Visualization of skeletal muscle based movement
May Report

Moises Alencastre-Miranda Octavio Navarro-Hinojosa Sergio Ruiz Loza

25th May, 2015

Abstract

In order to obtain realistic animations, simulations and/or visualizations of muscle based movements, we want to generate a model based on Lattice Boltzmann and biophysical activation models, to best simulate how these work, and generate more realistic and accurate human movement animations and simulations. Here, we present the progress that has been achieved in different areas, including development of a Lattice Boltzmann solver using the CPU, in CUDA using Thrust, and the CUDA API, and development of a user interface to render and present the simulations of the proposed model.

1 Introduction

Human character's animations and simulations are used in many fields: from video games and movies to robotic and medical simulations. Most of the animations of virtual human characters, the visualization or simulation of human movements, and many humanoid robot movements issues, are commonly solved assuming that the movement of each bone or extremity is produced in each joint, as if there were a motor in each one producing the movement. However, said approach does not simulate how real movement is generated, and does not produce realistic movements. In order to achieve more realistic movement, they should be based on skeletal muscles.

1.1 Skeletal Muscles

Skeletal muscles are among the most important structures in the human body. They are the most abundant tissue in the body (between 40% and 45% of the total body weight), they provide protection for internal tissues, they maintain the body's posture, and are a key component for force generation and movement. Skeletal muscle contraction is controlled through the somatic nervous system and, for the most part, is done so consciously. These voluntary contractions produce forces which transfer to the underlying skeleton, resulting in human body movement.

Each muscle is formed by two units: the muscle belly, and two tendinous units at each end of the muscle belly that connect it to the related bone. Figure 1 shows the hierarchical structure of the different tissues that compose a skeletal muscle. Skeletal muscles are wrapped by the episysium, a dense connective tissue which joins with the tendon. Internally, the muscle is composed of numerous muscle fiber bundles, called fascicles, which are separated from one another by a layer of connective tissue knowns as the perimysium. In turn, every fascicle consists of muscle fibers, which are isolated from one another by the endomysium. Muscle fibers are the structures that generate the contraction in a muscle. These are activated by motor neurons that receive activation signals from the nervous system. Each motor neuron activates a group of fibers, and each group of fibers and motor neuron is called the motor unit.

Another important component to be considered is tendon. It transmits forces produced by the attached muscle to bone. Tendon connects muscle to bone either at a narrow area or over a wide and flattened area, known as the aponeurosis. The attachment of muscle to more stationary bone (i.e., the proximal site) is called the origin while the other end to more movable bone (i.e., distal site) is called the insertion. Tendons are mostly composed of parallel arrays of collagen fibers closely packed together and have the mechanical property that they are much stiffer than muscles when they are pulled. In addition to force transmission, tendon has a function to passively modulate force during locomotion, providing additional stability [1, 2].

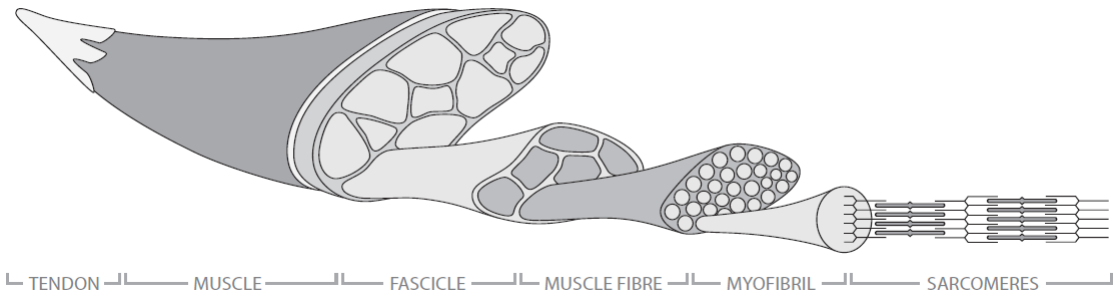


FIGURE 1: Major components of the muscle. Adapted from [2].

1.2 Activation models

In order to simulate the muscle's contraction and generate force and movement, there are two main approaches: phenomenological and biophysical models [3, 4]. Phenomenological models use mathematical and physics based constructs to describe the mechanical properties of biological tissue; in this case, skeletal muscles. One of the most used model is the Hill muscle model [5]. This model is based on a series of experiments conducted on frog muscles, and captures the mechanical properties of the muscle. It has three major components: the series element (SE), the parallel element (PE), and the contractile element (CE). The series element (SE) represents mainly the elastic effects of tendon and intrinsic elasticity within the sarcomere. The parallel element (PE) represents the passive elasticity of the muscle resulting from the penetration of connective tissues into the muscle body. The contractile element (CE) accounts for generation of active force which is dependent on the muscle length, and the time-varying neural signal, a(t), originating from the central nervous system.

