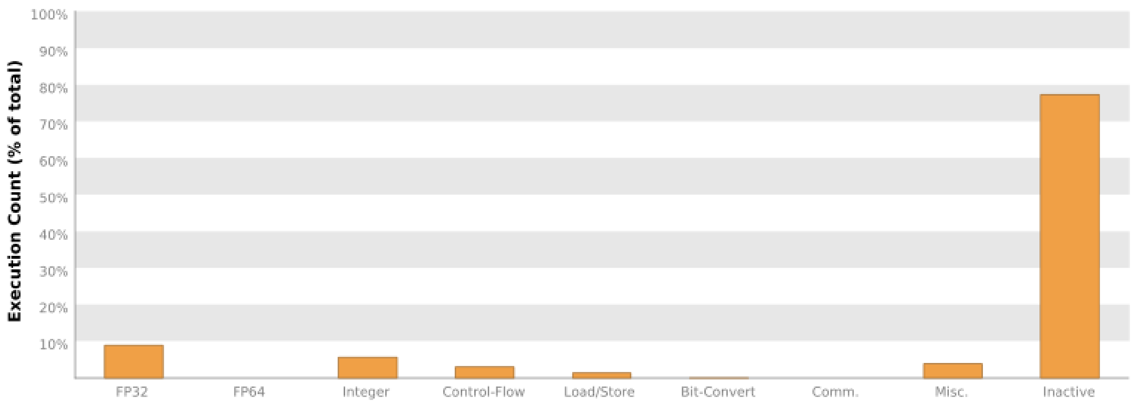


FIGURE 5.35: Divergence for the kernel that calculates the intermediate velocity with 128 threads per block.

5.8.4.3 Calculation of forces

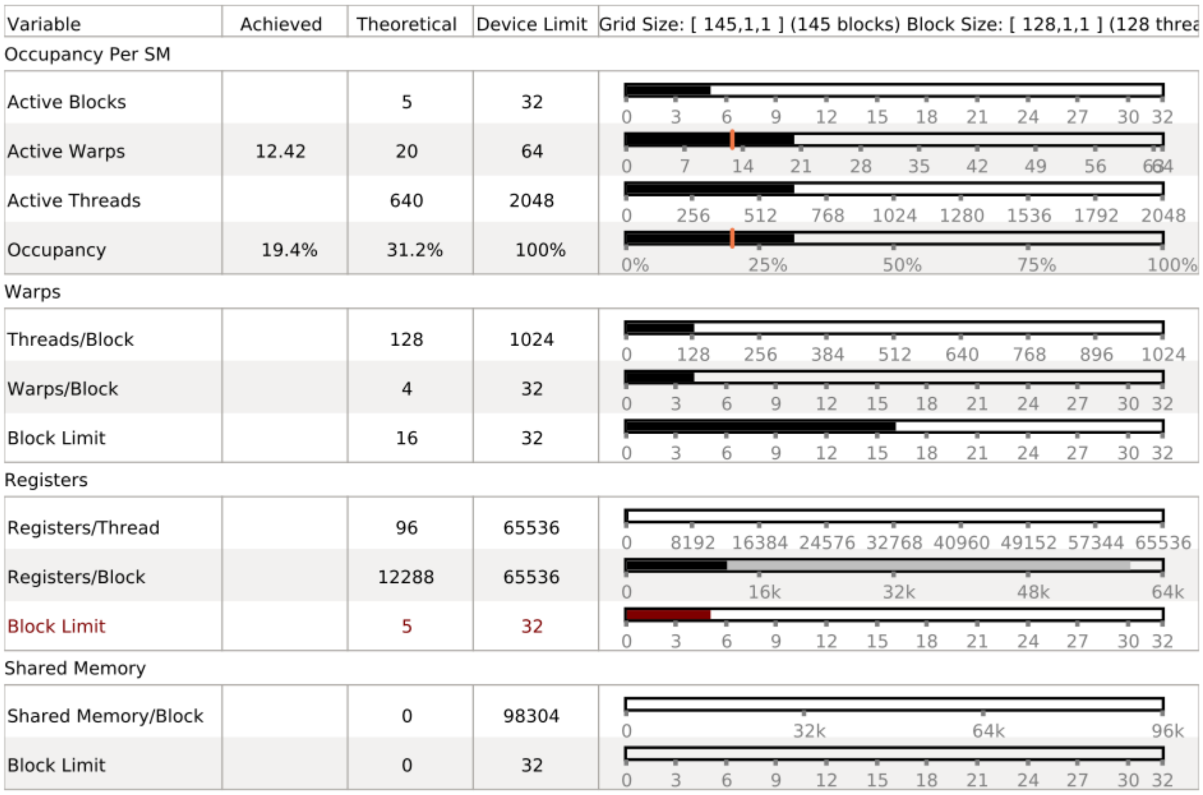
Finally, the results for the kernel that calculates the forces that are applied to the fluid is presented in Table 5.5.

Grid Size	**Block Size**	**Duration**	**Occupancy**	**Utilization**
[289, 1, 1]	[64, 1, 1]	1.413 ms	19.6%	23%
[145, 1, 1]	[128, 1, 1]	1.42 ms	19.4%	22%
[73, 1, 1]	[256, 1, 1]	1.41 ms	17.1%	20%
[37, 1, 1]	[512, 1, 1]	1.407 ms	16.2%	19%

TABLE 5.5: Results for the different configurations of the force calculation kernel.

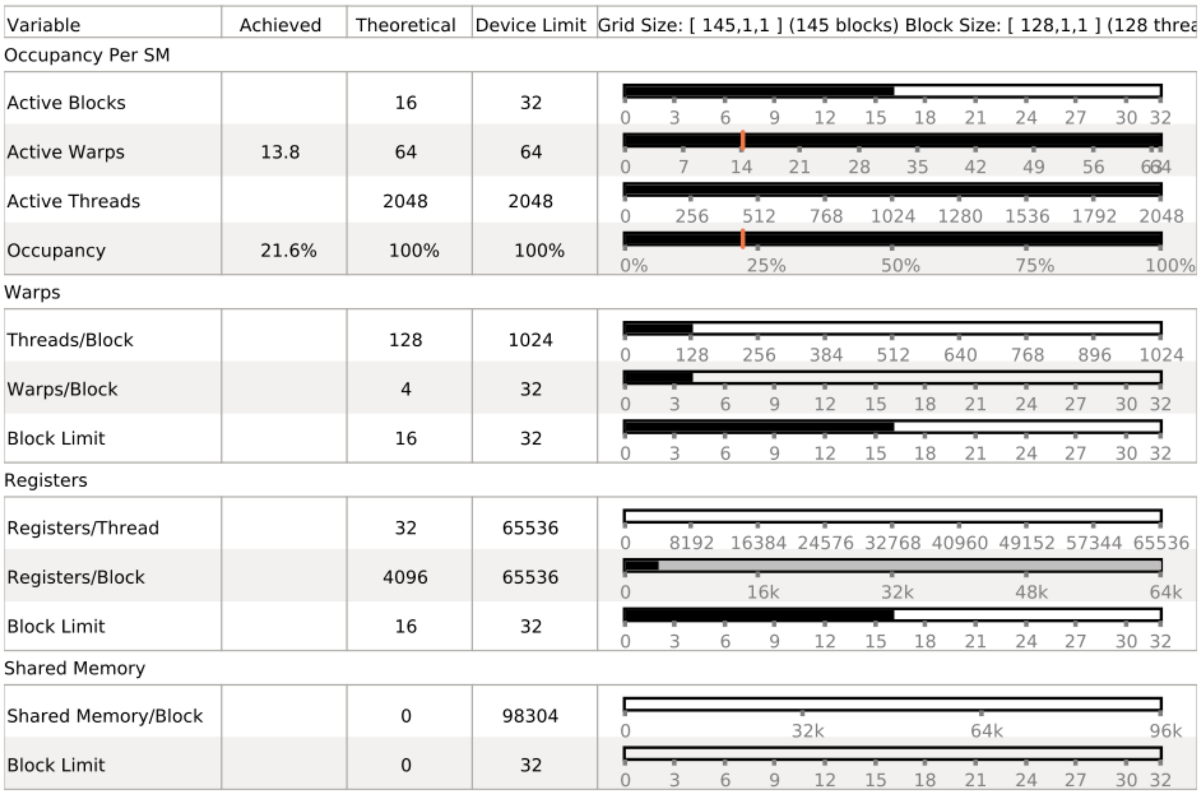
Similarly to the other kernels, this one still presents utilization, occupancy, and register usage issues. A possible fix for the occupancy and the register usage would be to use the *__launch_bound__* qualifier. However, this did not lead to increased performance; similarly to the previous kernel, performance worsened by around 18%, from 1.42 ms to 1.684 ms. Figure 5.36 shows the occupancy results before the qualifier was used, while Figure 5.37 shows the occupancy results after the fix was used.

Theoretical occupancy increased from 31.2% to 100%, but achieved occupancy was still significantly low. The main issue with this kernel was thread divergence, likely caused by execution dependency, and memory dependency, as can be seen in Figure 5.38. As previously mentioned, to improve performance a redefinition of the kernel, which is not a trivial task, would have to be considered. Different memory configurations and strategies to avoid divergence would need to be included.



Variable	Achieved	Theoretical	Device Limit	Grid Size: [145,1,1] (145 blocks) Block Size: [128,1,1] (128 threa
Occupancy Per SM				
Active Blocks		5	32	
Active Warps	12.42	20	64	
Active Threads		640	2048	
Occupancy	19.4%	31.2%	100%	
Warps				
Threads/Block		128	1024	
Warps/Block		4	32	
Block Limit		16	32	
Registers				
Registers/Thread		96	65536	
Registers/Block		12288	65536	
Block Limit		5	32	
Shared Memory				
Shared Memory/Block		0	98304	
Block Limit		0	32	

FIGURE 5.36: Occupancy for the kernel that calculates the forces of the fluid with 128 threads.



Variable	Achieved	Theoretical	Device Limit	Grid Size: [145,1,1] (145 blocks) Block Size: [128,1,1] (128 threa
Occupancy Per SM				
Active Blocks		16	32	
Active Warps	13.8	64	64	
Active Threads		2048	2048	
Occupancy	21.6%	100%	100%	
Warps				
Threads/Block		128	1024	
Warps/Block		4	32	
Block Limit		16	32	
Registers				
Registers/Thread		32	65536	
Registers/Block		4096	65536	
Block Limit		16	32	
Shared Memory				
Shared Memory/Block		0	98304	
Block Limit		0	32	

FIGURE 5.37: Occupancy for the kernel that calculates the forces of the fluid with 128 threads, with fixed register usage.

