MUSCLE ACTIVATION MAPPING OF SKELETAL HAND MOTION: AN EVOLUTIONARY APPROACH

ARUN SOMASEKHARAN

A thesis submitted in partial fulfilment of the requirements of Bournemouth University for the degree of Doctor of Philosophy

August 29, 2012

MUSCLE ACTIVATION MAPPING OF SKELETAL HAND MOTION: AN EVOLUTIONARY APPROACH

ARUN SOMASEKHARAN

Supervisors:

Prof. Jian Jun Zhang
Dr. Hammadi Nait-Charif
Dr. Xiaosong Yang

Abstract

Creating controlled dynamic character animation consists of mathematical modelling of muscles and solving the activation dynamics that form the key to coordination. But biomechanical simulation and control is computationally expensive involving complex differential equations and is not suitable for real-time platforms like games. Performing such computations at every time-step reduces frame rate. Modern games use generic software packages called *physics engines* to perform a wide variety of in-game physical effects. The physics engines are optimized for gaming platforms. Therefore, a physics engine compatible model of anatomical muscles and an alternative control architecture is essential to create biomechanical characters in games.

This thesis presents a system that generates muscle activations from captured motion by borrowing principles from biomechanics and neural control. A generic physics engine compliant muscle model primitive is also developed. The muscle model primitive forms the motion actuator and is an integral part of the physical model used in the simulation.

This thesis investigates a stochastic solution to create a controller that mimics the neural control system employed in the human body. The control system uses evolutionary neural networks that evolve its weights using genetic algorithms. Examples and guidance often act as templates in muscle training during all stages of human life. Similarly, the neural controller attempts to learn muscle coordination through input motion samples. The thesis also explores the objective functions developed that aids in the genetic evolution of the neural network.

Character interaction with the game world is still a pre-animated behaviour in most current games. Physically-based procedural hand animation is a step towards autonomous interaction of game characters with the game world. The neural controller and the muscle primitive developed are used to animate a dynamic model of a human hand within a real-time physics engine environment.

Table of Contents

List of Figures

List of Tables

Acknowledgements

This is the culmination of a journey that taught me not just academic subjects, but also helped me forge and strengthen relationships between old and new acquaintances.

I dedicate this thesis to my wife, Aarti and son, Aarav. In retrospect, without them in my life, I wouldn't have had the mental courage to finish what I started. Hats off to the sheer amount of patience my wife displayed throughout my long journey and relieving me of parent duties so that I could work in peace and for tolerating my "lost in deep thought" moments. Thank you so much, Aarti.

I thank my parents and my brother and my in-laws for their complete faith in me and the continuous support and strength they provided me. This is for all of them. Without their help, this would not have been possible. I hope I have made them proud. I would like to especially thank my father-in-law, Kamlesh Pande, for enlightening me about vectors which was crucial in defining the objective functions.

I would like to thank my supervisors, Professor Jian Jun Zhang and Dr. Xiaosang Yang for providing valuable advice and moral support through the course of the programme.

Special thanks to Jan Lewis and Daniel Cox for their constant moral support, encouragement and all the non-academic help that I required. Jan, as promised, here is my thesis for your shelf.

Big thanks to all of my friends in the Research Office (Leigh, Sola, Albert, Dennis, Rudra, Shihui, Safa) for the encouragement and great discussions which in some way contributed to and helped my research. Rudra, the machine learning discussions we had were crucial and absolutely helpful. Sola, talks with you prompted me to think deeper and critically at the work I did. Albert, thanks a bunch for keeping me company on the graveyard shift. I am glad we helped each other out. Chindu, thanks for all the encouragement and morale boost and for introducing me to Geoff and Linda. A lot of gratitude goes to Linda and Jeff for providing me a beautiful place to stay in Bournemouth. Geoff, thanks for being the guinea pig for my curry experiments.

A special thanks and gratitude to Dr. Hammadi Nait-Charif (who is also my third supervisor) for the invaluable discussions on Artificial Neural Networks and also for spending time for long discussions. Your suggestions were absolutely instrumental in getting good results.

Lastly, **I thank** Bournemouth University for giving me the opportunity that made it possible for the completion of this degree.

Declaration

There are two important assets used for this research. The first is the 3D model of the human hand used to generate the physical model. The second resource used is the 3D hand motion database in a single file encoded in the Autodesk FBX data format. The database consists of 6 motion samples that were used at various points in the training process.

Both assets are from the human model repository of the Computer Graphics and Visualisation Research Group (CGVRG) at Bournemouth University.

Copyright

Knowledge is limited.
But imagination encircles the world.
- Albert Einstein

1

OVERVIEW

1.1 Introduction

Virtual human modelling aims to create computer generated human characters in various applications of computer graphics and animation. It involves the rendering of the anatomical musculature as realistically as possible along with the simulation of muscle functions and control. The virtual humans thus modelled are very useful in a variety of applications like visual effects in films, games, biomechanical analysis, virtual reality and medicine. Depending on end-user application, there are numerous hurdles to be overcome. But from a distinctly biomechanical point-of-view, the challenge is in creating a methodology by which the complex dynamics of muscle simulation can be effectively implemented in real-time applications like computer games.

This thesis proposes a machine learning approach that defines a functional mapping between a motion envelope (surface motion) and muscle activations to produce the same motion. The research presented in this thesis is primarily applicable to physically-based animation of characters in computer games.

Modern games are complex in design. There are multiple sub-systems working in unison to create that sense of immersion. The sub-systems form a framework termed as a *game engine*. The sub-systems are generally in the following categories: User interaction/User Interfaces, Artificial Intelligence (AI), Physics, Networking, Rendering. But sometimes the core systems are sufficiently complex to exist as a standalone pluggable *middle-ware* system. Examples include AI middle-ware and physics engines. With the help of current hardware technology and middle-ware systems, games can make use of simplified or optimized biomechanical skeletal muscle models that mimic anatomical structure and function.

Virtual humans in current computer games are animated using data driven approaches like motion capture and key-frame animation. The data driven methods produce motion through direct manipulation of positions and orientations over time, allowing precise pose control of the characters. These data driven methods are termed as *kinematic*. In contrast, physically-based methods require muscle models or actuators to move the character limbs. But optimization methods are necessary to deal with the inherent redundancy problem in the human muscular system (number of muscles are greater than the number of degrees of freedom). And these optimization methods are classified into two categories: *static* and *dynamic* optimization techniques (Lee et al., 2009). Static optimization methods (inverse dynamics) use kinematic information of motion to calculate muscle forces, while dynamic optimization methods (forward dynamics) accept muscle excitations as input and determine muscle forces to produce skeletal body dynamics (Lee et al., 2009). Both the static and dynamic optimization methods are *deterministic* where future states of the system are precisely defined from the previous states through mathematical relationships.

But in the absence of the required data to form clear mathematical relationships, deterministic approaches fall short of giving problem solutions. The problems associated with deterministic approaches for muscle dynamics and activation when used in games are:

- Proportional Derivative (PD) controllers are limited in scope. PD controllers cannot simulate the complex physiological effects of a multi-muscled anatomical skeletal structure.

- Biomechanical muscle models for generating human motion have sub-models like activation dynamics, contraction dynamics and skeleton dynamics that involve solving complex differential equations at every time step (Lee et al., 2009). Therefore, in a game, since solutions of dynamic equations change with varying input, they have to be re-calculated every frame creating a large computational overhead that may hinder frame rates.

- A majority of dynamic optimization algorithms used in muscle-based human motion simulation use customised simulation environments that often compromise their compatibility with generic physics engine simulators often found in games.

- Muscle activation reference data is difficult to obtain without using expensive medical equipment and invasive procedures.

A skeletal muscle fibre is activated by an action potential travelling along the innervating motor neuron axon and the activation causes the muscle to contract. It is possible to record the action potential during the contraction of the muscle This biomedical signal measure of muscle activations is called an *Electromyogram* (EMG) (Ahsan and Ibrahimy, 2009) (Ohnishi et al., 2007). In order to capture an EMG, sensors are required. The sensors detect the activation signals that innervate the motor neuron. There are two types of commonly used sensors. They are:

- Surface EMG sensors: Surface EMG sensors are non-invasive and take the form of electrodes or conductive elements placed directly on the surface area of the skin from where muscle activity is to be recorded (Ahsan and Ibrahimy, 2009).
- Intramuscular EMG sensors: Intramuscular EMG utilizes a needle and

  ne wire, which are inserted through the skin into the muscle and then the activity is recorded (Ahsan and Ibrahimy, 2009).

In addition to requiring specialised medical equipment and trained personnel, the measurement of surface EMG is dependent on certain factors, as given in Day (2002):

- The timing and intensity of muscle contraction.
- The distance of the electrode from the active muscle area.
- The properties of the overlying tissue like tissue thickness.
- The quality of contact between the electrode and the skin.

Thus, it can be seen that there are a numerous specialised conditions that have to be met before a useful EMG recording can be extracted. The above mentioned complications make it difficult for graphics companies to use EMG on a daily basis. Hence there is the need for a simulated activation controller that can generate activations that a muscle model can use in a timely fashion.

The middle-ware packages used for simulating a wide variety of physical effects in current games are called *physics engines*. Thus, it is logical to use the same physical simulator to implement a simplified muscle model that can be adapted to game characters. A generic physics engine compliant muscle system and a motor control system capable of generating muscle activations in real-time can prove to be invaluable in creating physically-based animation of game characters. Such a system would be able to produce localised damages on game characters by

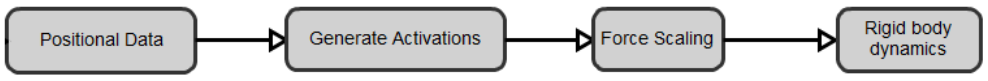


Figure 1.1: A dataflow that combines the properties of static and dynamic optimization methods.