On the contrary, biophysical models use physics to try to model biological systems. For muscle contraction, these models try to predict the muscle's response to a determinate stimulus. One commonly used biophysical model is the Huxley muscle contraction theory, that analyses the electrical activity of muscle fibers in order to generate movement. However, the Bidomain model [6] is better suited at modelling the electric activity throughout a biological tissue. In the case of muscle contraction, based on a stimulus it calculates the muscle fiber's state before, during, and after a muscle contraction.

1.3 State of the art

Since skeletal muscle are fundamental to maintain pose and to generate movement, there have been many studies that try to model the musculoskeletal system. For brevity, we will only mention the ones that have been relevant in the recent years.

Lee and Terzopoulos [7] developed a biomechanical model of the upper body. Their model is capable of modelling and controlling the muscles and bones, as well as simulating the physics-based deformations of the soft tissues. They incorporated 814 muscles, each of which is modeled as a piecewise uniaxial Hill-type force actuator. To simulate biomechanically-realistic flesh deformations, they developed a coupled finite element model with the appropriate constitutive behavior, in which are embedded the detailed 3D anatomical geometries of the hard and soft tissues. Finally, they created an associated physics-based animation controller that computes the muscle activation signals necessary to drive the elaborate musculoskeletal system in accordance with a sequence of target poses specified by an animator. A sample of their model can be seen in Figure 2.

Unlike the work of Lee and Terzopoulos that use phenomenological models to generate force and movement, the work of Röhrle [4, 6] uses the Bidomain model to describe the electrical activity and the subsequent contraction of the muscle fibers. The result is a physiologically based, multi-scale skeletal muscle finite element model that is capable of representing detailed, geometrical descriptions of skeletal muscle fibers and their grouping.

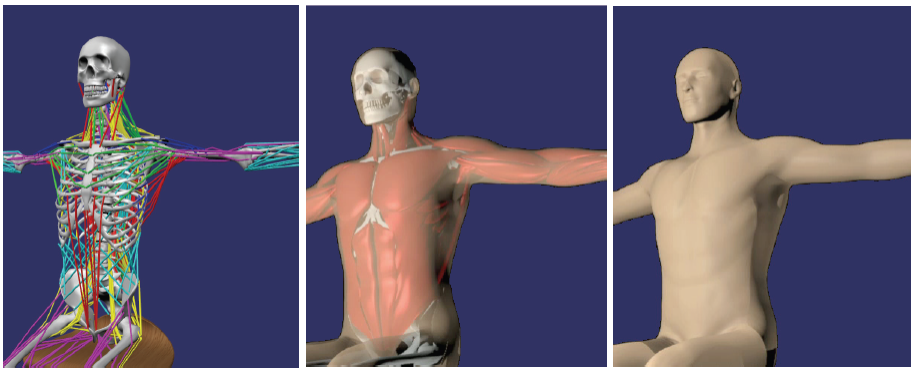


FIGURE 2: A skeleton moved by muscle modelled with a piecewise uniaxial Hill-type force actuator (left). The generated movement deforms the soft tissues (center), and the skin (right). Adapted from [7].

Both previous works focus on a muscle model that has a physically based activation model, and use the Finite Element Method (FEM) to represent the muscle. The work of Fan et. al. [8] focuses more on the visual accuracy rather than on the activation method. They introduce a framework for simulating the dynamics of musculoskeletal systems, with volumetric muscles in close contact and novel data-driven muscle activation model. Muscles are simulated using an Eulerian-on-Lagrangian discretization that handles volume preservation, large deformation, and close contact between adjacent tissues. Volume preservation is crucial for accurately capturing the dynamics of muscles and other biological tissues. Their model utilizes knowledge of the active shapes of muscles, which can be easily obtained from medical imaging data or designed to meet artistic needs. Their framework can be seen in Figure 3.

1.4 Problem statement

The musculoskeletal system consists of several biological structures, and is a crucial component for the analysis and simulation of human movement. However, it is inherently difficult to simulate this system, not just because of the complexity of the structures that conform it, but also because of all the relations between them that have to be taken into account.