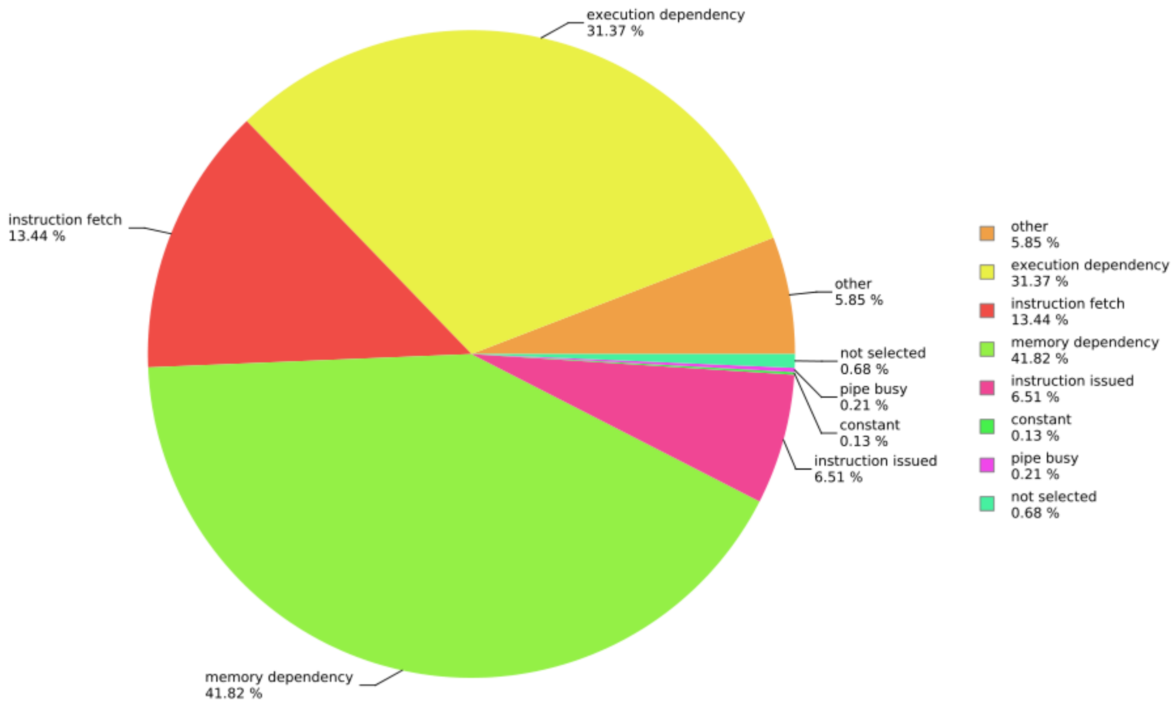


FIGURE 5.38: Execution dependency for the kernel that calculates the forces of the fluid with 128 threads.

Chapter 6

Conclusions and Future Work

In this thesis, a new particle-based meshless method to simulate biological tissue was introduced; specifically through the simulation of a skeletal muscle belly. The muscle tissue was simulated using a highly viscous fluid which preserved its shape and volume, and it was controlled and deformed with a biophysical cell model. In order for the simulation of the muscle to be viable in interactive applications, the method was accelerated with the use of GPGPU, allowing real time simulations.

The proposed method simulates a highly viscous fluid using SPH. A velocity correction scheme using Shape Matching was used to allow the fluid to preserve its shape and volume. And, finally, using the monodomain model, and the FitzHugh-Nagumo cell model, the fluid was activated with an electric stimulus, and its shape was contracted or relaxed accordingly. For this method, a meshless, particle-based approach was selected in order to avoid having to use a more computationally demanding method such as FEM, while at the same time taking advantage of the flexibility of the SPH method to solve diverse phenomena such as fluid and solid simulations, or electromagnetic equations. After comparing the simulation's results to a FEM simulation of an established model, it was shown that the proposal, although not perfect, was accurate with a difference of around 10%.

Even though SPH and Shape Matching had already been used to simulate viscoelastic fluids, to our knowledge, this was the first time a viscoelastic fluid was used in conjunction with a biophysical model, specifically, the monodomain model, to simulate biological tissue. Additionally, the use of SPH to provide an alternative mean to solve the monodomain model was implemented and used in the presented simulations. To our knowledge, this was also the first time the monodomain model was solved with SPH.

A comparison of some of the advantages and disadvantages of previous solutions versus the proposed model can be seen in Table 6.1.

References	**Mesh or Particle**	**Parallel implementation**	**Advantages**	**Disadvantages**
Palyanov et al. [64]	Particles: SPH	Yes: OpenCL	Uses PCISPH and OpenCL to model biological tissue including liquids, elastic, and contractile matter.	Does not add material properties, uses mass-spring for elastic matter, and does not include electrophysiology.
Röhrle et al. [34, 84], Heidlauf et al. [177]	Mesh: FEM	Yes: CPU with OpenCMISS	Accurate simulations that consider electrophysiology and chemoelectromechanical behaviour	Limited by FEM: computationally expensive with no real time simulations. Produces speedups of 4. Limited number of FEM nodes. Offline simulation.
Chen et al. [214], Basava et al. [213], Valizadeh et al. [215]	Particles: RKPM	No	Uses pixel data from medical images to model skeletal muscles, and diffusion tensor imaging to get the fiber direction.	Mainly used for solids, not viscoelastic tissues. They do not model electrophysiology. They define a RKPM formulation, using elasticity tensors, instead of using the actual form of the muscle, or its physiological properties to deform it. No mention of parallelism or computational time.
Blemker et al. [141]	Mesh: FEM	No	Creates 3D FE models of muscles from MR images. Proposed a method that prescribes the geometry of the fibers. Low error when compared to MR images of muscles.	Does not consider electrophysiology. No mention of a parallel solution or computational time. Offline simulation.
Ivanović et al. [288]	Mesh: FEM	Yes: MPI and CUDA	Adaptive load algorithm that allows for distributed calculations, and high utilization of both CPU and GPU. FEM calculations take place in CPU, while numerical solutions of the Huxley model are calculated on both CPUs and GPUs.	Although a 200 speedup was obtained, simulations still took 28 min to compute offline (instead of 96 hrs).

Table 6.1: Comparison of advantages and disadvantages of different models versus the proposed one.

According to the results obtained, the following contributions of this research work can be highlighted:

- **Biological tissue simulation.** Most of the previous work related to the simulation of biological tissue used methods such as FEM. However, the definition and pre-processing of the mesh, in addition to the high computational cost, even with the use of GPGPU, made the method less than ideal for real time simulations. Here, an alternative mean to simulate biological tissue was presented. In this case, two different skeletal muscles were simulated by using skeletal muscle geometry and a specific activation model. By changing the geometry and the constitutive model, different biological tissues could be simulated. Since SPH was used to simulate tissue, an integration with other particle-based models is also possible; for example, the inclusion of blood in the tissue.
- **Solution of the monodomain model with SPH.** The monodomain model was usually solved with FEM or the FDM. Here, the model was solved with SPH, which is also paralellizable with GPUs, and the results were similar to the ones of a simulation with the FEM. The use of GPUs was also explored to speed the simulations up. These results show that biophysical models can be solved with similar accuracy using the SPH method, and that the solved properties can be easily included in a more complex model. Additionally, when compared to FEM, implementing a meshless solution is considerably easier: from the creation of the particles that are needed, to the inclusion and solution of additional equations and models.
- **Tissue deformation.** The focus of this work was on the deformation of the muscle tissue by using a biophysical model. In order to apply the transmembrane potential to the tissue, and deform it, the potential was considered as a force of pressure that acted on the fluid. Since the fluid, and in this case, the tissue, flows from regions of high pressure to regions of low pressure, the added pressure made it so that the tissue contracted in a given direction when a stimulus current was applied, or it relaxed to its original shape when the current was removed. To our knowledge, this was the first time that such a model was used. The proposed method was able to achieve a contraction of around 26%, which is similar to the achieved contraction of a muscle. Additionally, the RMSE for the simulation was low when compared to a FEM simulation that has proven to be reliable to simulate skeletal muscles.
- **Achieved real time simulations.** In order for the model to be viable in interactive simulations, it had to be able to be simulated in real time, at least 30 FPS. The CPU version, with around 9888 particles, was able to run in real time; at an average of 27.5 FPS for the mesh-based simulation. If 18475 particles were simulated in CPU, an average of 3.7 FPS were obtained for the mesh-based simulation. In order to get a more detailed

tissue, and to be able to simulate more than one, the use of GPGPU was proposed. The GPU version, with 18475 particles, ran at 70.125 FPS for the mesh-based simulation. Even though the achieved speedups for FPS were not as high, speedups of more than 250 were achieved for specific parts of the method. Additionally, different techniques, such as reducing the neighbor search space, were used to reduce the computational complexity of the model.

In spite of the advantages of the method, there were several concerns that had to be worked around. For the GPU version, the use of expensive arithmetic operations, such as divisions, hindered the performance of the simulation. Additionally, the algorithm had branching paths that were mostly idle, which also caused the performance to be lower. Additionally, even though high speedups were obtained, utilization and occupancy were specially low. This shows that the GPGPU algorithm needs a rework to avoid divergence. Another issue is that the particle distributions may have been too low for the Titan X GPU. The issue has to do more with correctly balancing work throughout the SMs than with processing power or insufficient memory. The visual profiler tool gave some insights into how to optimize the GPU version of the solution; however, that could require considerable reformulation of the algorithm and its implementation.

Next, the model created visual artifacts in the simulation: when the cell size or the core radius were larger than 0.08, the particles would clump up at different points of the tissue, instead of returning to their original position. The clumps of particles was also the cause for the reconstructed mesh to have holes. By analyzing the particles' properties in those areas, it could be seen that they were activated more than in other areas. Improving the fiber orientation may help reduce this issue, since particle were not correctly moving in the expected direction. Acquiring fiber orientations from MRI data could have a great impact on the model's behaviour.

Finally, the accuracy and stability of the model could be considerably improved. To improve the accuracy of the simulation, the bidomain model could be used instead of the monodomain model. Additionally, the SPH method in its standard representation has also some shortcomings:

- Accuracy of flow variable approximation as an optimized point between the interpolation accuracy and numerical diffusion
- Modeling of large ratios of density/viscosity discontinuity
- Particle clustering in some region may cause insufficient particle resolution in some other region

Different versions of SPH have been developed to address these issues [289], and such modifications could also be considered to improve the model.