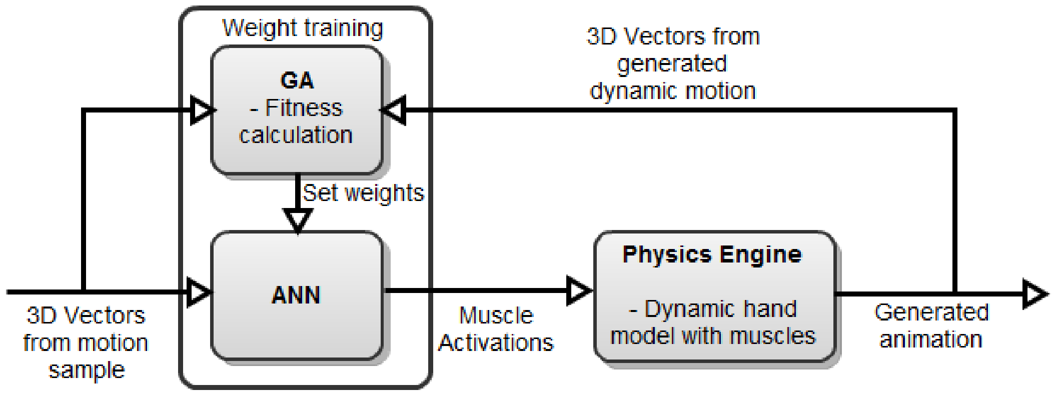


Figure 1.2: An architecture schematic showing the different frameworks and information flow between them.

de-activating the muscles in the proximity of the damaged area creating animation that progresses from natural to handicapped motion. This type of effect is impossible with traditional PD controllers.

A stochastic approach is adopted in this thesis to circumvent the control issues related to the redundancy problem in virtual human animation. The method combines relevant properties of both static and dynamic optimization by hybridizing two popular models in machine learning, namely, artificial neural networks (ANN) and genetic algorithms (GA) (see Figure 1.1).

Coordinated muscle activations are produced through an artificial neural network-based controller that drive a real-time physics engine compatible muscle model (see Figure 1.2). The ANN is based on a time series prediction technique that predicts patterns in a dynamic system. During the training stage, samples of motion capture data is given as input to the ANN. The motion data is formatted into 3D pose vectors which are set as input to the neurons. An evolutionary technique is used to evolve the weights of the ANN due to operational difficulties in using a gradient learning model. The neural network weights are encoded as chromosomes and subjected to simulated evolution using relevant genetic operators. Chromosome fitness is calculated using objective functions that compare the physical motion generated with the input motion. Chromosomes with high fitness are selected using an elitist selection technique. The converged network correctly identifies the time series pattern of the input and produces the corresponding muscle activations. The trained ANN is also capable of generalising to unknown input motions.

A physics engine compatible muscle model is developed that uses a unique *force scaling* method to apply the muscle activations on the rigid body linkages. The developed muscle model is suitably light weight to function effectively in a real-time environment. The light weight nature also allows for complex muscle layouts on multi-linked rigid bodies.

The application of machine learning techniques in muscle dynamics and animation has certain advantages, as given below:

- Using machine learning methods that adopt parallel processing nodes, like the artificial neurons of ANNs, with simple computational functions.

- Machine learning methods using stochastic optimization, like the genetic algorithm, are efficient in exploring vast search spaces that result from a large number of control variables. The muscle activation space is explored indirectly using GAs.

- Generalising ability of machine learning methods allow for re-usability and solving for unknown inputs.

The research given draws inspiration from a range of interdisciplinary fields like neuroscience, robotics, machine learning and computer animation. Hence, the results of the study is applicable both in computer animation and other fields. Below given are a few possible practical applications of the system:

- Converting kinematic animation to dynamic animation enabling characters to react to the game environment in a physically plausible manner.

- Adding a layer of secondary dynamic animation on top of a pre-animated motion.

- Medical uses by simulating neural dysfunction in motor control by adding noise.

- A controller for artificially muscled robotic limbs.

The human hand is used as a test bed for creating gestural animations using the concepts explained in this research. Humans use their hands for a variety of purposes - manipulating tools, communicating, interacting with the surroundings. The human hand is chosen for demonstrating the concepts explained in this research, due to the following reasons:

- The human hand is a complex organ with multiple skeletal segments and a well defined musculotendon structure that makes it ideal for applying the muscle model develped in this research and to test interactive performance in the physics engine environment.
- Problems of balance that arise when simulating bipedal characters do not arise in the simulation of hand movements.
- Virtual characters are ubiquitous in games and they form the medium through which the user interacts with the game world. So animating the characters' appendages procedurally makes the job of the animator easier.

1.2 Motivations and Contributions

Neuroscience states that motor learning and memory have behaviours that are genetically encoded and these behaviours can be modified to different levels based on new experiences (Shadmehr and Wise, 2005). And motion generation and control in the natural world is a complex process involving multiple dependent systems like biological neural networks and organic muscle actuators. So the important question that arose out of these facts was whether it was feasible to encode adaptiveness of muscle actuation with varying inputs as a long term memory. A key inspiration for the processes described in this thesis was the techniques used in Artificial Life (A-Life) that simulated evolutionary processes to find solutions in movement and locomotion. This paved way to devise a method that mimics biological methods to create a multi-part system capable of creating controlled animation of characters with fixed morphologies.

Human beings are adept at imitating a particular movement through observation. This is crucial in skill acquisition tasks like dancing. This demonstrates the ability of the human neuromuscular system to adapt to new movements through training. A biological description of adaptability and neural encoding can be found in section 5.2 in Chapter 5. This ability can also be attributed to the morphological similarity in skeletal, muscular and motor neural structure between different persons. Similarly, the system outlined in this thesis learns to reproduce an observed motion by training the muscles to activate in a coordinated fashion. This is achieved in the absence of knowledge about the underlying neural and muscular processes that originally created the sample motion.

Thus, the contributions of this thesis are as follows:

- An artificial neural network that predicts muscle activations which correspond to the direction of movement in motion data. Dynamic learning is achieved by treating motion as a time series problem.

- A force scaling method that is used by the muscle model in the simulation which allows for timing control in physical animation.
- A simplified linear piece-wise line segment-based muscle model that is optimized for a generic commercial quality real-time game physics engine. From a real-time perspective, this simplified model is very important.
- A training model for the artificial neural network that is based on an evolutionary model using suitable objective functions that selects the fittest controller network from a population of networks. The objective function is also capable of extremely fast retrieval of a pose from a motion sample.

1.3 Publications

Below are the publications that resulted during the course of this research:

- A. Somasekharan, H. N. Charif, J. J. Zhang, Generating Real-time Muscle Activations for Skeletal Hand Motion: An Evolutionary Approach, In the Proceedings of CGI 2012.
- A. Somasekharan, To grab or not to grab: a viable framework for physically-based hand animation in game characters, Future Play '08 Proceedings of the 2008 Conference on Future Play: Research, Play, Share, pp. 254-255
- X. Yang, A. Somasekharan, J. J. Zhang, Curve Skeleton skinning for human and creature characters, Computer Animation and Virtual Worlds 2006, Volume 17 Issue 3-4, July 2006, pp. 281-292.

The author contributed to the third publication by writing the content and also wrote code to implement the concepts in the commercial 3D modelling and animation package, Autodesk Maya. The concepts in the paper are not directly relevant to this thesis, but is useful in creating deformable skin for the physical model of the hand. A quadratic b-spline is used as a guide to create better deformations on the skin mesh, than conventional smooth binding techniques. The method is also suitable for porting to the GPU for real-time applications. This technique is a viable option, in the future, for creating realistic appearances of the skin of the hand during animation.

1.4 Summary of Chapters

This thesis comprises of seven chapters, with three associated appendices and a comprehensive reference section. The first two chapters provide some level of theoretical knowledge of the various related matter touched upon in the course of this research. The rest of the chapters explain the system developed as part of this research. Below are given the summary of the respective chapters.

Chapter 1, titled " OVERVIEW", is a detailed introduction that briefly explains the core idea.

Chapter 2, titled " SURVEY OF PREVIOUS WORK", is an exhaustive literature review, which looks into existing research that is "out there", pertaining to the same and related subject matter, which this thesis deals with. The chapter is broken into different sections depending on functionality. The chapter surveys literature related to physically-based actuators, control and learning in simulated motion, grasp psychology and planning that relating to the decision making processes involved in grasping objects.

Chapter 3, titled " HAND ANATOMY", is a brief and vital look into the anatomical structures of the hand. This is essential in understanding how various hand movements are executed and which muscles in turn cause those movements. Anatomical diagrams are also provided for clarity.

Chapter 4, titled " SYSTEM DESIGN", presents the three frameworks used in this research and the system implementation. The chapter provides a brief background into real-time physics engines and detailed description of the physics engine compliant muscle model and the force scaling method used to simulate muscle function. A brief introduction to the neural controller is also given.

Chapter 5, titled " SIMULATING BIOLOGICAL COMPUTING ELEMENTS IN DYNAMICAL SYSTEMS", presents detailed description of the Artificial Neural Network architecture, activation functions, Genetic Algorithms and objective functions used and how these computational techniques can be combined with a physics engine-based muscle system.

Chapter 6, titled " RESULTS AND ANALYSIS", provides the results of a Proportional Derivative controller-based hand animation and a grasp contact determination test. Following that, the chapter presents the results of the neural controller, with sample frames from the hand animation generated and analyses the how introducing noise and de-activating muscles affect the generated motion. The chapter also provides the fitness-time graph of the convergence.

Chapter 7, titled " CONCLUSIONS AND FUTURE WORK"concludes the thesis by summarising the findings of the research presented and also provides venues for potential future research.

Appendix A, is a descriptive write-up on few of the common numerical methods used in physics engines for simulation.

Appendix B, lists the inertia tensor equations for basic geometry primitives that are found in modern physics engines.

Appendix C, describes the tagged file format designed to export muscle data and mesh information from Autodesk Maya into the physics engine in order to create physical models of the muscles and convex hulls. The tagged file format is also used to store the evolved neural network for later use.

Appendix D, provides screenshots of the Maya Embedded Language (MEL) user interfaces.

Appendix E, is purely exploratory and exists to inform the reader on how the existing system can be improved upon and how possible animator friendly interfaces can be created to fit it into current production pipelines.

Bibliography, lists the references.

2

SURVEY OF PREVIOUS WORK

2.1 Introduction

An understanding of existing research is essential for its own advancement. This chapter lays a groundwork in the field of haptic research, grasp psychology, neural control, physics-based animation, stochastic techniques and controller design. The strategy to create physically-based controlled character animation followed in this thesis, borrows from various areas of the computer graphics research spectrum. The core areas relevant to this research are actuators that form the medium of force generation in physically-based animation, control and control learning strategies for efficiently directing physically-based motion and hand animation that incorporates the results from the former two areas.