There are many challenges that have to be resolved in order to correctly simulate the musculoskeletal system. Some of them are as follows:

- Most current simulations are simplifications of the overall system; either by simplifying the structures that the model uses, or the way they are controlled.
- The way in which the tissues are simulated not always accounts for certain key properties, such as volume conservation, or bulging; or are simulated using techniques such as FEM, that are computationally expensive, and more so if such properties are to be considered.
- Most previous work does not model the muscle fibers, which are crucial for a correct muscle simulation.
- The mainly used activation methods are phenomenological. These are simplifications of the real behaviour of muscles, and have been proven to be wrong when comparing their output to real muscle output data. These models are more similar to simple mechanic systems than to biological systems.
- The physical properties of the several components of the musculoskeletal system are normally not taken into account.
- Bones, ligaments, and tendons are either simplified or ignored for most simulations.

Currently, there does not exist a model of the musculoskeletal system that simulates the muscles considering their internal structure, the biological tissues that they are composed of, and their physical and biological properties, that is also activated using biophysical models that emulate how they respond to a signal from the nervous system. Additionally, most previous work focus on modelling a specific muscle, or a group of specific muscles, without providing a framework that can help model any muscle of the body.

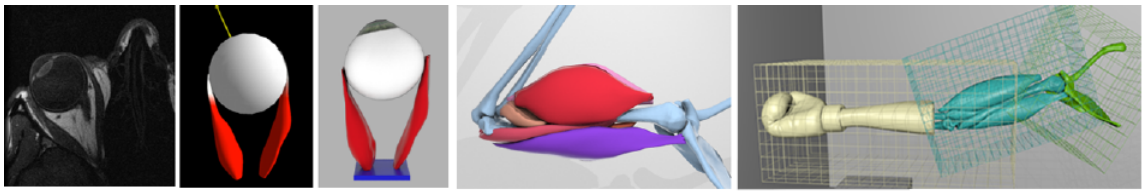


FIGURE 3: The framework spans the entire pipeline for simulating volumetric musculoskeletal systems, starting from data on the active shapes of muscles to physically based simulation of muscles and soft tissues coupled to the skeletal system. The framework starts with muscle MRI data (A), then a reconstruction of the shape of the muscle is done in a 3D modelling software (B), and a simulation with the proposed method is generated (C). A set of six muscles of the arm are modelled (D), and those muscles are used in a dynamic simulation, with real bulging, tissue deformation, and contact with a simulated environment (E) [8].

Furthermore, the computational cost to solve the different mathematical, and physical models, as well as the rendering of the various objects in the scene, can be high. However, not many studies rely on General Purpose computing on Graphics Processing Units (GPGPU), making their solutions not ideal for interactive or real time simulations.

1.5 Proposed method

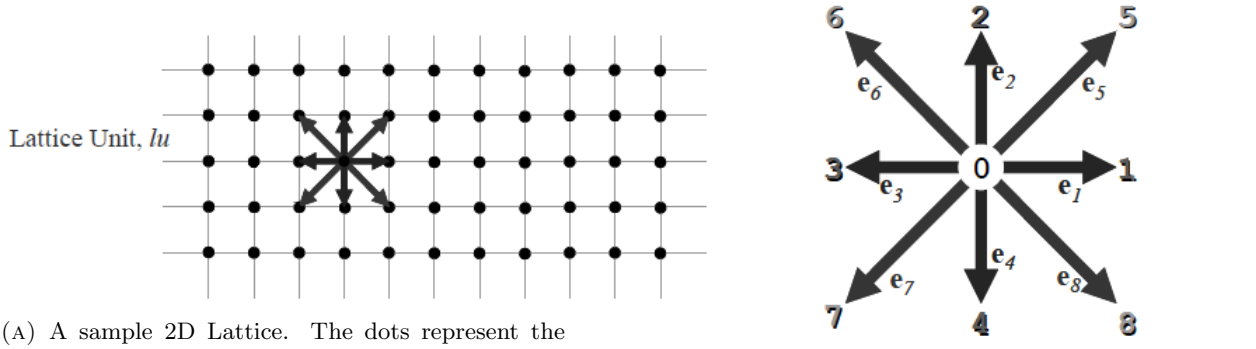
The first issue that needs to be solved is the simplification of the musculoskeletal system. We propose a model that encompasses several of the principal tissues and structures that are present in every muscle: the muscle fibers, connecting tissues, tendons, bones, and skin. Since the human body is comprised mostly of water, most of these tissues could be modelled using computational fluid dynamics (CFD) methods for fluid simulation. In this case, we propose the use of a modified Lattice Boltzmann Method (LBM) to model all the mentioned tissues.

Normally, the LBM models the flow of a fluid within a container. For this solution, the fluids are going to be the tissues of the body, each with their own material properties. We will use the lattice's speeds and material properties to simulate a very viscous, almost solid fluid, that modifies its container when needed (such as when an external force is present). This allows the tissue to have basic biological properties such as being deformable, incompressible, having volume preservation, and allowing the contact with other tissues.