6.1 Future Work

Even though a new muscle model was presented, which was capable of contracting and relaxing a meshless particle system very similarly to real muscles, there is still work to be performed before it can really be used to replace a real muscle in a simulation. Some of the areas of opportunity are the following:

- Model the activation and deformation of the method using the bidomain model instead of the monodomain model. The monodomain model was selected for its simplicity, and because it is an approximation of the bidomain model. However, to achieve more accurate results, using the bidomain model would be the preferred solution.
- Instead of the FitzHugh-Nagumo cell model, a model that considers the fatigue of the tissue is needed. The ability for muscle to repeatedly generate force is limited by fatigue. The cellular mechanisms behind muscle fatigue are complex and potentially include breakdown at many points along the excitation–contraction pathway. A model such as the one presented by Shorten et al [290] could be tested.
- An important factor when using a muscle model is the force production [67]. A more detailed analysis of the force produced by the contraction and relaxation of the model is needed before it can be used in a more elaborate simulation; for example, to move the skeleton of the arm.
- For the GPGPU implementation, the divergence was a serious bottleneck in performance. However, in order to reduce or remove the issue, a rework of the method has to be considered. Warp divergence occurs when threads in the warp don't execute the same instructions, for example, due to flow-control structures. Since all threads in a warp must execute the same instruction on each cycle, if threads diverge, the warp serially executes each branch path, disabling threads that do not take that path [120]. A possibility to solve this issue would be to rework the if-statements into equivalent mathematical operations, so that the divergence is eliminated. Another would be making even numbered threads take the *if* clause and odd numbered threads take the *else* clause.
- Since the proposed method is meshless, and particle-based, it could be easily modified to simulate other tissues. The Shape Matching scheme could be replace by a mass-spring system, and the cell model could also be replaced for another to try to simulate, for example, skin. This method could be the basis of a more elaborate biological system.
- Finally, a simulation of multiple muscles and their interactions could be developed. In order for such a simulation to run in real time, the computation could be split using multiple GPUs, either in a single node, or in a cluster.

Additionally, this work serves as the base for further muscle simulations; for example, using a GPU cluster to simulate, in real time, all the interacting tissues of the arm and forearm, including bone or ligaments, and even simulate a specific movement of the arm. Force production and correct movement of the muscles would also have to be considered. The model could be used for medical or entertainment applications, such as surgical simulators, or as a replacement for Motion Capture and Key Frame animations.

6.2 Publications

As part of this thesis, the following research articles were published:

- Navarro-Hinojosa, Octavio, Sergio Ruiz-Loza, and Moisés Alencastre-Miranda. "Physically based visual simulation of the Lattice Boltzmann method on the GPU: a survey." The Journal of Supercomputing 74.7 (2018): 3441-3467.
- Navarro-Hinojosa, Octavio, and Moisés Alencastre-Miranda. "Simulation of Skeletal Muscles in Real-Time with Parallel Computing in GPU." Applied Sciences 10.6 (2020): 2099.

Bibliography

[1] Winnie Tsang, Karan Singh, and Eugene Fiume. Helping hand: an anatomically accurate inverse dynamics solution for unconstrained hand motion. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 319–328. ACM, 2005.

[2] Irene Albrecht, Jörg Haber, and Hans-Peter Seidel. Construction and animation of anatomically based human hand models. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 98–109. Eurographics Association, 2003.

[3] Sung-Hee Lee and Demetri Terzopoulos. Heads up!: biomechanical modeling and neuromuscular control of the neck. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 1188–1198. ACM, 2006.

[4] Feng Dong, Gordon J Clapworthy, Meleagros A Krokos, and Jialiang Yao. An anatomy-based approach to human muscle modeling and deformation. *IEEE Transactions on visualization and computer graphics*, 8(2):154–170, 2002.

[5] K Waters. A muscle model for animating three dimensional facial expressions. *Computer Graphic*, 21(4):123–128, 1987.

[6] Yuencheng Lee, Demetri Terzopoulos, and Keith Waters. Realistic modeling for facial animation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 55–62. ACM, 1995.

[7] Eftychios Sifakis, Igor Neverov, and Ronald Fedkiw. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Trans. Graph.*, 24(3): 417–425, July 2005. ISSN 0730-0301.

[8] Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. Comprehensive biomechanical modeling and simulation of the upper body. *ACM Trans. Graph.*, 28(4):99:1–99:17, September 2009. ISSN 0730-0301.

[9] Victor Brian Zordan, Bhrigu Celly, Bill Chiu, and Paul C DiLorenzo. Breathe easy: model and control of simulated respiration for animation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 29–37. Eurographics Association, 2004.

[10] Archibald Vivian Hill. *First and last experiments in muscle mechanics*, volume 32. Cambridge University Press Cambridge, 1970.

[11] Scott L Delp, Frank C Anderson, Allison S Arnold, Peter Loan, Ayman Habib, Chand T John, Eran Guendelman, and Darryl G Thelen. Opensim: open-source software to create and analyze dynamic simulations of movement. *IEEE transactions on biomedical engineering*, 54(11):1940–1950, 2007.

[12] Ajay Seth, Jennifer L Hicks, Thomas K Uchida, Ayman Habib, Christopher L Dembia, James J Dunne, Carmichael F Ong, Matthew S DeMers, Apoorva Rajagopal, Matthew Millard, et al. Opensim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement. *PLoS computational biology*, 14(7), 2018.

[13] Edith M Arnold, Samuel R Ward, Richard L Lieber, and Scott L Delp. A model of the lower limb for analysis of human movement. *Annals of biomedical engineering*, 38(2): 269–279, 2010.

[14] Antonie J Van den Bogert, Thomas Geijtenbeek, Oshri Even-Zohar, Frans Steenbrink, and Elizabeth C Hardin. A real-time system for biomechanical analysis of human movement and muscle function. *Medical & biological engineering & computing*, 51(10):1069–1077, 2013.

[15] Prabhav Saraswat, Michael S Andersen, and Bruce A MacWilliams. A musculoskeletal foot model for clinical gait analysis. *Journal of biomechanics*, 43(9):1645–1652, 2010.

[16] Tomas A Correa, Richard Baker, H Kerr Graham, and Marcus G Pandy. Accuracy of generic musculoskeletal models in predicting the functional roles of muscles in human gait. *Journal of Biomechanics*, 44(11):2096–2105, 2011.

[17] Robert M Kay, Sandra Dennis, Susan Rethlefsen, Richard AK Reynolds, David L Skaggs, and Vernon T Tolo. The effect of preoperative gait analysis on orthopaedic decision making. *Clinical Orthopaedics and Related Research®*, 372:217–222, 2000.

[18] Shier-Chieg Huang, I-Pin Wei, Hui-Lien Chien, Ting-Ming Wang, Yen-Hung Liu, Hao-Ling Chen, Tung-Wu Lu, and Jaung-Geng Lin. Effects of severity of degeneration on gait patterns in patients with medial knee osteoarthritis. *Medical engineering & physics*, 30 (8):997–1003, 2008.

[19] Peter A DeLuca, Roy B Davis, Sylvia Õunpuu, Sally Rose, and Robert Sirkin. Alterations in surgical decision making in patients with cerebral palsy based on three-dimensional gait analysis. *Journal of Pediatric Orthopaedics*, 17(5):608–614, 1997.

[20] Anne K Silverman and Richard R Neptune. Muscle and prosthesis contributions to amputee walking mechanics: a modeling study. *Journal of biomechanics*, 45(13):2271–2278, 2012.

[21] Tung-Wu Lu, Hsiu-Chen Lin, and Horng-Chaung Hsu. Influence of functional bracing on the kinetics of anterior cruciate ligament-injured knees during level walking. *Clinical Biomechanics*, 21(5):517–524, 2006.

[22] Tung-wu Lu, I-pin Wei, Yen-hung Liu, Wei-chun Hsu, Ting-ming Wang, Chu-fen Chang, and Jaung-geng Lin. Immediate effects of acupuncture on gait patterns in patients with knee osteoarthritis. *Chinese Medical Journal (English Edition)*, 123(2):165, 2010.

[23] FR Noyes, DS Matthews, PA Mooar, and ES Grood. The symptomatic anterior cruciate-deficient knee. part ii: the results of rehabilitation, activity modification, and counseling on functional disability. *JBJS*, 65(2):163–174, 1983.

[24] Tung-Wu Lu, Hui-Lien Chien, and Hao-Ling Chen. Joint loading in the lower extremities during elliptical exercise. *Medicine and science in sports and exercise*, 39(9):1651–1658, 2007.

[25] Tsong-Rong Jang, Chu-Fen Chang, Sheng-Chang Chen, Yang-Chieh Fu, and Tung-Wu Lu. Biomechanics and potential injury mechanisms of wrestling. *Biomedical Engineering: Applications, Basis and Communications*, 21(03):215–222, 2009.

[26] Yohan Payan. *Soft tissue biomechanical modeling for computer assisted surgery*, volume 11. Springer, 2012.

[27] Nele Famaey, Jos Vander Sloten, and Ellen Kuhl. A three-constituent damage model for arterial clamping in computer-assisted surgery. *Biomechanics and modeling in mechanobiology*, 12(1):123–136, 2013.

[28] Songbai Ji, Xiaoyao Fan, Alex Hartov, David W Roberts, and Keith D Paulsen. Estimation of intraoperative brain deformation. In *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, pages 97–133. Springer, 2012.

[29] Septimiu E Salcudean, Ramin S Sahebjavaher, Orcun Goksel, Ali Baghani, Sara S Mahdavi, Guy Nir, Ralph Sinkus, and Mehdi Moradi. Biomechanical modeling of the prostate for procedure guidance and simulation. In *Soft tissue biomechanical modeling for computer assisted surgery*, pages 169–198. Springer, 2012.

[30] Lena Maier-Hein, Peter Mountney, Adrien Bartoli, Haytham Elhawary, D Elson, Anja Groch, Andreas Kolb, Marcos Rodrigues, J Sorger, Stefanie Speidel, et al. Optical techniques for 3d surface reconstruction in computer-assisted laparoscopic surgery. *Medical image analysis*, 17(8):974–996, 2013.

[31] George Fix and Gilbert Strang. *An analysis of the finite element method.* Wellesley-Cambridge, 2nd edition, 2008.

[32] Hadrien Courtecuisse, Hoeryong Jung, Jérémie Allard, Christian Duriez, Doo Yong Lee, and Stéphane Cotin. Gpu-based real-time soft tissue deformation with cutting and haptic feedback. *Progress in biophysics and molecular biology*, 103(2):159–168, 2010.

[33] Leonidas A Spyrou and Nikolaos Aravas. Muscle and tendon tissues: constitutive modeling and computational issues. *Journal of Applied Mechanics*, 78(4):041015, 2011.