Movement generation in biology is an extremely complex process involving conversion of visual goal acquisition into neural excitation levels that causes muscular contraction to physically create the motion. Simulating the same systems through deterministic optimization methods (static or dynamic) is computationally expensive. Using machine learning methods it is possible to create re-usable computational modules that use a totally different paradigm mimicking the neural systems found in the human brain and the central nervous system. Using connectionist machine learning architecture, motor learning patterns are learned through training and the system retains and generalises the learned patterns. The generalised motor patterns are transformed into the required motion via the action of forces transmitted through actuators. PD controllers are a common type of actuators employed by physically-based animation systems mainly due to the simple and linear formulation and the inherent closed-loop nature. In spite of the popularity of PD controllers, the PD controllers have a few notable flaws, namely, lack of a biological basis for the functionality, stability of generated motion, perform tracking of a motion without coordination between controllers and difficulty in tuning the variables as complexity in character increases. Biologically similar muscle actuators employ a more direct form of control through neural excitations and muscle synergies that execute intricate motions. Sensory systems also play an important role for training neural systems to adapt to environmental perturbations (Shadmehr and Wise, 2005).

Section 2.2 examines existing literature on the different types of actuators used in synthesising physically-based character animation. Based on the functionality, this research classifies the actuators into two broad categories: implicit and explicit actuators, which are explained in the sub-sections. Section 2.3 reviews the research on control of movement of characters and incorporating control learning so that characters can automatically execute directed motions. Section 2.4 is an introduction to the psychology of hand grasping and a survey of the research done in creating grasping systems. This is included to inform the reader regarding various aspects of grasp planning and standard animation techniques used to create the grasping animation. The section also touches upon robotics literature as grasp planning is an important problem often encountered in the field. In the end, the chapter summarises the information in the previous sections and provides a rationale for the methodology followed in this thesis.

2.2 Actuated Motion

Motion generated on a segmented body with the aid of active force generators can be termed as *actuated motion* and the structures that perform this task are called *actuators*. Depending on implementation and functionality, this thesis classifies actuators as *implicit* or *explicit*. Contrary to the notion of interactive speeds, games require extremely fast refresh rates. It is perfectly normal for modern games to render scenes at 30 or 60 frames per second (fps) NTSC and 25 or 50 fps PAL (Gregory et al., 2009). But a physics simulation system might require a much higher update rate in order to maintain stability. Normally performance overheads for explicit actuators are higher than implicit actuators. Taking update requirement into account, Table 2.1 lists some actuator simulation methods and corresponding game compatibility.

Table 2.1: Actuator methods

Actuators	**Biological basis**	**Computational costs**	**Game compatibility**
Hill model	Good approximation	Varies with number of muscles	Maybe
PD controllers	No similarity	Very little	Yes
Strand models	Very accurate	Expensive	No
Finite Element Meshes	Accurate	Expensive	No

2.2.1 Implicit Actuators

One of the most common methods of animating a jointed structure physically is through the application of joint torques. The torques can be applied directly via angular springs within the joint. These types of springs, which exist implicitly in the joint, can be termed as *implicit* actuators. Animating linked structures physically not only requires applying forces on the joints, either directly or via external actuators, but also requires tracking or control of the movement. So basically, it breaks down into a *control problem*. In physics-based animation techniques, Proportional Derivative controllers have gained a lot of popularity for effective control (Van de Panne et al., 1994)(Van de Panne and Fiume, 1993)(Gritz and Hahn, 1995). They are a type of feedback controllers, where the output is fed back into the control variable. Thus, a target state is achieved given the current state as input. The output varies with tasks, as does the control variable. In the case of physics-based animation controllers, normally the output is control torques on the joints of articulate structures. PD controllers often require a joint mapping to the linked structure, as well as the specification of target angles for goal.

Often PD controllers are used in conjunction with other data oriented techniques. Van de Panne et al. (1994) utilizes PD controllers for generating periodic motion often found in gaits and locomotion. Van de Panne et al. (1994) combines PDs with *pose control graphs*, which are essentially state machines whose states are associated with specific poses. Van de Panne et al. (1994) applies this combination of PD and pose control graphs to three different kinds of segmented "creatures", namely, the Luxo lamp, a cheetah and a hopper. Each node in the graph is a state and hence comprises a pose, which signifies the "desired internal configuration"(Van de Panne et al., 1994). The connections between the nodes have time intervals associated with it that denotes the transition time to the next state/node occur. Pose control graphs used in Van de Panne et al. (1994) are cyclic for generation of periodic motion and hence "reduces the search space for possible control strategies" (Van de Panne et al., 1994). Due to the segmented nature of the entities, each segment pair (connected by the joint) has a torque for each pose. The PD controllers calculate the required torques,

$$\tau_i = k_p(\theta_d - \theta) - k_d \dot{\theta} \tag{2.1}$$

where θ_d is the desired angle and k_p and k_d are the proportional derivative constants and $\dot{\theta}$ is the angular velocity.

Van de Panne et al. (1994) does not utilize any type of sensory information as a way of optimization for generating the controllers and hence since there is no feedback involved, it becomes an "open loop" controller and relies mostly on the interactions with the environment to reach a steady state. In order to get optimal performance from the pose control graphs, the PD controllers are tuned by varying the spring constants. Though Van de Panne et al. (1994) uses cyclic pose control graphs for generating periodic motion, Van de Panne et al. (1994) maintains that cyclic pose control graphs do not always produce periodic motion. Van de Panne et al. (1994) goes on to demonstrate the claim that non-linear dynamical systems are susceptible to bifurcations and chaotic motions, by taking the Luxo lamp creature and using the transition time between two states in a pose control graph as a bifurcation parameter. By varying the time of transition, new "styles" appear within the periodic motion, such as limping. Fattal and Lischinski (2006) on the other hand uses feedback controllers for full-body motion of articulated characters. For the mechanism of motion, it is similar to Van de Panne et al. (1994) in that each articulate joint has a PD controller attached to it and for the control aspect of the motion, animators specify target poses for the character. The PD controller automatically calculates the joint torques necessary to reach the target poses. Fattal and Lischinski (2006) also uses the feedback controllers to synthesize simple behaviours by constructing what is termed as autonomous objective controllers. In order to facilitate the animator with the ability of explicitly setting kinematic constraints (for some actions like jumping to a certain height) Fattal and Lischinski (2006) uses a shooting strategy. The shooting strategy relies on the application of minute quantities of stabilizing torques to match the kinematic constraints without compromising on the physical realism. Re-using the external torques used for balancing the character is not possible since it would violate the conservation of angular momentum and thus create physically implausible motion. The approach given in Fattal and Lischinski (2006) was first utilized in Popovi et al. (2000) where the animator could kinematically modify physical simulations providing a way for animators to interact with a simulation. Popovi et al. (2000) maintains the motion of the body in the normal physical property variables like positions and velocities, but when the animator performs kinematic modifications, the system does a fast differential update to get the new physical parameters that comply with the updated motion. The fast differential update allows real-time interaction. Simulation variables (such as changing collision normals with a different polygonal facet) can cause motion discontinuities, which are taken care of by a local search to retrieve, that which closely matches the given modifications.

Neff and Fiume (2002) takes a more intuitive and biomechanical approach, though still using a modified PD control. Neff and Fiume (2002) bases its approach on the fact that in the human body smooth and controlled motion arises out of excitation of agonist and antagonist muscles (as defined in Hogan (1984)). The model described in Neff and Fiume (2002) follows the equilibrium point hypothesis, which is popular among biomechanical researchers. Anatol Feldman put forward this theory in 1966 and according to it; the agonist and antagonist muscles both generate torques around the joint with the equilibrium position of the torques defining the position of the limbs. There is also an opposing theory called the impulse timing model which states that motor neurons generate timed impulses to control joint movement directly (Neff and Fiume, 2002). Traditional PD control neglects the effects of external forces like gravity. According to Neff and Fiume (2002), the equilibrium point is the angle at which the proportional term in the PD equation equals the torque generated by gravity. Antagonistic control in Neff and Fiume (2002) is implemented using two angular springs on a joint acting in the opposite directions along with a damper. The modified control equation followed in Neff and Fiume (2002) is :

$$\tau = k_L(\theta_L - \theta) + k_H(\theta_H - \theta) - k_d\,\dot{\theta} \tag{2.2}$$

where τ is the torque generated, k_L and k_H are the spring gains, H and L denote the high and low limits. $\dot{\theta}$ is the current angular velocity.

The sum of the two spring gains provides the tension of the joint. Representing the tension as a summation of the gain variables allows the use of simple transition functions to vary the shape (how the motion varies over time denoting the speed of the motion) of the motion purely through tension changes. This is important because timing is crucial in animation of any form to create physical realism. But unlike Fattal and Lischinski (2006) which generates autonomous behaviours, Neff and Fiume (2002) details a technique by which the animator can animate a character using a physics-based control system. The stiffness of a joint is an indication of how quickly or slowly the joint achieves a desired target angle and increasing the gain to overcome external forces creates stiff motion. Weinstein et al. (2008) modifies the PD controller using impulses and velocities instead of forces and accelerations in order to minimize drift which otherwise occurs. Weinstein et al. (2008) also takes into consideration global forces like gravity so that the controller can overcome it and reach the desired target. Weinstein et al. (2008) terms the technique described as a “inverse dynamics tracking of PD smoothed input motion”. In order to achieve tracking, Weinstein et al. (2008) uses a second order system, which is physically compatible with PD control. Inverse dynamics is used in Weinstein et al. (2008) to alleviate the problem of influence of torques from other joints in a multi-jointed articulate system. So Weinstein et al. (2008) practices post stabilization technique for the entire articulate body instead of iterating over all joints sequentially. Weinstein et al. (2008) also applies the techniques to line segment muscles, in which the operational method is different from PD control. Line segment muscles are line-of action based actuators (or linear springs). When simulating anatomical muscle effects using line segment muscles, anatomical arrangements of muscles need to be taken into consideration. Weinstein et al. (2008) uses a Hill-type model and applies the post-stabilization method to include muscle activations. Coupled with different control structures, like in Van de Panne et al. (1994)Fattal and Lischinski (2006)Van de Panne and Fiume (1993), these proportional derivative mechanisms become a powerful technique to generate physically compliant motions. Gritz and Hahn (1995) introduces a control mechanism for articulated figure motion. Though the paper mainly deals with the evolutionary aspect of generating coordinated motion, the underlying mechanism implemented is PD control.