The proposed simulation of tissues requires the use of a very detailed human 3D model from which the containers can be obtained. We propose the use of BodyParts3D: 3D structure database for anatomical concepts [9] from the Database Center for Life Science [10]. BodyParts3D is a dictionary-type database for anatomy in which anatomical concepts are represented by 3D structure data that specify corresponding segments of a 3D whole-body model for an adult human male. It encompasses morphological and geometrical knowledge in anatomy. BodyParts3D was constructed on the framework of a voxel human model for electromagnetic dosimetry, 'TARO', which was created from a whole-body set of 2 mm interval MRI images of a male volunteer.

Once we have a model that is able to represent a biological tissue, we propose the use of a biophysical model in order to simulate the activation of the tissue. In the case of skeletal muscles, the Bidomain model has been used to model their electrical activity [4]. We will solve a Bidomain equation for each muscle fiber present in each muscle we simulate. However, since motor units innervate several muscle fibers, a simplification where we only solve a bidomain model for each motor unit of the muscle could be considered.

In order to test our proposed method, we will simulate two simple motion of one upper limb of a human body. We will simulate the extension and flexion of the arm around the elbow. We decided to simulate only said movements because they are produces by the activation of only a small number of muscles (biceps brachii, brachioradialis, brachialis, pronator teres, triceps brachii, and anconeus). However, our method could be applied to other muscles and could be used to produce a wide range of movements.



(A) A sample 2D Lattice. The dots represent the nodes of the lattice.

(B) Each Lattice node has 8 possible velocity directions.

FIGURE 4: D2Q9 lattice. Adapted from [11].

Finally, since we will have to solve several mathematical and physical models in order model the biological tissues and their activations, the use of GPGPU will be essential if we intend to produce an interactive simulation. We propose the use of the CUDA API to implement the LBM, and the Bidomain model. We will render all the elements of the simulation using shaders that also run on a Graphics Processing Unit (GPU).

2 Lattice Boltzmann Models

Lattice Boltzmann models (LBM) is a class of computational fluid dynamics (CFD) methods for fluid simulation. Instead of solving the Navier–Stokes equations, the discrete Boltzmann equation is solved to simulate the flow of a Newtonian fluid with collision models such as Bhatnagar-Gross-Krook (BGK). By simulating streaming and collision processes across a limited number of particles, the intrinsic particle interactions create a flow behavior applicable across an entire lattice.

Lattice Gas Cellular Automaton (LGCA) models were the harbingers of LBM. LGCA were presented as a viable means of solving the Navier-Stokes equations of fluid motion. Ludwig Boltzmann used these models to introduce the Boltzmann Gas Concept, where the idea is that a gas is composed of interacting particles that can be described by classical mechanics, and, because there are so many particles, a statistical treatment is necessary and appropriate. The mechanics can be extremely simple and encapsulated by just the notions of streaming in space and billiard-like collision interactions.

LBM vastly simplify Boltzmann's original conceptual view by reducing the number of possible particle spatial positions and microscopic momenta from a continuum to just a handful and similarly discretizing time into distinct steps. Particle positions are confined to the nodes of the lattice. Variations in momenta that could have been due to a continuum of velocity directions and magnitudes and varying particle mass are reduced, in the simple 2-D model, to 8 microscopic velocity directions, and a single particle mass [11].

Using these microscopic velocities, we can obtain a macroscopic velocity, which is the one that dictates the flow within a lattice. Figure 4 shows the Cartesian lattice and the velocities e_a where a = 0, 1, ..., 8 is a direction index and $e_0 = 0$ denotes particles at rest. This model is known as D2Q9 as it is 2 dimensional and contains 9 velocities. The lattice unit (lu) is the fundamental measure of length in the LBM models and time steps (ts) are the time unit.

Other fundamental element of the LBM is the single-particle distribution function f. The distribution function can be thought of as a typical histogram representing a frequency of occurrence. The frequencies can be considered to be direction-specific fluid densities.

These models can also be implemented in three dimensions. The main difference with the two dimension models are the amount of velocities and their directions. There are several possibilities, from models that

use twenty seven direction, to models that use only thirteen. However, the most used model is the one that has nineteen velocities, the d3q19 model, because it yields the best equilibrium between precision and used memory [12].