[34] Oliver Röhrle, John B Davidson, and Andrew J Pullan. A physiologically based, multi-scale model of skeletal muscle structure and function. *Frontiers in physiology*, 3:1–14, 2012.

[35] LA Spyrou and N Aravas. Muscle-driven finite element simulation of human foot movements. *Computer methods in biomechanics and biomedical engineering*, 15(9):925–934, 2012.

[36] Ted Belytschko, Yury Krongauz, Daniel Organ, Mark Fleming, and Petr Krysl. Meshless methods: an overview and recent developments. *Computer methods in applied mechanics and engineering*, 139(1-4):3–47, 1996.

[37] YuanTong Gu. Meshfree methods and their comparisons. *International Journal of Computational Methods*, 2(04):477–515, 2005.

[38] Gui-Rong Liu and Yuan-Tong Gu. *An introduction to meshfree methods and their programming.* Springer Science & Business Media, 2005.

[39] Jorge Belinha. *Meshless methods in biomechanics.* Springer, 2014.

[40] LMJS Dinis, RM Natal Jorge, and J Belinha. Analysis of 3d solids using the natural neighbour radial point interpolation method. *Computer Methods in Applied Mechanics and Engineering*, 196(13-16):2009–2028, 2007.

[41] Ken CL Wong, Linwei Wang, Heye Zhang, Huafeng Liu, and Pengcheng Shi. Meshfree implementation of individualized active cardiac dynamics. *Computerized Medical Imaging and Graphics*, 34(1):91–103, 2010.

[42] Gongfa Chen, Beat Schmutz, Devakara Epari, Kanchana Rathnayaka, Salma Ibrahim, MA Schuetz, and MJ Pearcy. A new approach for assigning bone material properties from ct images into finite element models. *Journal of biomechanics*, 43(5):1011–1015, 2010.

[43] Thomas-Peter Fries and Ted Belytschko. The extended/generalized finite element method: an overview of the method and its applications. *International Journal for Numerical Methods in Engineering*, 84(3):253–304, 2010.

[44] Radomir Chabiniok, Vicky Y Wang, Myrianthi Hadjicharalambous, Liya Asner, Jack Lee, Maxime Sermesant, Ellen Kuhl, Alistair A Young, Philippe Moireau, Martyn P Nash, et al. Multiphysics and multiscale modelling, data–model fusion and integration of organ physiology in the clinic: ventricular cardiac mechanics. *Interface focus*, 6(2), 2016.

[45] Philip Boyer and Chris Joslin. A smoothed particle hydrodynamics approach to simulation of articular cartilage. *American Journal of Biomedical Engineering*, 4(2):41–52, 2014.

[46] Grand Roman Joldes, Adam Wittek, and Karol Miller. Real-time nonlinear finite element computations on gpu–application to neurosurgical simulation. *Computer methods in applied mechanics and engineering*, 199(49-52):3305–3314, 2010.

[47] Stian F Johnsen, Zeike A Taylor, Matthew J Clarkson, John Hipwell, Marc Modat, Bjoern Eiben, Lianghao Han, Yipeng Hu, Thomy Mertzanidou, David J Hawkes, et al. Niftysim: A gpu-based nonlinear finite element package for simulation of soft tissue biomechanics. *International journal of computer assisted radiology and surgery*, 10(7):1077–1095, 2015.

[48] Vukasin Strbac, Jos Vander Sloten, and Nele Famaey. Analyzing the potential of gpgpus for real-time explicit finite element analysis of soft tissue deformation using cuda. *Finite Elements in Analysis and Design*, 105:79–89, 2015.

[49] Delia Lorente, Francisco Martínez-Martínez, María José Rupérez, MA Lago, Marcelino Martínez-Sober, Pablo Escandell-Montero, José María Martínez-Martínez, Sandra Martínez-Sanchis, Antonio J Serrano-López, C Monserrat, et al. A framework for modelling the biomechanical behaviour of the human liver during breathing in real time using machine learning. *Expert Systems with Applications*, 71:342–357, 2017.

[50] Felix Meister, Tiziano Passerini, Viorel Mihalef, Ahmet Tuysuzoglu, Andreas Maier, and Tommaso Mansi. Towards fast biomechanical modeling of soft tissue using neural networks. *arXiv preprint arXiv:1812.06186*, 2018.

[51] Nic Smith, Adelaide de Vecchi, Matthew McCormick, David Nordsletten, Oscar Camara, Alejandro F Frangi, Hervé Delingette, Maxime Sermesant, Jatin Relan, Nicholas Ayache, et al. euheart: personalized and integrated cardiac care using patient-specific cardiovascular modelling. *Journal of The Royal Society Interface*, 1(3):349–364, 2011.

[52] Victor M Spitzer and David G Whitlock. *Atlas of the visible human male: Reverse engineering of the human body.* Jones & Bartlett Learning, 1998.

[53] Joe J Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30(1):543–574, 1992.

[54] MB Liu and GR Liu. Smoothed particle hydrodynamics (sph): an overview and recent developments. *Archives of computational methods in engineering*, 17(1):25–76, 2010.

[55] Barbara Solenthaler, Jürg Schläfli, and Renato Pajarola. A unified particle model for fluid–solid interactions. *Computer Animation and Virtual Worlds*, 18(1):69–82, 2007.

[56] Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. SPH Fluids in Computer Graphics. In Sylvain Lefebvre and Michela Spagnuolo, editors, *Eurographics 2014 - State of the Art Reports*. The Eurographics Association, 2014.

[57] Markus Becker, Markus Ihmsen, and Matthias Teschner. Corotated SPH for Deformable Solids. In Eric Galin and Jens Schneider, editors, *Eurographics Workshop on Natural Phenomena*, pages 27–34. The Eurographics Association, 2009.

[58] Yasunari Zempo and Soichiro Sugimoto. Development of the SSPH Method for Real-Space Electronic Structure Calculations. In *Journal of Physics: Conference Series*, pages 12–31. IOP Publishing, 2015.

[59] Soichiro Sugimoto and Yasunari Zempo. Smoothed particle method for real-space electronic structure calculations. In *Journal of Physics: Conference Series*, volume 510, pages 12–37. IOP Publishing, 2014.

[60] MK Rausch, GE Karniadakis, and JD Humphrey. Modelling soft tissue damage and failure using a combined particle/continuum approach. *Biomechanics and modeling in mechanobiology.*, 16:249–261, 2016.

[61] Grand Joldes, George Bourantas, Benjamin Zwick, Habib Chowdhury, Adam Wittek, Sudip Agrawal, Konstantinos Mountris, Damon Hyde, Simon K Warfield, and Karol Miller. Suite of meshless algorithms for accurate computation of soft tissue deformation for surgical simulation. *arXiv preprint arXiv:1903.04760*, 2019.

[62] A Gastelum, M Krueger, J Marquez, G Gimel'farb, and P Delmas. Automatic 3d lip shape segmentation and modelling. In *Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference*, pages 1–6. IEEE, 2008.

[63] Alfonso Gastelum, Jose L Mosso, Patrice Delmas, and Jorge Marquez. A mesh-free mechanical model of the upper gastrointestinal system. In *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, pages 555–558. IEEE, 2008.

[64] Andrey Palyanov, Sergey Khayrulin, and Stephen D Larson. Application of smoothed particle hydrodynamics to modeling mechanisms of biological tissue. *Advances in Engineering Software*, 98:1–11, 2016.

[65] Davide Zerbato, Stefano Galvan, and Paolo Fiorini. Calibration of mass spring models for organ simulations. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 370–375. IEEE, 2007.

[66] Carol Oatis. *Kinesiology: The Mechanics and Pathomechanics of Human Movement.* Lippincott Williams & Wilkins, 2th edition, 2009.

[67] Margaret Nordin and Victor H. Frankel. *Basic biomechanics of the musculoskeletal system.* Lippincott Williams & Wilkins, 4th edition, 2012.

[68] Dongwoon Lee, Michael Glueck, Khan, Eugene Fiume, and Ken Jackson. A survey of modeling and simulation of skeletal muscle. *ACM Transactions on Graphics*, 28(4):162, 2010.

[69] Eleanor Criswell. *Cram's introduction to surface electromyography.* Jones & Bartlett Publishers, 2011.

[70] Felix E Zajac. Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. *Critical reviews in biomedical engineering*, 17(4):359–411, 1989.

[71] Andrew F Huxley and Ro M Simmons. Proposed mechanism of force generation in striated muscle. *Nature*, 233(5321):533–538, 1971.

[72] A.F. Huxley. Muscular contraction. *The Journal of physiology*, 243(1):1, 1974.

[73] Felix E Zajac. How musculotendon architecture and joint geometry affect the capacity of muscles to move and exert force on objects: a review with application to arm and forearm tendon transfer design. *The Journal of hand surgery*, 17(5):799–804, 1992.

[74] Richard L Lieber, Mark D Jacobson, Babak M Fazeli, Reid A Abrams, and Michael J Botte. Architecture of selected muscles of the arm and forearm: anatomy and implications for tendon transfer. *The Journal of hand surgery*, 17(5):787–798, 1992.

[75] Paul W Brand, RB Beach, and DE Thompson. Relative tension and potential excursion of muscles in the forearm and hand. *The Journal of hand surgery*, 6(3):209–219, 1981.

[76] Carl Gans. Fiber architecture and muscle function. *Exercise and sport sciences reviews*, 10(1):160–207, 1982.

[77] Richard L Lieber. *Skeletal muscle structure and function: Implications for rehabilitation and sports medicine.* Williams and Wilkins, 1992.

[78] Thomas L Wickiewicz, Roland R Roy, Perry L Powell, and V Reggie Edgerton. Muscle architecture of the human lower limb. *Clinical orthopaedics and related research*, 179: 275–283, 1983.

[79] CY Tang, G Zhang, and CP Tsui. A 3d skeletal muscle model coupled with active contraction of muscle fibres and hyperelastic behaviour. *Journal of biomechanics*, 42(7): 865–872, 2009.

[80] V Ng-Thow-Hing, A Agur, and N McKee. A muscle model that captures external shape, internal fibre architecture, and permits simulation of active contraction with volume preservation. In *International Symposium on Computer Methods in Biomechanics & Biomedical Engineering (5.: 2001: Rome). Anais. Rome*, 2001.

[81] R McN Alexander. Three uses for springs in legged locomotion. *The International Journal of Robotics Research*, 9(2):53–61, 1990.

[82] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 43–50. ACM, 1994.

[83] Gavin SP Miller. The motion dynamics of snakes and worms. In *ACM Siggraph Computer Graphics*, volume 22, pages 169–173. ACM, 1988.