Another example of effective PD-based animation is detailed in Laszlo et al. (2000), where planar articulate figures, like a two dimensional Luxo Jr lamp with two actuated joints, are controlled through traditional human-computer interfaces like the mouse. Laszlo et al. (2000) defines a linear mapping between the two dimensional mouse coordinates and the desired angles of the two joints in the lamp figure. So the movement of the mouse creates time related motion in the figure. Laszlo et al. (2000) also makes use of check points inserted at specific points in the motion. These check points facilitate reversal or rewind of the motion to rectify mistakes. Laszlo et al. (2000) maintains that mapping the mouse coordinates to the joints create a *continuous control* while key-stroke based animations (used for articulate figures of higher complexity like a planar cat and bipedal figure) gives rise to *discrete control*. It is also stated that continuous control is often effective if the mapping of joints is made with continuously varying parameters normally coinciding with the number of degrees of freedom (DOF) in the articulate figure.

Artificial Life (A-Life) research investigates the theory of evolution and adaptiveness of organisms in their environment and simulates the processes on computers. Implicit application of torques in the joints of articulate structures is one of the most common techniques of physics-based character animation, and is used by Karl Sims A-Life work. Sims' pioneering paper (Sims, 1994) on evolving organisms within a simulated physical world proved that adaptive animation was possible through a convergence of machine learning algorithms and physics simulation. While Sims (1994) refrains from a PD implementation, it uses evolved control structures to modulate joint torques through effectors. The force output of effectors are limited proportionally by the cross sectional area of the segments they join (Sims, 1994). Unlike PD where there is a constant feedback of the current deviation from the desired target, the effectors in Sims (1994) are modulated and adjusted only through each generation of evolution. So during one specific period of simulation, effectors produce a specific envelope or shape of motion.

2.2.2 Explicit Actuators

Articulate body animation can also be done through external application of controlled forces on the linkages. These types of actuators can be termed as *explicit actuators*. Physically-based anatomical methods of animation, often use this type of actuation. The two goals of muscle simulation can be termed as morphological or surface behaviour modelling for enhancing an existing animation and a physiological goal where biomechanical parameters of muscle function is studied to generate animation (Lee et al., 2009). A morphological behaviour attempts to model the deformations of the muscle volume in accordance with the skeletal movements (purely based on animation with no biological basis), like bulging and stretching. These internal muscle deformations are key references to simulating the deformations occurring on the overlying skin. Modelling the physiological behaviour requires the understanding of the biomechanics of muscle contractions and force generation related to muscle length. This is a true anatomical approach whereby muscle activations cause skeletal motion and hence is classified under explicit actuators.

The most popular biomechanical model of a muscle is the Hill model. A. V. Hill in 1938 measured shortening velocities of stimulated frog muscles and defined force-length relationship equations (Lee et al., 2009)(Holmes, 2006). Since Hill's relationship equations have been observed in most types of biological muscles, it has become a standard simulation method.

When a muscle contracts under activation, active force is generated. The total force is the sum of both active and passive forces (see Figure 2.1). The velocity with which the muscle contract is also an indication of the amount of force it produces. This relationship in muscle contraction is also shown in Figure 2.1. This property of the muscle is attributed to the sliding filament theory and cross-bridges (IvyRose, 2010). Hill created a mechanical model that exhibits

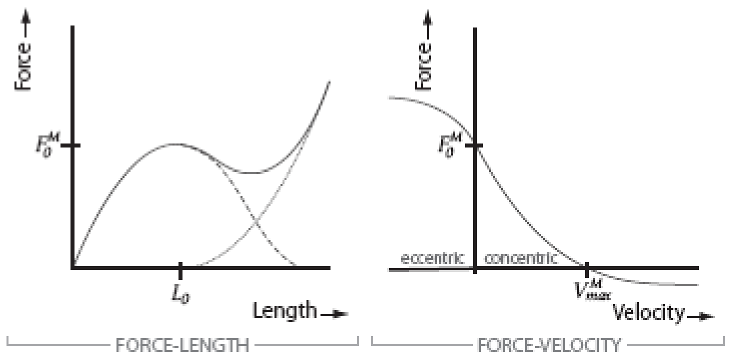


Figure 2.1: Force-Length and Force-Velocity Relationship graphs

this behaviour, having three major components: a series element, parallel element and the contractile element. The series element replicates the elastic nature of tendons and also the elasticity prevalent in the sarcomere. The passive elasticity of the muscle is simulated using the parallel element and finally the contractile element takes into account the muscle length and time dependent neural signal to calculate the generation of active force (Lee et al., 2009). The Hill model, though biomechanically accurate, is not a convenient form for replicating any musculotendon unit. So the Hill model was modified in Zajac (1989) in order to achieve that goal. Zajac (1989) developed a model that is dimensionless, where the series element is combined with the tendon modeling and thus removed and pennation effects are intrinsically modelled. It models muscle fibres directly. The Hill model describes macroscopic behaviour of skeletal muscles. Another model called the Huxley model, describes microscopic contractile elements, the sliding filaments and the cross-bridges (Huxley, 1957). Cross-bridges are modelled using Hookean springs and the total force exerted by the muscle is computed by taking the sum of all the forces of the connected bridges (Lee et al., 2009).

Muscle simulation can be classified into three groups depending on the nature of the simulation. They are: Geometry-based, Physically-based and Data driven (Lee et al., 2009). Geometry-based approaches, as the name suggests, simulate muscle animation and related deformation effects, rather than true biomechanical behaviour. Physically-based approaches, in order to bring greater realism and accuracy, rely on the physical principles of muscle contraction and deformation. Data driven approaches are relatively new and involves extraction of skin deformation directly from the skin surface using markers and using mathematical methods create new deformation effects for new poses (Lee et al., 2009). Thalmann et al. (1996) is an example of a geometry-based method of creating realistic deformations during animation. An interactive system for human character modelling called “Body Builder” is created. It implements implicit surface technique using *metaballs* and ellipsoids to represent the bones, muscles and fat tissue (Thalmann et al., 1996). The implicit primitives are defined in the local joint coordinate system of the underlying skeleton system. Using the implicit system of modelling allows for an additive methodology in modelling for creating realistic muscles. Also, during animation, the use of deformable implicit primitives provides for a convenient and visually accurate deformation. The underlying skeletal joint acts as a reference joint that adjusts a variable that drives the centre, orientation and shape of the implicit surface attached (Thalmann et al., 1996). There are various other space deformation methods used for recreating the effects of muscle deformations, like Free Form Deformations (FFD), Dirichlet Free Form Deformation (DFFD), Skeleton-Subspace Deformation (SSD) and Pose Space Deformation (PSD) (Lee et al., 2009). FFD makes use of a lattice to deform the shape of the muscle. The shape of the lattice in turn is deformed when the underlying skeleton moves. Parametric solids or composite tri-cubic bezier-based hyperpatches form the FFD basis for deformation (Chadwick et al., 1989). The muscles, in this case, are represented by pairs of FFDs. SSD and its derivative, PSD are defined in (Lewis et al., 2000). SSD is a skeletal-based deformation with obvious drawbacks like lack of interpolated blending. The movement of the skeleton underneath the skin deforms the skin surface.

PSD improves the SSD algorithm through iterative-layered refinement (Lewis et al., 2000). PSD uses abstract manipulators or data points that control whole or part of the deformation. These data points are set by the artists and then for a particular pose they sculpt deformations and also a falloff that defines how the set points change when the pose changes (Lewis et al., 2000). Parametric and cubic surfaces are also used in the geometric simulation of muscles. Komatsu (1988) uses Bezier surfaces to model human skin over skeletal structures. Implicit surfaces, blending graph, weighted blending with proximity, parametric ellipsoids etc are other techniques used by researchers to model muscle deformations. Physically-based approaches target two problems, namely, representing and determination of muscle contraction forces and computing the subsequent deformation of the muscle geometry (Lee et al., 2009). Some of the computational techniques used for this purpose are mass-spring systems, Finite Element Method (FEM), Finite Volume Method (FVM) (Lee et al., 2009). Since accurate geometric and physical musculoskeletal models are beneficial in a wide area of research, researchers are always looking for methods to create them using different techniques.

A promising method is *volumetric methods*. To extract precise volumetric data of muscles, one needs a good source and the visible human data set is one such source. Teran et al. (2005) uses a segmented version of the visible human data set to create muscle, tendon and skeleton geometry. The segmented volumetric data is not perfect and hence consists of disparities. A level set-based smoothing method is used to repair the imperfect data (Teran et al., 2005). The volumetric data is converted to the respective meshes using implicit surface meshing. After the geometry creation process, additional data like temporally varying muscle fibre direction vectors and skeleton structure to create skeletal motion (Teran et al., 2005). The FVM is used to integrate the equations of motion (force computation) with the generated musculoskeletal model. Mass-spring systems are a common way of representing muscle deformations. Chadwick et al. (1989) connects the FFD lattice control points to a mass-spring system to give visco-elastic dynamics to the deformation. Mass-spring system combined with action lines and force fields for joint wrapping is implemented in (Aubel and Thalmann, 2001). The insertion and origin nodes are constrained on the skeletal structure and driven by the skeletal motion. The intermediate nodes are calculated using an elastic relaxation method (Aubel and Thalmann, 2001).

Data-driven approaches ignore the anatomical mechanisms involved in muscle shape determination and instead model the surface of the skin directly. The skin surface is modelled for specific poses using data acquisition techniques like range scanning device and markers in a motion capture system. Generation of new skin surfaces for differing poses is done by interpolation (Lee et al., 2009). Allen et al. (2003) creates a shape-fitting method to fit high resolution meshes to range scan data with sparse markers. Allen et al. (2003) explores the space of human shapes using morphing. During the scanning process, sparse white markers are placed on the body at specified landmark points (where the bone pushes through the skin). Allen et al. (2003) creates a correspondence between surfaces having the same global structure but with sufficient local variability. Capturing and animating dynamic deformation of the human skin surface is explored in Park and Hodgins (2006). Optical motion capture using approximately 350 markers and 12 cameras is used to capture the motion and then the deformation of the skin is computed from the fiducial data. Computation is done with the help of quadratic transformation and radial basis functions (Park and Hodgins, 2006). The marker data is cleaned and damaged marker data is reconstructed using a local reference framing defined at each marker and also using spatial information relationship between the markers (Park and Hodgins, 2006). Treating the marker data as a scatter data interpolation problem and deforming the vertices, performs the skin animation. Tsang et al. (2005) presents an inverse dynamic musculotendon model of the hand and forearm. It also integrates a forward simulation model, which takes muscle activation values as inputs and outputs rotation values and angular velocities for the joints. Thus Tsang et al. (2005) considers forward simulation as a function that maps muscle activation values to angles and angular velocities.