2.1 Lattice Boltzmann Solver

In order to develop a LBM solver, we have to take into account the basic parts and equations that define the models. These are:

- Streaming: Each node's distribution function is updated in relation to the other nodes in the lattice. Specifically, the direction-specific densities f_a to the nearest neighbor lattice nodes. The streaming equation is as follows:
$$f_a(x + e_a\Delta t, t + \Delta t) = f_a(x, t) \tag{1}$$
where x is the lattice node position, e_a is the direction index, t is the time step, and Δt is the next step in the simulation.
- Collision: Collision of the fluid particles is considered as a relaxation towards a local equilibrium. The collision equations are as follows:
$$f_a(x, t) = f_a(x, t) - \frac{f_a(x, t) - f_a^{eq}(x, t)}{\tau} \tag{2}$$
where f_a^{eq} is the equilibrium function, and τ is a relaxation parameter that controls the fluid viscosity. The equilibrium function is defined as:
$$f_a^{eq}(x, t) = w_a\rho(x)[1 + \frac{e_a \cdot u}{c^2} + \frac{(e_a \cdot u)^2}{2c^4} - \frac{u^2}{2c^2}] \tag{3}$$
where w_a are weights corresponding to each lattice velocity, ρ is the macroscopic density, and c is the basic speed of the lattice.
- Boundary Condition: We have to take into account the boundaries of the lattice since the nodes situated at the edges have no neighbor cells from which to obtain a distribution in the streaming step. We can also incorporate solid boundaries in the models, in order to simulate realistic porous media, for example.

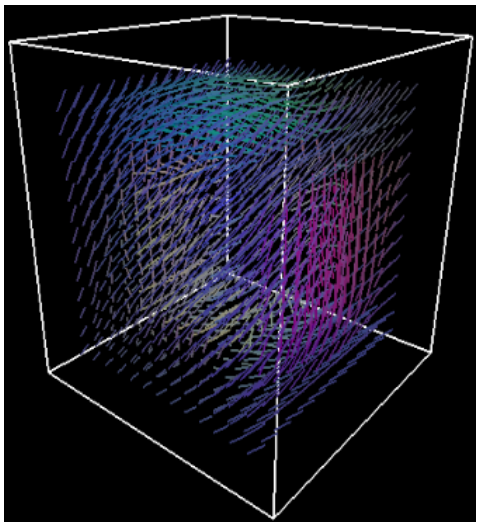
For more details on the formulation of LBM and its equations, refer to [11, 12] and the references mentioned within.

We have implemented the d3q19 model using the CUDA API. Since we want to simulate biological tissue, normal boundary conditions do not apply; we only need the solid boundary conditions. We surrounded the lattice with solid elements, which serve as a container, in order to get the fluid to "flow" inside said container.

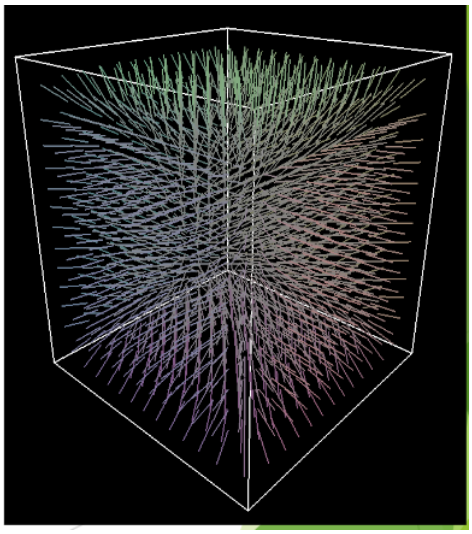
Figure 5a shows a sample lattice were the the macroscopic velocities are displayed. To better represent the flow within the sample lattice, we induced the flow by defining a direction vector, and applying it at a determinate point of the lattice. We created a cube within the lattice in order for the macroscopic velocities to move around said cube. Figure 5b shows a different lattice, where no flow direction vector is added. We found that this lattice is in an equilibrium state, and all the macroscopic velocities flow toward the outside of the lattice. This is a property of the LBM that will enable us to create tissues that are able to preserve their volume.

2.2 Solver details and results

We tested two different approaches for the CUDA implementation of the Lattice Boltzmann Solver. The first approach was using Thrust. Thrust is a C++ template library for CUDA based on the Standard Template Library (STL). It allows the implementation of high performance parallel applications with minimal programming effort through a high-level interface that is fully interoperable with CUDA C [13]. Using thrust the development of the solver was straightforward. After a CPU version was developed, we used the *thrust::for_each* algorithm to map the different steps of the solver to each lattice node using different kernels.



(A) A 3D lattice with an external flow.



(B) A 3D lattice without an external flow.

FIGURE 5: Implementation of the 3D Lattice Boltzmann method using CUDA.