[84] Oliver Röhrle. Simulating the electro-mechanical behavior of skeletal muscles. *Computing in Science & Engineering*, 12(6):48–58, 2010.

[85] G Matthias. *3D bidomain equation for muscle fibers.* PhD thesis, Fredrich Alexander Universität Erlangen Nürnberg, Germany, 2011.

[86] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117 (4):500–544, 1952.

[87] Richard FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal*, 1(6):445–466, 1961.

[88] Jinichi Nagumo, Suguru Arimoto, and Shuji Yoshizawa. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50(10):2061–2070, 1962.

[89] David Phillip Nickerson et al. *Cardiac electro-mechanics: from CellML to the whole heart.* PhD thesis, ResearchSpace@ Auckland, 2005.

[90] Craig S Henriquez. Simulating the electrical behavior of cardiac tissue using the bidomain model. *Critical reviews in biomedical engineering*, 21(1):1–77, 1993.

[91] Otto H Schmitt. Biological information processing using the concept of interpenetrating domains. In *Information processing in the nervous system*, pages 325–331. Springer, 1969.

[92] Leslie Tung. *A bi-domain model for describing ischemic myocardial dc potentials.* PhD thesis, Massachusetts Institute of Technology, 1978.

[93] Joakim Sundnes, Bjørn Fredrik Nielsen, Kent Andre Mardal, Xing Cai, Glenn Terje Lines, and Aslak Tveito. On the computational complexity of the bidomain and the monodomain models of electrophysiology. *Annals of biomedical engineering*, 34(7):1088–1097, 2006.

[94] James Keener and James Sneyd. *Mathematical Physiology.* Springer, second edition edition, 2009. ISBN 9780387758466.

[95] Robert Plonsey. Bioelectric sources arising in excitable fibers (alza lecture). *Annals of biomedical engineering*, 16(6):519–546, 1988.

[96] Jonathan P Whiteley. An efficient numerical technique for the solution of the monodomain and bidomain equations. *IEEE Transactions on Biomedical Engineering*, 53(11):2139–2147, 2006.

[97] JO Campos, Rafael Sachetto Oliveira, Rodrigo Weber dos Santos, and Bernardo M Rocha. Lattice boltzmann method for parallel simulations of cardiac electrophysiology using gpus. *Journal of Computational and Applied Mathematics*, 295:70–82, 2016.

[98] Rafel Bordas, Bruno Carpentieri, Giorgio Fotia, Fabio Maggio, Ross Nobes, Joe Pitt-Francis, and James Southern. Simulation of cardiac electrophysiology on next-generation high-performance computers. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 367(1895):1951–1969, 2009.

[99] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *ACM Siggraph Computer Graphics*, volume 21, pages 205–214. ACM, 1987.

[100] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. In *Computer graphics forum*, volume 25, pages 809–836. Wiley Online Library, 2006.

[101] Patricia Moore and Derek Molloy. A survey of computer-based deformable models. In *Machine Vision and Image Processing Conference, 2007. IMVIP 2007. International*, pages 55–66. IEEE, 2007.

[102] Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems.* SIAM, 2007.

[103] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C: the art of scientific computing.* Cambridge University Press, 1992.

[104] Ihor Farmaga, Petro Shmigelskyi, Piotr Spiewak, and Lukasz Ciupinski. Evaluation of computational complexity of finite element analysis. In *CAD Systems in Microelectronics (CADSM), 2011 11th International Conference The Experience of Designing and Application of*, pages 213–214. IEEE, 2011.

[105] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM transactions on graphics (TOG)*, 24(3): 471–478, 2005.

[106] Mathieu Desbrun, Marie-Paule Cani, et al. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics workshop on Computer animation and simulation*, volume 96, pages 61–76. Springer, 1996.

[107] J Bonet, S Kulasegaram, MX Rodriguez-Paz, and M Profit. Variational formulation for the smooth particle hydrodynamics (sph) simulation of fluid and solid problems. *Computer Methods in Applied Mechanics and Engineering*, 193(12):1245–1256, 2004.

[108] E-S Lee, Charles Moulinec, Rui Xu, Damien Violeau, Dominique Laurence, and Peter Stansby. Comparisons of weakly compressible and truly incompressible algorithms for the sph mesh free particle method. *Journal of computational physics*, 227(18):8417–8436, 2008.

[109] Paul W Cleary and Rajarshi Das. The potential for sph modelling of solid deformation and fracture. In *IUTAM symposium on theoretical, computational and modelling aspects of inelastic media*, pages 287–296. Springer, 2008.

[110] Damien Violeau and Benedict D Rogers. Smoothed particle hydrodynamics (sph) for free-surface flows: past, present and future. *Journal of Hydraulic Research*, 54(1):1–26, 2016.

[111] Robert A Gingold and Joseph J Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

[112] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159. Eurographics Association, 2003.

[113] MB Liu, GR Liu, and KY Lam. Constructing smoothing functions in smoothed particle hydrodynamics with applications. *Journal of Computational and applied Mathematics*, 155(2):263–284, 2003.

[114] XF Yang, SL Peng, and MB Liu. A new kernel function for sph with applications to free surface flows. *Applied Mathematical Modelling*, 38(15):3822–3833, 2014.

[115] JJ Monaghan. Extrapolating b splines for interpolation. *Journal of Computational Physics*, 60(2):253–262, 1985.

[116] Holger Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in computational Mathematics*, 4(1):389–396, 1995.

[117] Wenxiao Pan, Kyungjoo Kim, Mauro Perego, Alexandre M Tartakovsky, and Michael L Parks. Modeling electrokinetic flows by consistent implicit incompressible smoothed particle hydrodynamics. *Journal of Computational Physics*, 334:125–144, 2017.

[118] Herb Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's journal*, 30(3):202–210, 2005.

[119] Anthony Williams. *C++ concurrency in action.* Manning; Pearson Education, 2012.

[120] John Cheng, Max Grossman, and Ty McKercher. *Professional Cuda C Programming.* Wrox; John Wiley & Sons, Inc., 1st edition, 2014.

[121] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming, Portable Documents.* Addison-Wesley Professional, 2010.

[122] David B Kirk and W Hwu Wen-Mei. *Programming massively parallel processors: a hands-on approach.* Morgan Kaufmann, 2016.

[123] Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, and Stephen F. May. Anatomy-based modeling of the human musculature. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 163–172, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. ISBN 0-89791-896-7.

[124] Jane Wilhelms and Allen Van Gelder. Anatomically based modeling. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 173–180. ACM Press/Addison-Wesley Publishing Co., 1997.

[125] J. E. Chadwick, D. R. Haumann, and R. E. Parent. Layered construction for deformable animated characters. *SIGGRAPH Comput. Graph.*, 23(3):243–252, July 1989. ISSN 0097-8930.

[126] Laurent Moccozet and Nadia Magnenat Thalmann. Dirichlet free-form deformations and their application to hand simulation. In *Computer Animation'97*, pages 93–102. IEEE, 1997.

[127] Koji Komatsu. Human skin model capable of natural shape variation. *The visual computer*, 3(5):265–271, 1988.

[128] Jane Wilhelms. Animals with anatomy. *Computer Graphics and Applications, IEEE*, 17 (3):22–30, 1997.

[129] Juan Ramos and Caroline Larboulette. A muscle model for enhanced character skinning. *Journal of WSCG*, 21(2):107–116, 2013.

[130] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. In *ACM SIGGRAPH Computer Graphics*, volume 25, pages 251–256. ACM, 1991.

[131] Daniel Thalmann, Jianhua Shen, and Eric Chauvineau. Fast realistic human body deformations for animation and vr applications. In *Computer Graphics International, 1996. Proceedings*, pages 166–174. IEEE, 1996.

[132] Luciana Porcher Nedel and Daniel Thalmann. Real time muscle deformations using mass-spring systems. In *Computer Graphics International, 1998. Proceedings*, pages 156–165. IEEE, 1998.

[133] Amaury Aubel and Daniel Thalmann. Interactive modeling of the human musculature. In *Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings*, pages 167–255. IEEE, 2001.

[134] Victor Ng-Thow-Hing. *Anatomically-based models for physical and geometric reconstruction of humans and other animals.* PhD thesis, National Library of Canada= Bibliothèque nationale du Canada, 2001.

[135] David T Chen and David Zeltzer. *Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method*, volume 26. ACM, 1992.

[136] Qing-hong Zhu, Yan Chen, and Arie Kaufman. Real-time biomechanically-based muscle volume deformation using fem. In *Computer Graphics Forum*, volume 17, pages 275–284. Wiley Online Library, 1998.

[137] Jack T Stern Jr. Computer modelling of gross muscle dynamics. *Journal of biomechanics*, 7(5):411–428, 1974.

[138] Gentaro Hirota, Susan Fisher, A State, C Lee, and H Fuchs. An implicit finite element method for elastic solids in contact. In *Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings*, pages 136–254. IEEE, 2001.

[139] Robson Lemos, Marcelo Epstein, Walter Herzog, and Brian Wyvill. Realistic skeletal muscle deformation using finite element analysis. In *Computer Graphics and Image Processing, 2001 Proceedings of XIV Brazilian Symposium on*, pages 192–199. IEEE, 2001.

[140] Can A Yucesoy, Bart HFJM Koopman, Peter A Huijing, and Henk J Grootenboer. Three-dimensional finite element modeling of skeletal muscle using a two-domain approach: linked fiber-matrix mesh model. *Journal of biomechanics*, 35(9):1253–1262, 2002.

[141] Silvia S Blemker and Scott L Delp. Three-dimensional representation of complex muscle architectures and geometries. *Annals of biomedical engineering*, 33(5):661–673, 2005.

[142] Joseph Teran, Silvia Blemker, V Hing, and Ronald Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 68–74. Eurographics Association, 2003.

[143] Joseph Teran, Eftychios Sifakis, Silvia S Blemker, Victor Ng-Thow-Hing, Cynthia Lau, and Ronald Fedkiw. Creating and simulating skeletal muscle from the visible human data set. *Visualization and Computer Graphics, IEEE Transactions on*, 11(3):317–328, 2005.

[144] Kyung-Ha Min, Seung-Min Baek, G Lee, Haeock Choi, and Chan-Mo Park. Anatomically-based modeling and animation of human upper limbs. In *Proceedings of International Conference on Human Modeling and Animation*. Citeseer, 2000.

[145] Peter-Pike J Sloan, Charles F Rose III, and Michael F Cohen. Shape by example. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 135–143. ACM, 2001.