The inverse dynamics computation takes as input joint rotations or finger configurations. The Hill's three-element model is used for the simulation with direction computed along the line of action. The tendons are represented by piece-wise linear segments (Tsang et al., 2005).

Machine learning techniques are useful in teaching a system highly non-linear functions and where the problem space is vast. Lee and Terzopoulos (2006) models a neuromuscular controller capable of animating the human neck. The muscle model is anatomically relevant and comprises of linear actuators. The anatomical muscle activations are emulated by an off-line trained neural network. The linear muscle actuator is based on the Hill model with parallel and contractile elements.

A new modelling primitive called *strand* is implemented in Sueda et al. (2008) making it possible to model the complex interactions of muscle-tendon and bones. Levin et al. (2008) describes a method using diffusion tensor data to model the strands from MRI scan data. Energy minimizing curves are used to fit the muscle fibre field. The strand primitive is a biomechanical element that can incorporate complex routing constraints (Sueda et al., 2008). It uses a cubic B-Spline path curve with dynamic control points. For contacts with the skeletal structure, the system uses pre-calculated potential contact points, which are, updated every time-step. The strand primitive has three types of constraints that can be attached to it. They are: Fixed constraints (which are locally stationary and is used to create the origin and insertion points of muscles), Surface constraints (constrained to be on the surface of the bones) and sliding constraints (constraints that allow the strand to move along its axis). Formulation of these constraints allows the primitive to model the interactions between muscles and bones accurately (Sueda et al., 2008). Sueda et al. (2008) also implements an algorithmic controller that computes activation levels given a target motion. Even though the strands are capable of modelling accurate tendon routing dynamics and produce high quality biomechanical motion, it is impractical to use in a game character. This is due to the high cost of computing motion equations of a dynamic b-spline and constraints and the inability to develop a customised strand primitive using a generic physics engine.

Another popular method of creating musculotendon units that wrap around joints and skeletal structure is known as the *Obstacle Set Method*. In biomechanical simulation, defining muscle paths is very important, as the arrangement of muscles is a key element in establishing the direction and behaviour of motion generated. In the obstacle set method, muscle path is represented by a series of line segments (Brian and Marcus, 2000). These line segments are connected using *via points*. At points where the muscle path curves around a joint, a rigid body

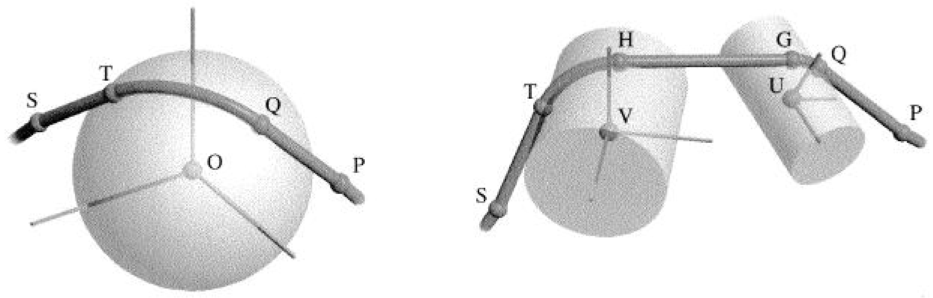


Figure 2.2: Obstacle set method. Image courtesy Brian and Marcus (2000)

primitive (spheres, cylinders) is used as an obstacle upon which the via points are laid (see Figure 2.2) (Brian and Marcus, 2000). The via points can have two reference frames depending on which underlying structure they are attached to. Bone reference frames are attached to bones while obstacle reference frames are attached to obstacles (Brian and Marcus, 2000). The via points lying on the obstacle surface are constrained to move on its surface, while the via points on the bones are fixed. A problem with the obstacle set method is the necessity of the collision primitives to facilitate joint wrapping. This is an additional overhead in the context of a game character.

To compute muscle functions is challenging because of the high complexity and number of constraint equations and also due to the high redundancy of the muscle system (Lee et al., 2009). The redundancy exists because the number of muscles is greater than the number of DoFs available for skeletal motion. Thus the problem is highly under-determined. Static (inverse dynamic) and dynamic (forward dynamics) optimization techniques are commonly used to solve this problem. But these often require large amounts of differential equations as the number of participating rigid bodies and DoFs increase. Optimization approaches like in Thelen et al. (2003) implements state feedback and static optimization techniques to reduce this increase in computations.

2.3 Control and Learning

In physically-based animation, control is an essential element by which the animator can directly or indirectly affect motion in order to achieve a specific physical action. Target oriented motion is impossible without a finite control guiding and activating the actuators at precise moments. Graphical interface-based control systems and motion tracking are examples of two different ways by which desired animation is created or applied on virtual characters (Parent, 2007). The former is the most common type found in standard 3D animation packages and includes the normal key frame controls, kinematic "handlers" (like IK tools), spline-based animation tools etc. The latter is a more complicated form of control of the body where the character executes the desired animation by following a pre-generated motion sample, usually motion capture. A prime example of a controlled dynamical system would be an articulate rigid body rag doll[1] tracking the motion of a running humanoid, through the application of forces via actuators. Exacting precise control in physics-based characters is necessary to create motion that is comparable to kinematic methods. Due to such control issues, physically-based character animation was seldom used to drive performance of characters. In games, physically-based character animation is normally used for portraying character death using a physically-based puppet called rag doll. There are two notable pieces of software that use biomechanical principles to generate motion, with one used in the entertainment industry and the other used in analysing motion, *Endorphin* and *OpenSim*.

Natural Motion's Endorphin software applies machine learning techniques to create physically-based realistic full body motion of characters using biomechanical muscles and neural controllers modelled after the central pattern generators found in the human CNS. Endorphin terms the rag dolls as active rag dolls as opposed to passive rag dolls often found in games. Though the exact methods implemented in Endorphin are not disclosed, the original research the technology is based,relies on recurrent neural network architectures for the controller and PD controllers as actuators for the physically-based characters (Reil and Husbands, 2002). Endorphin evolves motion behaviours based on customised objective functions and these evolved behaviours are provided as a list for the animator to choose and apply to the characters. The behaviours can be modified in-software using a variety of parameters that control the simulation. Endorphin also allows to integrate motion captured animation to blend between simulations. From observed workings of the software, the muscular system seems like a simplified model that does not model the muscle dynamics in its entirety (Motion, 2011a).

OpenSim is an open source software using deterministic methods that helps users to create and analyse the biomechanics of human movement (OpenSim, 1990). It fits recorded marker data to musculoskeletal model computes inverse dynamics and also muscle-based forward dynamics simulations using gait data. It allows for model scaling to fit any subject and uses linear muscles based on the Hill model and joint wrapping using via points.

For understanding controlled motion generation on anatomical models, re-

[1]A rag doll is a physical simulation of a humanoid body consisting of constrained rigid bodies. It is normally used to simulate an unconscious or dead character in a game.

searchers look into neuromechanics. "Neuromechanics seeks to understand how muscles, sense organs, motor pattern generators, and brain interact to produce coordinated movement, not only in complex terrain but also when confronted with unexpected perturbations." (Nishikawa et al., 2007). The mechanical behaviour of muscles is a direct result of neural activity produced by conscious or unconscious thought. But the pattern of neural activity is still relatively unknown barring some educated guesses. The process of taking visual cues or conscious stimuli and converting it to mechanical motion consists of a series of transformations of the information available - from the brain to the central nervous system (spinal cord) to the muscles and then back again during feedback (Nishikawa et al., 2007). This control problem is represented in the form of a *transform function*, which takes place at each stage of the information transference (Nishikawa et al., 2007). The activation pattern of skeletal muscles contribute to the force output of the muscle, single stimulus causes twitches in the muscle, while multiple low frequency stimuli causes greater force output and contractions like tetanic contraction is caused by high frequency stimuli and thereby have the maximum force output (Nishikawa et al., 2007). Muscles serve a variety of functions other than cause movement (Nishikawa et al., 2007). Muscles are instrumental in stabilizing joint motion and storing elastic energy in the fibres. A single nerve signal can cause different mechanical output in different muscles as well as different muscle segments in a single fascicle (Nishikawa et al., 2007). Thus muscles act as devices that convert a neural signal into mechanical output.

And yet, according to Nishikawa et al. (2007), there are functions of skeletal muscle which are still unclear, some of them being, inter-filament spacing, muscle architecture and protein isoforms. Muscles exhibit self-stabilizing behaviour also, the self-stabilizing arising from the inverse force velocity relationship exhibited by striated muscles (Nishikawa et al., 2007). The speed of a movement is not merely dependent on how fast a muscle contracts (in fact, contraction contribution of a muscle to movement speed is less when compared to elastic recoil), but depends on the muscles, tendons and skeletal elements, together forming a mass-spring system. Sensory feedback in the muscles from the various feedback sensors in the same, contribute to the variances in muscle activation and force control, which in turn helps in stability and coordinated motion (Nishikawa et al., 2007) (Eccles et al., 1957). Feedback comes from various sensory inputs like proprioception, vision, auditory etc. There are two principles of neuromechanics that affects the control of movement, being, structural properties of the sensory and neural networks influence the timing and frequency of muscle activations and secondly, that these properties themselves introduce timing delays (Nishikawa et al., 2007). An efficient controller must encapsulate these principles in order to create coordinated motion. According to Nishikawa et al. (2007), the factors that play a role in the generation of coordinated motion are: command delegation and interaction in the central nervous system, biomechanics of the human body, the environment and sensory feedback. Nishikawa et al. (2007) formulates three key points to emphasise the importance of timing in neural stimulation. The first being that delays occur either due to neural processing or due to the properties of the mechanical system. Second, timing delays cause a reactive resonant like behaviour. Thirdly, the dynamic nature of a system is fully understood only when the time delays in the signals are considered. Nishikawa et al. (2007) further theorizes that deriving a control model from these observations, especially for a system that has many-input-many-output parts, requires a full knowledge of the delays in the neural processing in the mapping between these parts. Nishikawa et al. (2007) also explores the neuromechanics problem in the context of locomotion using Central Patterns Generators (CPG) for periodic motion, use of Genetic Algorithms to evolve timing of neural activation patterns.