The second approach was using the CUDA API directly. For that approach, we designed two main kernels, one for the streaming and one for the collide steps. The stream kernel is configured using a bidimensional array of blocks defined by the lattice width and height, and a bidimensional array of threads; this array will vary in one dimension, and will be fixed to nineteen in the other, this because the first dimension will process the lattice depth, and the nineteen additional threads will process each lattice element velocities. We varied the size of the thread array to try to obtain optimal execution times. For the collision step, we found that using fixed block and thread sizes we obtained the best results. We used a bidimensional array of blocks defined by the lattice width and height, and a number of threads equal to the lattice depth.

In order to test the performance of our solution, we limited a simulation to a thousand iterations of the solver, and we measured the execution times of the streaming and collide steps for different grid sizes. We also measured the performance in terms of Million Lattices Updates Per Second (MLUPS) and GPU/CPU speedup. Here, the MLUPS is given by:

$$MLUPS = \frac{gridSize * 10^{-6}}{iterationtime(s)} \tag{4}$$

All our test were performed on a PC with the following specifications:

Testing PC specifications	
Processor	Intel Core i5, 3.0 MHz
RAM	8192.0 MB
GPU	NVIDIA GeForce GTX 670, 1344 CUDA Cores
OS	Windows 7

Table 1 shows the average times of a simulation of the solver in CPU.

Based on the times obtained using just the CPU, in Table 2 we present the average times, speedup, and MLUPS for a simulation using CUDA THRUST. As it will be seen in the following tables, the thrust implementation yielded good results, but as we give up a certain amount of control, we can not control the simulation as much as needed to improve performance. Another issue was that we were not able to

LBM using CPU				
Lattice Size	**Stream Avg (ms)**	**Collide Avg (ms)**	**Total Avg (ms)**	**MLUPS Avg**
30x30x30	2.5697	4.2447	6.8145	3.9995
50x50x50	12.5627	21.4232	33.9860	3.6944
100x100x100	110.4029	198.6268	309.0298	3.2368
132x132x132	258.7472	464.9450	723.6923	3.1783

Table 1: LBM simulation times using the CPU.

process lattices with sizes greater than one hundred because of a thrust defined error. We will need to check if said mistake can be resolved.

LBM using GPU, CUDA THRUST					
Lattice Size	**Stream Avg (ms)**	**Collide Avg (ms)**	**Total Avg (ms)**	**MLUPS Avg**	**Speedup**
30x30x30	0.2751	3.9406	4.2158	6.4045	1.6164
50x50x50	1.2667	17.1502	18.4169	6.7992	1.8453
100x 100x100	12.5523	135.9719	148.5243	6.7342	2.0806

Table 2: LBM simulation times using the GPU and the CUDA Thrust template library.

In Table 3 we present the average times, speedup, and MLUPS for a simulation using the CUDA API and an array of 8 by 19 threads for the streaming step. Table 4 shows the results of the same simulation for an array of 53 by 19 threads.

LBM using GPU, 8 Threads					
Lattice Size	**Stream Avg (ms)**	**Collide Avg (ms)**	**Total Avg (ms)**	**MLUPS Avg**	**Speedup**
30x30x30	0.1984	0.6624	0.8609	31.3618	7.9151
50x50x50	1.1635	2.7446	3.9081	31.9848	8.6961
132x132x132	27.3471	86.4619	113.8091	20.2109	6.3588

Table 3: LBM simulation times using the GPU with the CUDA API and an array of 8 by 19 threads to calculate the streaming step.

As can be seen from the performance of the solver from the several tables, we still have some work to do in order to improve its performance and achieve interactive execution times for both a large lattice and for the solution of several lattices. We will modify the solver in order to use the shared memory of the GPU and try to improve performance even further.

3 Muscle Visualization

In order to test, evaluate and later validate the proposed method, we started the development of a visualization software tool that is able to display geometry efficiently by using parallel hardware (GPU) techniques; the visualization software will also provide an interface to control simulation parameters.

3.1 Visualization Implementation

By using a set of third party software tools, we assembled a rendering engine (Figure 6) that allows efficient display and control of a simulation that uses the proposed method. Open Source libraries for rendering, shading, graphical user interface, texture and geometry loading have been integrated to the visualization software; these tools are listed in Table 5.

LBM using GPU, 53 Threads					
Lattice Size	**Stream Avg (ms)**	**Collide Avg (ms)**	**Total Avg (ms)**	**MLUPS Avg**	**Speedup**
30x30x30	0.4269	0.6627	1.0896	24.7784	6.2536
50x50x50	1.714	2.6427	4.3570	28.7034	7.8001
132x132x132	49.1889	87.4510	136.64004	16.8329	5.2963

TABLE 4: LBM simulation times using the GPU with the CUDA API and an array of 53 by 19 threads to calculate the streaming step.