[146] Yong-You Ma, Hui Zhang, and Shou-Wei Jiang. Realistic modeling and animation of human body based on scanned data. *Journal of Computer Science and Technology*, 19(4): 529–537, 2004.

[147] Brett Allen, Brian Curless, and Zoran Popović. Articulated body deformation from range scan data. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 612–619. ACM, 2002.

[148] Brett Allen, Brian Curless, and Zoran Popović. The space of human body shapes: reconstruction and parameterization from range scans. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 587–594. ACM, 2003.

[149] Hyewon Seo and Nadia Magnenat-Thalmann. An automatic modeling of human bodies from sizing parameters. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 19–26. ACM, 2003.

[150] Peter Sand, Leonard McMillan, and Jovan Popović. Continuous capture of skin deformation. *ACM Transactions on Graphics (TOG)*, 22(3):578–586, 2003.

[151] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: shape completion and animation of people. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 408–416. ACM, 2005.

[152] Sang Il Park and Jessica K Hodgins. Capturing and animating skin deformation in human motion. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 881–889. ACM, 2006.

[153] Sang Il Park and Jessica K Hodgins. Data-driven modeling of skin and muscle deformation. *ACM Transactions on Graphics (TOG)*, 27(3):96, 2008.

[154] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer-Verlag, 2006.

[155] Taku Komura, Yoshihisa Shinagawa, and Tosiyasu L Kunii. A muscle-based feed-forward controller of the human body. In *Computer Graphics Forum*, volume 16, pages C165–C176. Wiley Online Library, 1997.

[156] Taku Komura, Yoshihisa Shinagawa, and Tosiyasu L Kunii. Creating and retargetting motion by the musculoskeletal human body model. *The Visual Computer*, 16(5):254–270, 2000.

[157] Shinjiro Sueda, Andrew Kaufman, and Dinesh K Pai. Musculotendon simulation for hand animation. In *ACM Transactions on Graphics (TOG)*, volume 27, page 83. ACM, 2008.

[158] Ana Lucia Cruz Ruiz. *Low-Dimensional Control Representations for Muscle-Based Characters: Application to Overhead Throwing*. PhD thesis, École normale supérieure de Rennes, 2016.

[159] Marcus G Pandy, Felix E Zajac, Eunsup Sim, and William S Levine. An optimal control model for maximum-height human jumping. *Journal of biomechanics*, 23(12):1185–1198, 1990.

[160] Frank C Anderson and Marcus G Pandy. A dynamic optimization solution for vertical jumping in three dimensions. *Computer methods in biomechanics and biomedical engineering*, 2(3):201–231, 1999.

[161] Frank C Anderson and Marcus G Pandy. Dynamic optimization of human walking. *Journal of biomechanical engineering*, 123(5):381–390, 2001.

[162] Frank C Anderson and Marcus G Pandy. Static and dynamic optimization solutions for gait are practically equivalent. *Journal of biomechanics*, 34(2):153–161, 2001.

[163] Dinesh K Pai, Shinjiro Sueda, and David IW Levin. Dynamics of biomechanisms: musculotendon mass, constraints, and architecture. *Procedia IUTAM*, 2:158–167, 2011.

[164] Marcelo Epstein, Max Wong, and Walter Herzog. Should tendon and aponeurosis be considered in series? *Journal of biomechanics*, 39(11):2020–2025, 2006.

[165] Walter Herzog. History dependence of skeletal muscle force production: implications for movement control. *Human movement science*, 23(5):591–604, 2004.

[166] Victor Ng-Thow-Hing, Anne Agur, Kevin A Ball, Eugene Fiume, and Nancy McKee. Shape reconstruction and subsequent deformation of soleus muscle models using b-spline

solid primitives. In *BiOS'98 International Biomedical Optics Symposium*, pages 423–434. International Society for Optics and Photonics, 1998.

[167] U.S. National Library of Medicine. The visible human project. http://www.nlm.nih.gov/research/visible/visible_human.html, 2003. Accessed: 2015-02-04.

[168] Victor Ng-Thow-Hing and Eugene Fiume. Physically-based modelling of musculoskeletal systems. In *Siggraph Conference Abstracts and Applications*, page 264, 1999.

[169] Josef Kohout, Gordon J Clapworthy, Saulo Martelli, and Marco Vieconti. Muscle fibres modelling. In *Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications*, pages 58–66. SciTePress, 2012.

[170] Dinesh K Pai. Strands: Interactive simulation of thin solids using cosserat models. In *Computer Graphics Forum*, volume 21, pages 347–352. Wiley Online Library, 2002.

[171] L Joshua Leon and B Milan Horáček. Computer model of excitation and recovery in the anisotropic myocardium: I. rectangular and cubic arrays of excitable elements. *Journal of electrocardiology*, 24(1):1–15, 1991.

[172] Edward J Vigmond, Felipe Aguel, and Natalia A Trayanova. Computational techniques for solving the bidomain equations in three dimensions. *Biomedical Engineering, IEEE Transactions on*, 49(11):1260–1269, 2002.

[173] JP Keener and K Bogar. A numerical method for the solution of the bidomain equations in cardiac tissue. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 8(1):234–241, 1998.

[174] Travis M Austin, Mark L Trew, and Andrew J Pullan. Solving the cardiac bidomain equations for discontinuous conductivities. *IEEE Transactions on Biomedical Engineering*, 53(7):1265–1272, 2006.

[175] EJ Vigmond, R Weber Dos Santos, AJ Prassl, M Deo, and G Plank. Solvers for the cardiac bidomain equations. *Progress in biophysics and molecular biology*, 96(1):3–18, 2008.

[176] Jonathan P Whiteley. An efficient technique for the numerical solution of the bidomain equations. *Annals of biomedical engineering*, 36(8):1398–1408, 2008.

[177] Thomas Heidlauf and Oliver Röhrle. Modeling the chemoelectromechanical behavior of skeletal muscle using the parallel open-source software library opencmiss. *Computational and mathematical methods in medicine*, 2013:14, 2013.

[178] BR Epstein and KR Foster. Anisotropy in the dielectric properties of skeletal muscle. *Medical and Biological Engineering and Computing*, 21(1):51, 1983.

[179] FLH Gielen, W Wallinga-de Jonge, and KL Boon. Electrical conductivity of skeletal muscle tissue: Experimental results from different musclesin vivo. *Medical and Biological Engineering and Computing*, 22(6):569–577, 1984.

[180] Svein Linge, Joakim Sundnes, Monica Hanslien, Glenn Terje Lines, and Aslak Tveito. Numerical solution of the bidomain equations. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 367(1895):1931–1950, 2009.

[181] Steven A Niederer, Eric Kerfoot, Alan P Benson, Miguel O Bernabeu, Olivier Bernus, Chris Bradley, Elizabeth M Cherry, Richard Clayton, Flavio H Fenton, Alan Garny, et al. Verification of cardiac tissue electrophysiology simulators using an n-version benchmark. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 369(1954):4331–4351, 2011.

[182] Prabhu Lal Bhatnagar, Eugene P Gross, and Max Krook. A model for collision processes in gases. i. small amplitude processes in charged and neutral one-component systems. *Physical review*, 94(3):511, 1954.

[183] Samuel Corre and Aziz Belmiloudi. Coupled lattice boltzmann modeling of bidomain type models in cardiac electrophysiology. In *Mathematical and Computational Approaches in Advancing Modern Science and Engineering*, pages 209–221. Springer, 2016.

[184] Mark Potse, Bruno Dubé, Jacques Richer, Alain Vinet, and Ramesh M Gulrajani. A comparison of monodomain and bidomain reaction-diffusion models for action potential propagation in the human heart. *IEEE Transactions on Biomedical Engineering*, 53(12): 2425–2435, 2006.

[185] Piero Colli Franzone and Luca F Pavarino. A parallel solver for reaction–diffusion systems in computational electrocardiology. *Mathematical models and methods in applied sciences*, 14(06):883–911, 2004.

[186] Luca F Pavarino and Simone Scacchi. Multilevel additive schwarz preconditioners for the bidomain reaction-diffusion system. *SIAM Journal on Scientific Computing*, 31(1): 420–443, 2008.

[187] John D Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E Lefohn, and Timothy J Purcell. A survey of general-purpose computation on graphics hardware. In *Computer graphics forum*, volume 26, pages 80–113. Wiley Online Library, 2007.

[188] Edward J Vigmond, Patrick M Boyle, L Joshua Leon, and Gernot Plank. Near-real-time simulations of biolelectric activity in small mammalian hearts using graphical processing units. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, pages 3290–3293. IEEE, 2009.

[189] Daisuke Sato, Yuanfang Xie, James N Weiss, Zhilin Qu, Alan Garfinkel, and Allen R Sanderson. Acceleration of cardiac tissue simulation with graphic processing units. *Medical & biological engineering & computing*, 47(9):1011–1015, 2009.

[190] Ezio Bartocci, Elizabeth M Cherry, James Glimm, Radu Grosu, Scott A Smolka, and Flavio H Fenton. Toward real-time simulation of cardiac dynamics. In *Proceedings of the 9th International Conference on Computational Methods in Systems Biology*, pages 103–112. ACM, 2011.

[191] Yaser Jararweh, Moath Jarrah, and Salim Hariri. Exploiting gpus for compute-intensive medical applications. In *Multimedia Computing and Systems (ICMCS), 2012 International Conference on*, pages 29–34. IEEE, 2012.

[192] Aurel Neic, Manfred Liebmann, Elena Hoetzl, Lawrence Mitchell, Edward J Vigmond, Gundolf Haase, and Gernot Plank. Accelerating cardiac bidomain simulations using graphics processing units. *IEEE Transactions on Biomedical Engineering*, 59(8):2281–2290, 2012.

[193] Yong Xia, Kuanquan Wang, and Henggui Zhang. Parallel optimization of 3d cardiac electrophysiological model using gpu. *Computational and mathematical methods in medicine*, 2015.

[194] Bruno Gouvêa de Barros, Rafael Sachetto Oliveira, Wagner Meira, Marcelo Lobosco, and Rodrigo Weber dos Santos. Simulations of complex and microscopic models of cardiac electrophysiology powered by multi-gpu platforms. *Computational and mathematical methods in medicine*, pages 1–13, 2012.

[195] Raphael Pereira Cordeiro, Rafael Sachetto Oliveira, Rodrigo Weber dos Santos, and Marcelo Lobosco. Improving the performance of cardiac simulations in a multi-gpu architecture using a coalesced data and kernel scheme. In *Algorithms and Architectures for Parallel Processing*, pages 546–553. Springer, 2016.

[196] Venkata Krishna Nimmagadda, Ali Akoglu, Salim Hariri, and Talal Moukabary. Cardiac simulation on multi-gpu platform. *The Journal of Supercomputing*, 59(3):1360–1378, 2012.