Just as Karl Sims in Sims (1994), showed that the physical environment plays a role in how the morphology and locomotion pattern of the creature evolves, Nishikawa et al. (2007) also delves into the functional dependencies of an organism on its ecological morphology and tries to throw light on determining musculoskeletal performance in its natural environment. Temporal information plays an important role in ecological morphology. It has been found that similarity in the design of an organism can produce differing performance levels and at the same time, different morphological design produces comparable performance levels when suitably controlled (Enoka, 2008). This unique nature in musculoskeletal performance is attributed to a tight interaction between the neural system, the mechanical system of the organism and the environment or the ecology. Mechanical attributes consist of shape, structure and dimensions. Motor control requires highly complex muscle activations and studies suggest that the central nervous system uses muscle synergies to create various types of movements (Ting and McKay, 2007). Ting and McKay (2007) defines muscle synergy as "a vector specifying a pattern of relative levels of muscle activation". Ting and McKay (2007) proposes that the central nervous system executes task level commands through the use of muscle synergies. The muscle synergies are the solution to convert the commands into a complex set of spatio-temporal activation patterns (Ting and McKay, 2007). Perturbations happen frequently causing instability during a task execution (like balancing or locomotion or reaching). Using one or more combination of synergies, the central nervous system is able to counteract the perturbations. The combined effect of the synergies defines the complex muscle activation patterns (Ting and McKay, 2007). (Ting and McKay, 2007) proposes that muscle synergy is an emergent consequence of the interactions between the nervous and musculoskeletal systems and that the existence of synergies helps the nervous system to encode task-level variables and better adaptation to environmental changes and perturbations. Bernstein formulated the degrees of freedom problem in motor control (Macpherson, 1991). The musculoskeletal system is comprised of many linked segments, joints and muscle actuators. The task of moving an arm to a target point offers many solutions, with a variable amount of parameters (DoFs). There is no unique solution. So how does the nervous system produce the required muscle activation patterns to achieve the task? Bernstein proposed that the nervous system pooled the parameters into functional units. Combining individual muscles in groups causes a reduction in the number of parameters and the central nervous system can create various synergestic units for different tasks (Macpherson, 1991).

For posture control and locomotion, neural-based foundations have been theorized (Deliagina et al., 2008). It is termed as a "closed-loop" control system (Deliagina et al., 2008). Maintaining balance or up-right posture is essential for a variety of physical activities and forms a basic and vital motor function in the human body (Deliagina et al., 2008). In order to achieve this with the ability to change postural functions, the central nervous system distributes the functions in different parts within itself. Locomotion using limbs is well-worn research bed in robotics and simulation. Artificial neural networks have been used to generate muscle activation patterns for locomotion (Prentice et al., 1998). In order to generate the correct timing of activation, simple sine and cosine wave forms were used to represent the duration of a gait and then the waves were moulded into muscle activation patterns for limb-based locomotion using a back-propagating neural network (Prentice et al., 1998). The neural network used in this research is time independent (unlike spiking neural network which are time dependent), instead using secondary functions (sine and cosine) to modulate the time. The shape or envelope of the function is modulated using the neural network (Prentice et al., 1998). Gradient learning is performed using actual muscle activation recordings (EMG).

For dynamical systems without control, muscle actuators would produce random motions. The control problem in dynamic characters often necessitates principles from control theory, like the research detailed in Section 2.2.1. Bringing about autonomous behaviour in virtual characters requires the implementation of procedural techniques or machine learning techniques. Procedural techniques often use higher order mathematical principles to direct motion into the desired trajectories. Some examples in procedural techniques are (Van de Panne et al., 1994)(Witkin and Kass, 1988)(Sifakis et al., 2005). The space-time constraint system described in Witkin and Kass (1988) is a form of directed character animation control that preserves the principles of animation like squash and stretch and anticipation. The technique is used to animate, again, a Luxo. Jr lamp. The objective of the technique is to treat the character animation problem as a constrained optimization problem, the constraints being the start and end positions of the motion and the time within which to execute the motion.

Many procedural control techniques in physics-based animation employs some form of PD control as described in Section 2.2.1. But adaptive learning is performed by tuning the control parameters, often through machine learning techniques. Faloutsos et al. (2001) synthesizes controllers for physics-based character animation, which are highly procedural with relationships and constraints involving several parameters called "pre-conditions". The individual controllers are generated through manual intervention and later composed into single complex controllers. This particular design process is automated as well, using machine-learning techniques. The technique used in this case is *support vector machine* (SVM), which is a linear classifier. The SVM used utilizes kernel functions to map non-linear support vectors to higher dimensional spaces in order to linearly separate them. The SVM is trained to determine the controller pre-conditions and the training is performed off-line. The transitions between the controllers are achieved through state machines. Due to the high non-linearity and complexity, design of controllers for multi-limbed characters is often very difficult and for this reason, self-evolving controllers are used.

Genetic programming is used to evolve control expressions for articulated characters. Gritz and Hahn (1995) implements genetic programming to evolve expression for a dynamic Luxo. Jr lamp. PD control generates torque at the joints of the Luxo lamp. The torques produced is a resultant of LISP expressions that the genetic program outputs. Constraints are imposed as optimizing or objective functions for the genetic programming algorithm. Evolutionary methods are commonly used in A-Life research to automatically generate means of locomotion for virtual entities. Sims evolved both the control system as well as the morphology of the virtual entities using genetic algorithms (Sims, 1994). Other researchers have either reproduced or modified Sims' work and most of them use PD controllers as actuators (Lassabe et al., 2007)(Ruebsamen, 2002)(Miconi and Channon, 2006)(Smith, 1998a). But from a practical production pipeline point of view, A-Life on its own has little value. There are two reasons:

- Random morphology - Most A-Life research focuses on evolving the structure of characters and also on evolution of locomotion suited for that particular structure. The evolved structure might have no correlation with any plausible character morphology. In real life production, character morphology is fixed, based on concept designs.

- Specificity of objective function - The objective functions that guide the evolution in A-Life are often simple target-based (distance travelled) and used for locomotion generation. These functions do not specify the shape of the motion. On the contrary in character animation, motion styles are required.

A-Life entities exist in isolated physical (or non-physical) environments interacting with each other and evolving behaviours based on simplistic objective functions. Evolving motion behaviours that emulate human motion require more complex objective functions. But the A-Life techniques proved inspirational in evolving the system developed in this thesis.

Grasp animation is an area of research, which is gaining interest. It provides avenues for automatic control both in planning and animation. Recent research has indicated that though grasping is a generalized movement, there are individual coordinated controllers that perform the reaching and curling of fingers around the object (Oztop et al., 2004)(Jeannerod et al., 1998). In fact the development and coordination of fingertip forces for manipulation of objects, arise from independent neural networks controlling each of the digit involved in the task (Burstedt et al., 1997). The neural system coordinating the manipulation was found to be independent of the number of manipulators used (two handed) by a single subject or two subjects. This suggests that the neural mechanisms and the effectors can be anatomically independent for controlled manipulative grasping tasks (Burstedt et al., 1997). There are two main areas that the research in Burstedt et al. (1997) helps clarify regarding independent neural control during manipulation of objects. They are: digit specific control of the normal-tangential force ratios, temporal coordination (Burstedt et al., 1997). In the first case, experimental subjects coordinated the fingertip forces compensating for changes in local friction. Testing with various grasp configurations also yielded the same result, with evident adjustments of the normal-tangential fingertip force ratios to balance frictional slips. A decrease in the rate of increase of finger tip tangential forces was also observed prior to lifting tasks indicating anticipatory systems, which again operate via independent neural networks separately controlling the digits (Burstedt et al., 1997). Even though lifting tasks can have independent neural mechanisms controlling and modulating force development, higher-level control mechanisms are required to coordinate the digits for various other tasks. This is where temporal coordination becomes important (Burstedt et al., 1997). In this case, coordination showed an increase in the single subject case than the two subject case, mainly because neural systems function better with self-contained sensory feedbacks than when the neural systems belong to completely independent subjects and have to rely mainly on visual cues (Burstedt et al., 1997). Grasping with hands and manipulation undergoes training from infancy to adulthood. Oztop et al. (2004) looks into how grasping is a learned process in infancy and goes deeper into the behaviour of grasping. In that process, a neural network-based learning model is investigated and implemented.

The main objective of the research in Oztop et al. (2004) is exploring the effects of the environment and grasp constraints in the development of grasping skill in infants. Two main points are suggested as being important for an infant to learn an efficient grasp: the ability of the infants to sense the effects of their motor action and interpreting feedback sensations to adjust their grasping parameters (Oztop et al., 2004). According to (Oztop et al., 2004), basic motor requirements for reach to grasp are firmly established quite early in the developmental stages. Sensorimotor skills rapidly develop as motor learning takes place. Infants have a very basic and crudely coordinated grasping skills at birth, which later on develops into reaching and grasping by 4-5 months. By 9 months, their grasping strategy is sufficiently advanced for reaching and grasping and by the time the baby is a year old, manipulatory and precision grasping skills comparable to adults are developed (Oztop et al., 2004). The model developed, named Infant Learning to Grasp Model (ILGM), learns to grasp by interacting with the environment, adjusts parameters based on feedback and selectively modifies neural parameters to fine tune grasp action and patterns (Oztop et al., 2004).

Robotics is a field where learning algorithms and mechanical control of articulate rigid bodies, grasp planning and execution are priority research goals. Inverse kinematics (IK) is a common problem and roboticists are looking for better and faster ways to solve it. Oyama et al. (2005) implements modular neural networks to learn IK. Using neural networks for learning IK has been done previously (Kuperstein, 1988)(Jordan, 1988). IK being a multi-valued discontinuous function, is perfect for a modular neural network rather than conventional neural network design using single valued function. By breaking down the IK function into a finite number of IK function groups and then allow individual neural network "experts" to find a solution for each of the groups. By approximating a region of the IK function with each module, a viable solution to the function as a whole is calculated (Oyama et al., 2005).

The multi-fingered gripper is a very common robotic body part. Robots require grippers to perform many of the dexterous tasks. And the key problem faced by roboticists is force optimization (Xia et al., 2004). The force transferred by the robotic gripper on the object being grasped has to carefully modulated.

The object might collapse under excessive force or slip from the gripper if the force applied is too little. Therefore, optimal control of the forces is essential. Xia et al. (2004) investigates this particular problem and uses recurrent neural networks to learn the force optimization.