TABLE 5: Third party software for virtual muscle rendering

Third party software tools	
Rendering Library	OpenGL
Shading	GLSL
GUI	Qt 5.4.1 Community
Texture loader	DevIL
Geometry loader	ASSIMP

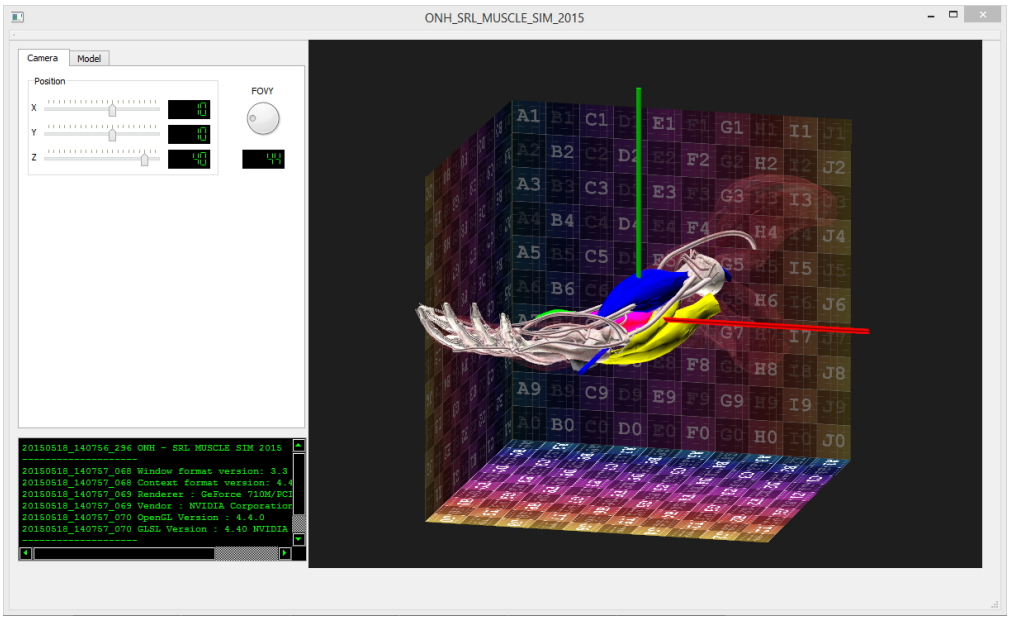
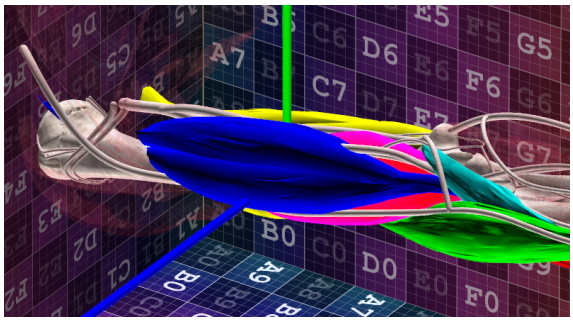


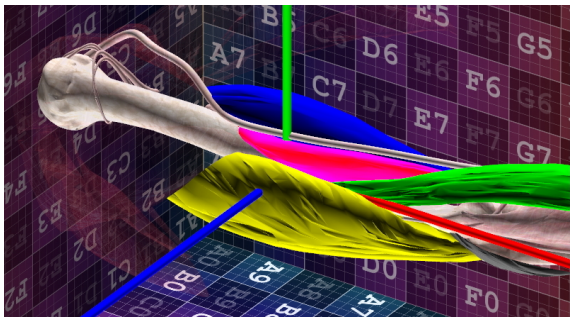
FIGURE 6: Muscle rendering tool.

3.2 Rendering Advances

From the BodyParts3d [14] database, we have identified, colored and separated virtual muscle geometries (Figure 7) of interest—right biceps brachii, brachialis, brachiordialis (Figure 7a), pronator teres, triceps brachii, anconeus (Figure 7b)—that will serve as input for the muscle simulation. Together, all the arm meshes are composed of 271,004 faces which are rendered at interactive frame rates: up to 0.00588235 seconds per frame using a Nvidia GeForce 710M GPU as shown in Figure 8, using transparent (Figure 8a) or opaque (Figure 8b) geometry, thus allowing for simulation computations to be performed even on the same hardware.