[197] Ronan Mendonça Amorim and Rodrigo Weber dos Santos. Solving the cardiac bidomain equations using graphics processing units. *Journal of Computational Science*, 4(5):370–376, 2013.

[198] Guillermo Vigueras, Ishani Roy, Andrew Cookson, Jack Lee, Nicolas Smith, and David Nordsletten. Toward gpgpu accelerated human electromechanical cardiac simulations. *International journal for numerical methods in biomedical engineering*, 30(1):117–134, 2014.

[199] Lei Zhang, Kuanquan Wang, Wangmeng Zuo, and Changqing Gai. G-heart: a gpu-based system for electrophysiological simulation and multi-modality cardiac visualization. *Journal of Computers*, 9(2):360–368, 2014.

[200] KHWJ Ten Tusscher, D Noble, PJ Noble, and Alexander V Panfilov. A model for human ventricular tissue. *American Journal of Physiology-Heart and Circulatory Physiology*, 286 (4):H1573–H1589, 2004.

[201] Eike M Wülfers, Zhasur Zhamoliddinov, Olaf Dössel, and Gunnar Seemann. Accelerating mono-domain cardiac electrophysiology simulations using opencl. *Current Directions in Biomedical Engineering*, 1(1):413–417, 2015.

[202] Andres Mena, Jose M Ferrero, and Jose F Rodriguez Matas. Gpu accelerated solver for nonlinear reaction–diffusion systems. application to the electrophysiology problem. *Computer Physics Communications*, 196:280–289, 2015.

[203] Rafael S Oliveira, Bernardo M Rocha, Denise Burgarelli, Wagner Meira Jr, Christakis Constantinides, and Rodrigo Weber dos Santos. Performance evaluation of gpu parallelization, space-time adaptive algorithms and their combination for simulating cardiac electrophysiology. *International Journal for Numerical Methods in Biomedical Engineering*, 2017.

[204] MH Doweidar, B Calvo, I Alfaro, P Groenenboom, and M Doblaré. A comparison of implicit and explicit natural element methods in large strains problems: Application to soft biological tissues modeling. *Computer methods in applied mechanics and engineering*, 199(25-28):1691–1700, 2010.

[205] GY Zhang, A Wittek, GR Joldes, X Jin, and K Miller. A three-dimensional nonlinear meshfree algorithm for simulating mechanical responses of soft tissue. *Engineering Analysis with Boundary Elements*, 42:60–66, 2014.

[206] Ken-ichi Tsubota, Shigeo Wada, and Takami Yamaguchi. Particle method for computer simulation of red blood cell motion in blood flow. *Computer methods and programs in biomedicine*, 83(2):139–146, 2006.

[207] Daisuke Mori, Koichiro Yano, Ken-ichi Tsubota, Takuji Ishikawa, Shigeo Wada, and Takami Yamaguchi. Computational study on effect of red blood cells on primary thrombus formation. *Thrombosis research*, 123(1):114–121, 2008.

[208] M Doblare, E Cueto, B Calvo, MA Martinez, JM Garcia, and J Cegonino. On the employ of meshless methods in biomechanics. *Computer Methods in Applied Mechanics and Engineering*, 194(6-8):801–821, 2005.

[209] James D Lee, Youping Chen, Xiaowei Zeng, Azim Eskandarian, and Morton Oskard. Modeling and simulation of osteoporosis and fracture of trabecular bone by meshless method. *International journal of engineering science*, 45(2-8):329–338, 2007.

[210] Jorge Belinha, Renato M Natal Jorge, and Lúcia MJS Dinis. A meshless microscale bone tissue trabecular remodelling analysis considering a new anisotropic bone tissue material law. *Computer methods in biomechanics and biomedical engineering*, 16(11):1170–1184, 2013.

[211] Ashley Horton, Adam Wittek, Grand Roman Joldes, and Karol Miller. A meshless total lagrangian explicit dynamics algorithm for surgical simulation. *International Journal for Numerical Methods in Biomedical Engineering*, 26(8):977–998, 2010.

[212] Suleiman Banihani, Timon Rabczuk, and Thakir Almomani. Pod for real-time simulation of hyperelastic soft biological tissue using the point collocation method of finite spheres. *Mathematical Problems in Engineering*, 2013, 2013.

[213] Ramya Rao Basava. *Meshfree Image-Based Reduced Order Modeling of Multiple Muscle Components with Connective Tissue and Fat.* PhD thesis, UC San Diego, 2015.

[214] Jiun-Shyan Chen, Ramya Rao Basava, Yantao Zhang, Robert Csapo, Vadim Malis, Usha Sinha, John Hodgson, and Shantanu Sinha. Pixel-based meshfree modelling of skeletal muscles. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 4(2):73–85, 2016.

[215] N Valizadeh, Y Bazilevs, JS Chen, and T Rabczuk. A coupled iga–meshfree discretization of arbitrary order of accuracy and without global geometry parameterization. *Computer Methods in Applied Mechanics and Engineering*, 293:20–37, 2015.

[216] Matthias Müller, Simon Schirm, Matthias Teschner, Bruno Heidelberger, and Markus Gross. Interaction of fluids with deformable solids. *Computer Animation and Virtual Worlds*, 15(3-4):159–171, 2004.

[217] Jing Qin, Wai-Man Pang, Binh P Nguyen, Dong Ni, and Chee-Kong Chui. Particle-based simulation of blood flow and vessel wall interactions in virtual surgery. In *Proceedings of the 2010 Symposium on Information and Communication Technology*, pages 128–133. ACM, 2010.

[218] Yim Pan Chui and Pheng Ann Heng. A particle-based modeling framework for thromboemboli simulation. In *Proceedings of the 11th ACM SIGGRAPH International Conference*

on Virtual-Reality Continuum and its Applications in Industry, pages 213–222. ACM, 2012.

[219] Moshiur Rahman Farazi, Bonnie Martin-Harris, Negar M Harandi, Sidney Fels, and Rafeef Abugharbieh. A 3d dynamic biomechanical swallowing model for training and diagnosis of dysphagia. In *Biomedical Imaging (ISBI), 2015 IEEE 12th International Symposium on*, pages 1385–1388. IEEE, 2015.

[220] Simone E Hieber, Jens H Walther, and Petros Koumoutsakos. Remeshed smoothed particle hydrodynamics simulation of the mechanical behavior of human organs. *Technology and Health Care*, 12(4):305–314, 2004.

[221] Andrew Kenneth Ho, Ling Tsou, Sheldon Green, and Sidney Fels. A 3d swallowing simulation using smoothed particle hydrodynamics. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 2(4):237–244, 2014.

[222] Yim-Pan Chui and Pheng-Ann Heng. A meshless rheological model for blood-vessel interaction in endovascular simulation. *Progress in biophysics and molecular biology*, 103(2): 252–261, 2010.

[223] Shikha Gupta, Jeremy Lin, Paul Ashby, and Lisa Pruitt. A fiber reinforced poroelastic model of nanoindentation of porcine costal cartilage: a combined experimental and finite element approach. *Journal of the mechanical behavior of biomedical materials*, 2(4):326–338, 2009.

[224] Barbara Solenthaler and Renato Pajarola. Predictive-corrective incompressible sph. In *ACM transactions on graphics (TOG)*, volume 28, page 40. ACM, 2009.

[225] Afonso Paiva, Fabiano Petronetto, Thomas Lewiner, and Geovan Tavares. Particle-based non-newtonian fluid animation for melting objects. In *Computer Graphics and Image Processing, 2006. SIBGRAPI'06. 19th Brazilian Symposium on*, pages 78–85. IEEE, 2006.

[226] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 49–54. ACM, 2002.

[227] Michael Hauth and Wolfgang Strasser. Corotational simulation of deformable solids. *Journal of World Society for Computer Graphics*, 2004.

[228] Joe J Monaghan. Simulating free surface flows with sph. *Journal of computational physics*, 110(2):399–406, 1994.

[229] Songdong Shao and Edmond YM Lo. Incompressible sph method for simulating newtonian and non-newtonian flows with a free surface. *Advances in water resources*, 26(7):787–800, 2003.

[230] Pierre Maruzewski, David Le Touzé, Guillaume Oger, and François Avellan. Sph high-performance computing simulations of rigid solids impacting the free-surface of water. *Journal of Hydraulic Research*, 48(S1):126–134, 2010.

[231] C Moulinec, R Issa, J Marongiu, and D Violeau. Parallel 3-d sph simulations. *Computer Modeling in Engineering and Sciences*, 25(3):133, 2008.

[232] Andreas Kolb and Nicolas Cuntz. Dynamic particle coupling for gpu-based fluid simulation. In *Proc. Symposium on Simulation Technique*, pages 722–727, 2005.

[233] Kyle Hegeman, Nathan A Carr, and Gavin SP Miller. Particle-based fluid simulation on the gpu. In *International Conference on Computational Science*, pages 228–235. Springer, 2006.

[234] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. Smoothed particle hydrodynamics on gpus. In *Computer Graphics International*, pages 63–70. SBC Petropolis, 2007.

[235] Yanci Zhang, Barbara Solenthaler, and Renato Pajarola. Adaptive sampling and rendering of fluids on the gpu. In *Volume Graphics*, pages 137–146, 2008.

[236] He Yan, Zhangye Wang, Jian He, Xi Chen, Changbo Wang, and Qunsheng Peng. Real-time fluid simulation with adaptive sph. *Computer Animation and Virtual Worlds*, 20 (2-3):417–426, 2009.

[237] Alexis Hérault, Giuseppe Bilotta, and Robert A Dalrymple. Sph on gpu with cuda. *Journal of Hydraulic Research*, 48(S1):74–79, 2010.

[238] Simon Green. Cuda particles. *NVIDIA whitepaper*, 2008.

[239] Xiaopeng Gao, Zhiqiang Wang, Han Wan, and Xiang Long. Accelerate smoothed particle hydrodynamics using gpu. In *Information Computing and Telecommunications (YC-ICT), 2010 IEEE Youth Conference on*, pages 399–402. IEEE, 2010.

[240] Øystein E Krog and Anne C Elster. Fast gpu-based fluid simulations using sph. In *International Workshop on Applied Parallel Computing*, pages 98–109. Springer, 2010.

[241] Prashant Goswami, Philipp Schlegel, Barbara Solenthaler, and Renato Pajarola. Interactive sph simulation and rendering on the gpu. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 55–64. Eurographics Association, 2010.