In grasp planning, if a neural model can encapsulate a better object feature set and affordances in grasp, then the generalizing ability of the network for a wider range of features is enhanced (Molina-Vilaplana et al., 2007). Molina-Vilaplana et al. (2007) proposed a multi-part neural network that learned grasping tasks. The model was designed as a controller for robotic grippers. For accuracy comparison, the neural activity in the different parts of the network is compared to biological neurons that are involved in visual guidance for grasping (Molina-Vilaplana et al., 2007).

Advanced prosthetics is robotics in part (replacing lost limbs with robotic limbs), but retaining biological control rather than being completely autonomous. One of the shared goals of robotics and prosthetics is to create a manipulator as dexterous as human hands. Electronically enhanced devices like robots or even computer graphic software receives input directly from the user's brain by extracting or sensing biomedical signals such as *Electromyogram* (EMG), *Electroencephalogram* (EEG), or *Electrooculogram* (EOG) (Ahsan and Ibrahimy, 2009). Most neural prosthetic devices use EMG as a control signal. In the human body, muscles are the actuators, which translate electrical impulses from the brain into motion. As described in IvyRose (2010), a skeletal muscle fibre is activated by an action potential travelling along the innervating motor neuron axon and the activation causes the muscle to contract. By placing sensors to pick up the activation signals of muscles, the prosthetic device is able to coordinate the activation of the mechanical motors through complex pattern classification techniques (self-organizing maps, genetic algorithms, fuzzy logic, wavelet analysis). But a majority of the classification is performed using classifiers that are based on neural networks (Ahsan and Ibrahimy, 2009).

Surface EMG sensors are non-invasive and take the form of electrodes placed directly on the surface area of the skin from where muscle activity is to be recorded. Intramuscular EMG utilizes a needle and fine wire, which are inserted through the skin into the muscle and then activity recorded (Ahsan and Ibrahimy, 2009) (Ohnishi et al., 2007). Todd Kuiken, a physician and biomedical engineer at the Rehabilitation Institute of Chicago (RIC), developed a technique called "targeted re-innervation" to amplify nerve signals on an amputee's stump, so that sensors could pick up the signals in a non-invasive manner (Kuiken et al., 2009). This improves the myoelectric prosthesis function. Amputated nerves still carry action potentials from the motor cortex in the brain. Kuiken's technique involves a surgical procedure by which amputated nerves are transferred to new muscle and skin in unused areas such as the chest or to the unused muscles on the arm stump, if it was remaining. The transplanted nerves grow into the muscles and skin and the patient is able to feel parts of the amputated arm through the muscles of the chest or stump accordingly. The amputee could practically feel the amputated hand just by touching various parts of the chest/stump. She was actually touching the "phantom arm". At this stage, the patient is fitted with the customized myoelectric arm. The nerve signals are picked up by the electrodes of the arm and fed in to a computer, which is programmed to differentiate the activation signals. The decoded signals are used to drive the servomotors in the arm to give a semblance of the same DoFs experienced with the biological one (Ohnishi et al., 2007) (Kuiken et al., 2009). Thus in Kuiken's method, the key factor for driving the prosthetic arm is the amplification of the EMG signals.

The model developed in Rezzoug and Gorce (2003) is termed as a "biocybernetic method" that learns hand grasping posture. The model consists of two parts. The first part deals with learning the IK of the fingers, given the fingertip position. And the second part, deals with optimization of the configuration space of the hand through direct associative learning (Rezzoug and Gorce, 2003). The learning algorithm used in the networks is back-propagation. The hand pose configuration network consists of two hidden layers and an output layer. Another example of neural network-based animation is Kim et al. (2000), where the neural network controls the hand movements to play the violin. Since a lot of finger dynamics is involved in animating a hand playing a violin, it is difficult to animate it by hand. So a neural network takes care of each hand configuration. In addition to that, a finger positioning algorithm is used to convert musical scores and decide the finger that touches specific violin strings to play specific notes. The system comprises of three modules, which are: active finger determination module, optimization module and passive finger determination module. A musical score has beats, duration the note has to be played and intensity. So the active finger determination module analyses the state and finger tip position of each of the fingers on the frets for each beat. The optimization module takes as input the fingertip position calculated from the active finger module and outputs the wrist state. The passive finger determination module uses the optimization module with a decreased search space and a spring-damper model to position the fingers (Kim et al., 2000).

Controller synthesis is computationally demanding. The NeuroAnimator in Grzeszczuk (1998) uses neural networks to bypass the common computational burden of complex dynamic equations, and attempts to approximate the perceived reality of conventional physical models. The emulators in Grzeszczuk (1998) use neural networks using back-propagation algorithm and the networks are trained using samples generated by actual simulations of the model. Sampling the simulation at random points along the length of the simulation generates the training data. The input vector is extracted from the sample data and consists of the state of the model, the external forces and the control inputs at a specific time (Grzeszczuk, 1998). The neuromuscular network of the human body is the key to understanding the dynamics of the human body. Murai et al. (2008) combines a neural model with a physically accurate biomechanical system. There are different types of feedback and feedforward loops in the neuromuscular system. Murai et al. (2008) uses the neural network to model these loops. The physiological properties of the muscles are considered in this research. Murai et al. (2008) models the patellar tendon reflex (the reflex induced when the knee is tapped) by simulating the nerve signals. This neuromusculoskeletal model converts muscle tension output to muscle activations and works on a reduced model in terms of the number of muscles and nerves. The model also incorporates a muscle spindle system to feed back the length and velocity of the muscle to the nerve. Optical motion capture is analysed and muscular tension is calculated. The motion capture data is dimensionally reduced using Independent Component Analysis (ICA).

Pollard and Zordan (2005) creates a physically-based animation of hands that is comparable in quality with motion capture using modified forms of PD control implementing both passive and active control of the hand. Motion capture is used to extract parameters (limits and positions) for both passive and active control. Pollard and Zordan (2005) uses three tuning parameters in the equation which are set by the animator for the simulation. Active control is achieved via a finite state machine (FSM) system that activates states depending on proximity of the hand to objects to trigger directed grasp animation. Joint angles and angular velocities are linearly blended between setpoints, as distance to the object decreases (Pollard and Zordan, 2005). The motion library is used to extract the parameters and the set of linear equations, generated from each frame of the motion capture, is used to solve for the unknown position and velocity of the setpoints. Thus Pollard and Zordan (2005) demonstrates a layered approach to passive and active control in a physically-based manner. The hand animations are created using the popular real-time physics engine, Open Dynamics Engine (ODE) Pollard and Zordan (2005)

2.4 Grasp Psychology and Learning: A Primer

Grasping is as much a mental task as it is physical. The human brain performs numerous calculations subconsciously right from the time of subject acquisition to approach to the actual execution of the grasp. Grasping psychology mainly deals with the higher level task-based instructions (decision making) which internally is translated to low level motor neuron functions to navigate space through the motion of limbs caused by the activation of muscles. So there are many different aspects that have to be considered when understanding a seemingly trivial task of reach and grasp, both from a physical as well as psychological point of view. Weiss and Jeannerod (1998) is a survey paper that looks into a few of the problems and research that fall in the category of neurophysiology, anatomy and related mathematical formulation for simulation. Weiss and Jeannerod (1998) tries to understand the motor coordination of limb movements during the execution of different tasks. The limbs have a wide range of degrees of freedom (DoF) and this flexibility comes with a computational load on the motor control system of the brain. Weiss and Jeannerod (1998) specifies that this computational load can be simplified by the command signatures of the movements and selected based on optimality rules that induces the limb to move as a single unit rather than as muscle and joint groups. This idea is derived from the Bernstein concept of "muscle synergies" and "coordinative structures" (Macpherson, 1991). The end effector of the limb is a good candidate for the optimization of DoFs in space. Hogan and Flash (1987) cited in Weiss and Jeannerod (1998) demonstrated this in the "minimum jerk" model, where the jerk is defined as the "fourth derivative of position" and went on to predict that the central nervous system uses the spatial path of the hand as a control variable, implying that movements are planned in Cartesian coordinates (Weiss and Jeannerod, 1998). This theory directly conflicts with the other theories that propose that movements occur in the joint space of limbs. In this alternate theory, the controlling parameter is the individual joint position and it predicts curved trajectories. The concluding theory, that combines the two disparate theories, as stated in Weiss and Jeannerod (1998), suggests that the control system in the human brain resorts to different control strategies depending on the type of tasks at hand. There is a concept termed as "postural coding" which states that different muscle synergies (activation of different muscles at the same time) cause some of the DoFs in the limb to "freeze" (Weiss and Jeannerod, 1998). This is a way by which the motor control system manages redundancies in DoFs, by freeing them when a change in a grasping task or change in the object shape demands it.

In reaching and grasping tasks, even changing the orientation of the object during grasp reach produces corrective movements to ensure an efficient limb posture. Weiss and Jeannerod (1998) states that these results fall into "the general framework of the equilibrium-point hypothesis". The equilibrium-point hypothesis states "the motor system specifies goal positions for the limbs, not movements between position" (Weiss and Jeannerod, 1998). In theory, it works like a PD controller, where the motor system presets the stiffness of the muscles and the spring-like nature of the muscles attain the equilibrium position for the particular stiffness. Weiss and Jeannerod (1998) cites research whose findings contradicts the equilibrium-point hypothesis, where computed velocity of the limb trajectory during a pointing action does not correlate with the stiffness and the corresponding torque.

Proprioception, or the ability of the brain to map the position of the body's external organs/limbs in relation to each other in space is the outcome of the contradicting theory. Weiss and Jeannerod (1998) suggests that this structural map of the body in the brain is an offshoot of sensory information feedback, from the body's interaction with the environment. Weiss and Jeannerod (1998) continues by giving the example of a grasping task, where observations on grasping behaviour has shown that "spontaneous hand positions tends to tolerate initial discomfort for the sake of final comfort", whereby the end-state comfort shows that the system is planning for future task demands. According to further research cited in Weiss and Jeannerod (1998), the cerebellum plays an active and important role in executing the feed-forward control mechanism. In a related research, Miall et al. (1993) describes the cerebellum as a "Smith predictor", which means that the cerebellum retains a prediction of the sensory results of a movement in order to compare it with actual sensory feedbacks. This enables the cerebellum to provide corrections and adapt to the actual goal at hand. Hogan and Flash (1987) also provides a hypothesis that the existence of twin Smith predictors in the cerebellum cannot be discounted, as it can provide an explanation of how a visual sensory cue is translated into actual motor commands that can move the limbs to the intended target. The Smith predictors would be situated in the lateral cerebellum (working in a visual and peripersonal - the space within the reach of a limb - coordinate space) and in the intermediate cerebellum (using motor coordinates). It has been observed in cerebellar patients, their inability to produce a sustained limb trajectory, due to the occurrence of uncoordinated muscle activations. This is suggestive of corrupted or complete erasure of the internal limb representation (Weiss and Jeannerod, 1998). Research has also shown that proprioceptive information is necessary for updating the inertial properties of the limbs (Weiss and Jeannerod, 1998). This is essential for a feed-forward control mechanism.