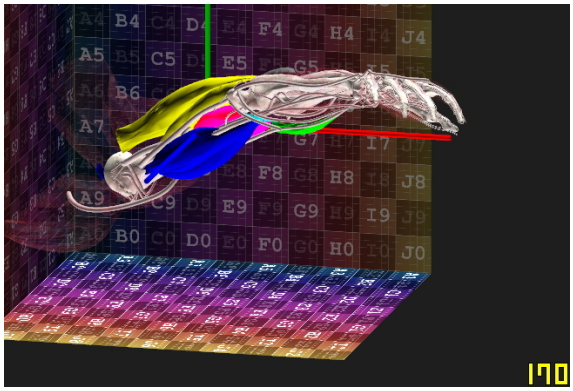


(A) Blue: biceps brachii. Purple: brachialis. Green: brachiordialis. Aqua: pronator teres.

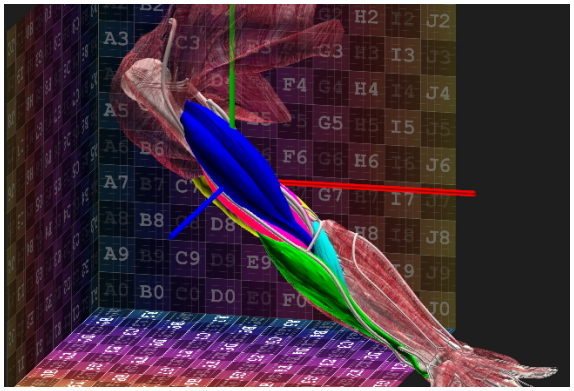


(B) Yellow: triceps brachii. Gray: anconeus.

FIGURE 7: Muscle geometry



(A) Interactive rate rendering.



(B) Transparency control.

FIGURE 8: Muscle rendering

4 Future Work

In addition to the Lattice Boltzmann Solver and Visualization integration, we have identified several future work opportunities:

Octrees Vertex identification as part of a simulated muscle will be efficient by means of a specialized data structure. We will integrate the Point Cloud Library (PCL) to our rendering engine making use of its parallel octree implementation.

Level of Detail As computational resources become scarce due to real-time simulation and rendering requirements, implementation of a Level of Detail technique will improve hardware performance.

Clipping Sections of a virtual extended arm will be useful to calculate intermuscular space from different poses, as an extension of the muscular simulation.

Simulation control LBM simulations require fine tuning of viscosity and grid size parameters. Future work will extend the user interface to facilitate parameter adjustment in real-time.

Bidomain Fiber simulation We will implement a fiber simulation and rendering as the basic activation unit for our model, based on the Bidomain simulation. Bidomain Fiber simulation force will be the input to the LBM mesh deformation algorithm.

Inter mesh Collision In future work we will solve intermuscular collision and tension forces, adding precision to the muscle simulation.

Muscle-Bone Hierarchy For animation, we will establish not only the bone rigging hierarchy, but also the various insertion points for tendons, achieving precise physics-based simulations activated by the Bidomain-LBM algorithm.

Kernel configuration We will test further Kernel (grid/block/trhead) launch configurations as well as other LBM models to ensure optimal algorithm performance.

5 Acknowledgments

This project is supported by the Google Faculty Research Awards. This is the first progress report for the ongoing project.

References

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

[2] 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.

[3] 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.

[4] 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, 2012.

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

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

[7] 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. doi: 10.1145/1559755.1559756. URL http://doi.acm.org/10.1145/1559755.1559756.

[8] Ye Fan, Joshua Litven, and Dinesh K Pai. Active volumetric musculoskeletal systems. *ACM Transactions on Graphics (TOG)*, 33(4):152, 2014.

[9] 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, 2009.

[10] Database Center for Life Science. Database center for life science. http://dbcls.rois.ac.jp/en, 2015.

[11] MC Sukop and DT Thorne Jr. Lattice boltzmann modeling: An introduction for geoscientists and engineers. *ISBN 3540279814. URL*, 29, 2006.

[12] Pablo R Rinaldi. *Modelos de autómatas celulares sobre unidades de procesamiento gráfico de alta performance*. PhD thesis, Universidad Nacional de Cuyo, 2011.

[13] NVIDIA Corporation. Thrust :: Cuda toolkit documentation. http://docs.nvidia.com/cuda/thrust/, 2015.

[14] The Database Center for Life Science. Bodyparts3d. http://dbarchive.biosciencedbc.jp/en/bodyparts3d/download.html, 2011. BodyParts3D, (c) The Database Center for Life Science licensed under CC Attribution-Share Alike 2.1 Japan.