[242] Alejandro C Crespo, Jose M Dominguez, Anxo Barreiro, Moncho Gómez-Gesteira, and Benedict D Rogers. Gpus, a new tool of acceleration in cfd: efficiency and reliability on smoothed particle hydrodynamics methods. *PLOS ONE*, 6(6):1–13, 06 2011.

[243] Chen Huang, Jian Zhu, Hanqiu Sun, and Enhua Wu. Efficient fluids simulation and rendering on gpu. In *Proceedings of the 12th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, pages 25–30. ACM, 2013.

[244] Chen Huang, Jian Zhu, Hanqiu Sun, and Enhua Wu. Parallel-optimizing sph fluid simulation for realistic vr environments. *Computer Animation and Virtual Worlds*, 26(1):43–54, 2013.

[245] Jose M Domínguez, Alejandro JC Crespo, and Moncho Gómez-Gesteira. Optimization strategies for cpu and gpu implementations of a smoothed particle hydrodynamics method. *Computer Physics Communications*, 184(3):617–627, 2013.

[246] CUDA Nvidia. Cuda programming guide. http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf, 2017. "[Online; accessed 23-October-2017]".

[247] Qingang Xiong, Bo Li, and Ji Xu. Gpu-accelerated adaptive particle splitting and merging in sph. *Computer Physics Communications*, 184(7):1701–1707, 2013.

[248] Xiao Nie, Leiting Chen, and Tao Xiang. Real-time incompressible fluid simulation on the gpu. *International Journal of Computer Games Technology*, 2015:2, 2015.

[249] Mark Joselli, José Ricardo da S Junior, Esteban W Clua, Anselmo Montenegro, Marcos Lage, and Paulo Pagliosa. Neighborhood grid: A novel data structure for fluids animation with gpu computing. *Journal of Parallel and Distributed Computing*, 75:20–28, 2015.

[250] Xilin Xia and Qiuhua Liang. A gpu-accelerated smoothed particle hydrodynamics (sph) model for the shallow water equations. *Environmental Modelling & Software*, 75:28–43, 2016.

[251] Eugenio Rustico, Giuseppe Bilotta, Giovanni Gallo, Alexis Herault, and Ciro Del Negro. Smoothed particle hydrodynamics simulations on multi-gpu systems. In *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on*, pages 384–391. IEEE, 2012.

[252] Daniel Valdez-Balderas, José M Domínguez, Benedict D Rogers, and Alejandro JC Crespo. Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-gpu clusters. *Journal of Parallel and Distributed Computing*, 73(11):1483–1493, 2013.

[253] Eugenio Rustico, Giuseppe Bilotta, Alexis Herault, Ciro Del Negro, and Giovanni Gallo. Advances in multi-gpu smoothed particle hydrodynamics simulations. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):43–52, 2014.

[254] RM Forbes, AR Cooper, HH Mitchell, et al. The composition of the adult human body as determined by chemical analysis. *J Biol Chem*, 203(1):359–366, 1953.

[255] Tetsuya Takahashi, Issei Fujishiro, and Tomoyuki Nishita. A velocity correcting method for volume preserving viscoelastic fluids. In *Proceedings of the Computer Graphics International*, Sydney, Australia, June 2014.

[256] Robert Bridson. *Fluid simulation for computer graphics.* CRC Press, 2015.

[257] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 147–155. ACM, 2015.

[258] Andrew J Pullan, Martin L Buist, and Leo K Cheng. *Mathematically modelling the electrical activity of the heart: from cell to body surface and back again.* World Scientific, 2005.

[259] Phani Chinchapatnam, Kawal Rhode, Matthew Ginks, Prasanth Nair, Reza Razavi, Simon Arridge, and Maxime Sermesant. Voxel based adaptive meshless method for cardiac electrophysiology simulation. *Functional Imaging and Modeling of the Heart*, pages 182–190, 2009.

[260] Samuel R Ward and Richard L Lieber. Density and hydration of fresh and fixed human skeletal muscle. *Journal of biomechanics*, 38(11):2317–2320, 2005.

[261] Rachel C Payne, Robin H Crompton, Karin Isler, Russell Savage, Evie E Vereecke, Michael M Günther, SKS Thorpe, and Kristiaan D'Août. Morphological analysis of the hindlimb in apes and humans. i. muscle architecture. *Journal of anatomy*, 208(6):709–724, 2006.

[262] Samuel R Ward, Carolyn M Eng, Laura H Smallwood, and Richard L Lieber. Are current measurements of lower extremity muscle architecture accurate? *Clinical orthopaedics and related research*, 467(4):1074–1082, 2009.

[263] José Méndez. Density and composition of mammalian muscle. *Metabolism*, 9:184–188, 1960.

[264] Takanori Uchiyama and Kaito Saito. Stiffness and viscosity of the vastus lateralis muscle in cycling exercises at low constant power output. *Advanced Biomedical Engineering*, 7: 124–130, 2018.

[265] Alexis Hérault, Giuseppe Bilotta, Annamaria Vicari, Eugenio Rustico, and Ciro Del Negro. Numerical simulation of lava flow using a gpu sph model. *Annals of Geophysics*, 54 (5), 2011.

[266] Octavio Navarro-Hinojosa. Implementation of smoothed particle hydrodynamics, shape matching, and monodomain using cuda, 2020. URL `https://github.com/Hagen23/SPH_SM_MD_CUDA`.

[267] Octavio Navarro-Hinojosa. Implementation of smoothed particle hydrodynamics, shape matching, and monodomain using cuda, 2020. URL https://github.com/Hagen23/SPH-SM-Monodomain.

[268] SPH Research and engineering International Community. Sph - validation tests, 2020. URL https://spheric-sph.org/validation-tests.

[269] Aalborg University, University of Tokyo, Gävle University, and Syracuse University. Cfd benchmarks, 2020. URL https://www.cfd-benchmarks.com/.

[270] ER Galea, CS Ierotheou, P Leggett, and MK Patel. Cfdmark: A cfd-based benchmark for high-performance computer systems. *Applied mathematical modelling*, 16(11):610–614, 1992.

[271] Technische Universität Dortmund. The cfd benchmarking project, 2020. URL http://www.featflow.de/en/benchmarks/cfdbenchmarking.html.

[272] R Issa and D Violeau. 3d schematic dam break and evolution of the free surface. *R. Issa and*, 2006.

[273] Evren Bayraktar, Otto Mierka, and Stefan Turek. Benchmark computations of 3d laminar flow around a cylinder with cfx, openfoam and featflow. *International Journal of Computational Science and Engineering*, 7(3):253–266, 2012.

[274] Richard A Malinauskas, Prasanna Hariharan, Steven W Day, Luke H Herbertson, Martin Buesen, Ulrich Steinseifer, Kenneth I Aycock, Bryan C Good, Steven Deutsch, Keefe B Manning, et al. Fda benchmark medical device flow models for cfd validation. *Asaio Journal*, 63(2):150–160, 2017.

[275] Wei-Tao Wu, Fang Yang, James F Antaki, Nadine Aubry, and Mehrdad Massoudi. Study of blood flow in several benchmark micro-channels using a two-fluid approach. *International journal of engineering science*, 95:49–59, 2015.

[276] Ron Alterovitz and Ken Goldberg. Comparing algorithms for soft tissue deformation: accuracy metrics and benchmarks. *Rapport technique, UC Berkeley: Alpha Lab*, 4244, 2002.

[277] Amy E Kerdok, Stephane M Cotin, Mark P Ottensmeyer, Anna M Galea, Robert D Howe, and Steven L Dawson. Truth cube: Establishing physical standards for soft tissue simulation. *Medical Image Analysis*, 7(3):283–291, 2003.

[278] Matthew Millard, Thomas Uchida, Ajay Seth, and Scott L Delp. Flexing computational muscle: modeling and simulation of musculotendon dynamics. *Journal of biomechanical engineering*, 135(2), 2013.

[279] Nobutaka Mitsuhashi, Kaori Fujieda, Takuro Tamura, Shoko Kawamoto, Toshihisa Takagi, and Kousaku Okubo. Bodyparts3d: 3d structure database for anatomical concepts. *Nucleic acids research*, 37(suppl_1):D782–D785, 2008.

[280] G Ala, E Francomano, A Tortorici, E Toscano, and F Viola. Smoothed particle electromagnetics: A mesh-free solver for transients. *Journal of Computational and Applied Mathematics*, 191(2):194–205, 2006.

[281] Elisa Francomano, A Tortorici, E Toscano, Guido Ala, and Fabio Viola. On the use of a meshless solver for pdes governing electromagnetic transients. *Applied Mathematics and Computation*, 209(1):42–51, 2009.

[282] MS Shadloo, Amin Rahmat, and M Yildiz. A smoothed particle hydrodynamics study on the electrohydrodynamic deformation of a droplet suspended in a neutrally buoyant newtonian fluid. *Computational Mechanics*, 52(3):693–707, 2013.

[283] Octavio Navarro-Hinojosa. Simulation of triceps with points, 2020. URL https://www.youtube.com/watch?v=RuyYpptRJ0E.

[284] Octavio Navarro-Hinojosa. Simulation of meshed triceps, 2020. URL https://www.youtube.com/watch?v=EY1siYXxtNA.

[285] Steve A Maas, Gerard A Ateshian, and Jeffrey A Weiss. Febio: History and advances. *Annual review of biomedical engineering*, 19:279–299, 2017.

[286] Wendy M Murray, Thomas S Buchanan, and Scott L Delp. The isometric functional capacity of muscles that cross the elbow. *Journal of biomechanics*, 33(8):943–952, 2000.

[287] Chris Bradley, Andy Bowery, Randall Britten, Vincent Budelmann, Oscar Camara, Richard Christie, Andrew Cookson, Alejandro F Frangi, Thiranja Babarenda Gamage, Thomas Heidlauf, et al. Opencmiss: a multi-physics & multi-scale computational infrastructure for the vph/physiome project. *Progress in biophysics and molecular biology*, 107 (1):32–47, 2011.

[288] Miloš Ivanović, Boban Stojanović, Ana Kaplarević-Mališić, Richard Gilbert, and Srboljub Mijailovich. Distributed multi-scale muscle simulation in a hybrid mpi–cuda computational environment. *simulation*, 92(1):19–31, 2016.

[289] MS Shadloo, G Oger, and David Le Touzé. Smoothed particle hydrodynamics method for fluid flows, towards industrial applications: Motivations, current state, and challenges. *Computers & Fluids*, 136:11–34, 2016.

[290] Paul R Shorten, Paul O'Callaghan, John B Davidson, and Tanya K Soboleva. A mathematical model of fatigue in skeletal muscle force contraction. *Journal of muscle research and cell motility*, 28(6):293–313, 2007.