Motor coordination in the temporal domain is another concept that Weiss and Jeannerod (1998) looks into, where it was found that for a goal oriented pointing task involving the eye, head and arm movements, the motor commands were all fired simultaneously. The areas of the brain found to be affecting temporal coordination of limb movements include the cortical areas like the premotor cortex, the parietal cortex, the mesial motor cortices and also the supplementary motor area (Weiss and Jeannerod, 1998). The observation of patients with severely disrupted parietal area (a condition called *parietal lesion*) provides verification in the research. Parietal lesions cause inhibition in the coordination required in reaching and grasping and also finger movements during tactile exploration of an object. Parietal patients have also been found to lack the ability to synchronize the movements for manipulation required for the correct contact between the object and skin receptors, during an exploration task.

Finger movements comprise a majority of the DoFs of the human hand. Research, such as in Nakamura et al. (1998), has been done to study finger movements in detail during grasping tasks. Nakamura et al. (1998) investigates the movements of the fingers during simple flexion and hyperflexion and then compares the results with a grasping task and analyzes the ability of the hand to adjust its phalangeal orientation to suit the shape of a wide variety of objects. Nakamura et al. (1998) uses a two dimensional motion analyzer which analyzes motion from video sampled at 1/60 frames. The finger has three joints, namely, the metacarpo-phalangeal (MP), proximal inter-phalangeal (PIP) and distal inter-phalangeal (DIP) joint. For generating smooth, versatile and practical movements, all these joints move in a coordinated manner. In the case of simple finger extension movement, the experiment was able to deduce that the initial movement started from the proximal joint to the distal joint and the end of the action started from the distal and ended in the proximal joint. In deliberate actions like grasping a disc and during the planning stages (prehension), the end-effector joint initiated the movement, and termination initiated from proximal to distal. The muscles for the MP joint movements in the extension are the extensor digitorum and the extensor indices, which are the extrinsic muscles of the hand. It is theorized in Nakamura et al. (1998) that for during the sequence of activation for extension, the extrinsic muscles of extension activate earlier than the interosseous and lumbricalis which are intrinsic muscles. For grasping an object (disc), it was found in Nakamura et al. (1998), that end-effector movements are more prominent and are initiated by the intrinsic muscles of the hand and starts from the distal to the proximal joints. According to Nakamura et al. (1998), even though the intrinsic muscles initiate the grasp movement, it is the extrinsic muscles, the finger flexors, which play a role during the actual act of grasping the disc. In spite of that, distinct intrinsic muscle activity is detected in the grasping part of the movement. Nakamura et al. (1998) suggests that this is indicative of co-activation of both sets of muscles in order to perform minute adjustments of the fingers to the size and shape of the disc. The MP and DIP joints accomplish the adjustment of the fingers to different sizes of discs in a coordinated manner during the grasping movement.

As quoted in Smeets and Brenner (1999), “Grasping is a complex movement involving rotations of several joints and one end-effector”. Interpretation of grasps involves selecting variables that would define choice (Smeets and Brenner, 1999). Research in Jeannerod (1981) written by Jeannerod, cited in Smeets and Brenner (1999), proposed a selection. He theorized that two channels, one controlling the transport of the hand and the other the size of the grip. This is the “classical approach” (Smeets and Brenner, 1999). The popularity of this approach is due to the fact that the two channels correspond to two anatomical structures and to two distinct types of perceptual information (Smeets and Brenner, 1999). From an anatomical perspective, the transport component moves the wrist to the target object irrespective of the pre-shaping of the fingers by the grip component. From an informational perspective, the transport component is based on extrinsic properties (position and orientation) while the grip component is based on intrinsic properties (size, mass, shape, color). Thus perceived size of the object is assumed to be the deciding factor on the grip size. But the ambiguous classification of orientation (originally classified as extrinsic but sometimes as intrinsic) creates problems. Objects change orientation, so grasp orientation remaining stationary; the size of the grip changes to accommodate the object with its new orientation. This is also true if the grasp orientation is changed keeping object orientation stationary. Thus it is seen that grip size is depending on both extrinsic and intrinsic properties of the object. Thus the originally proposed independence of the visuo-motor channels no longer applies and shows where the classical approach fail.

Smeets and Brenner (1999) also elaborates on the failure of the anatomical perspective in the classical approach. The first contradiction provided by Smeets and Brenner (1999) is that there is a difference when using the terms proximal and distal at the level of muscles and at the level of joints. Proximal or extrinsic hand muscles affect the pre-shaping of the fingers and also orienting while the distal or intrinsic hand muscles activate affect the distal joints when the object is touched. The classical approach emphasizes that the activation of the distal muscles define the grip and the proximal muscles influence the movement of the arm. Transport of the arm from an anatomical point of view, is the other problem in the classical approach, given in Smeets and Brenner (1999). Suffice it to say that Smeets and Brenner (1999) refutes the “classical approach” claims that the movement variables “transport” and “grip” in grasp control can be directly mapped to the anatomical attributes of “proximal” and “distal”. The classical approach is a model that is based on parameters or variables derived from experimental results. The alternative approach taken by Smeets and Brenner (1999) to avoid the discrepancies of the classical approach is by taking into consideration purely the requirements of a stable grasp than variables such as “grip” and “transport”. The stability of a grasp, according to Smeets and Brenner (1999), depends on the positioning of the fingers on the surface of the object. This is achieved by looking at the perpendicularity of the lines connecting the fingers to the surface of the object. The lines go through the object and through the center of gravity of the object. Suitable grasp positions are determined by the central nervous system before actual grasping takes place and how it decides the position is still little known and highly debatable.

Smeets and Brenner (1999) approaches a grasp as merely the positioning of the thumb and the other fingers into these positions and does not consider any extraneous factors like object size determination. The trajectory of the finger in a pointing task is a straight line, but not so during prehension as the grip begins to close around the object. There is strong curvature in the trajectory (Smeets and Brenner, 1999). The aim of Smeets and Brenner (1999) is to model the straight line and curved trajectory behaviour using the same principles. Smeets and Brenner (1999) later explains that “slow curvatures of slow pointing movements depend on the orientation of the surface to which the movement s are directed such that the trajectories tend to end perpendicular to the surface” and that the curved path is due to constraints imposed on the movement. Smeets and Brenner (1999) uses an existing model called the “minimum-jerk” model, as it is a simple model with optimized constraints at the beginning and end of the movement, to build on and implement a generic principle that adheres to the straight line and curved trajectory behaviour.

It is vital to place the fingers on strategic locations on the surface of the object to have good grasp control during manipulation tasks. The research in Jeannerod (1981) ventures out to prove this above stated fact. Jeannerod (1981) looks deeper into how the center of mass (CoM) of objects govern the decision for optimal grasping locations on the object surface. The experiment in Jeannerod (1981) consisted of test subjects who were provided the CoM of the objects they were to grasp, a priori. They were also made to grasp objects whose CoM were not known. One of the preconditions of the grasp test was to minimize the object roll. It was seen that, when the subjects experienced the same CoM over many trials, they were able to anticipate the digit forces necessary to achieve the optimization criteria. And this was for objects whose CoM was not known at the beginning and also when the subjects did not have the freedom to choose digit placements on the object. Jeannerod (1981) successfully demonstrated that “when the subjects can choose contact points and can anticipate object properties, they implement anticipatory force mechanisms in parallel with careful selection of digit placement". Jeannerod (1981) also proved that a priori knowledge of object attributes like shape and size plays an important role in the choosing of digit placements in grasping. Jeannerod (1981) was able to study the interaction between the anticipatory control mechanisms and digit placement control during the process of CoM prediction. Humans seem to have a bias towards a center CoM location on objects. This could be because of our daily interaction with symmetrical objects. So for an off-center CoM, the subjects tend to use digit placements as if for a centered CoM location. According to Jeannerod (1981) this strategy is successful in minimizing the roll of the object. It was also found that subjects adapted to the changes in CoM location by varying the thumb and index finger contact points. Jeannerod (1981) conjectures that this is related to the larger force production abilities of the two digits compared to the rest. And so, strategic placement of these two fingers plays a crucial role in minimizing the roll.

In a similar wake, Lukos et al. (2007) also delves into the physicality of the grasping process. Fingers act as agonists and antagonists during the application of forces for a grip. To prevent an object from slipping, fingers act as agonists contributing to the net grip force, but the fingers with the exception of the thumb exert moments of force or torque which is opposite in direction about the pivot point created by the thumb. Thus these two pairs form torque antagonists. There is a minimization of total finger force required. In order to achieve that, the fingers involved in torque generation should not produce any force. But at the same time, in order to prevent the object from slipping, a contribution to the total grip force from the very same fingers is required. So these are conflicting requirements and the central nervous system (CNS) has to find a balance between the two. Lukos et al. (2007) investigates the different strategies the CNS undertakes to achieve the two contradicting force and torque requirements. The study in Zatsiorsky et al. (2002a) is used to develop a neural model of torque control in Zatsiorsky et al. (2002b) and it is detailed in Section 2.3 of this chapter. Another study Shim et al. (2005) also investigates the synergistic activity of hand muscles, especially in the digits, while an external torque is applied on the object that is grasped. The study evaluates the digit forces/moments required to cancel out the external torque and keep the object steady. The data recorded from the subjects proved that forces were applied in all three dimensions, even though the external torque was applied only along one axis on the object. Shim et al. (2005) explains the cause of this as the "chain effects prompted by the non-collinearity of the normal forces of the thumb and the four fingers". The observations in Shim et al. (2005) support that the CNS applies the normal and tangential forces through the fingers on the object and that this is hierarchical.

Research like in Miller et al. (2003) investigated algorithms that would help in automating the grasp planning, given arbitrary shapes. Daily use objects come in a variety of shapes and sizes. But due to human design, they can be more or less fitted into a shape primitive group, like spheres, cylinders, cuboids etc (Miller et al., 2003). Miller et al. (2003) uses these shape primitives to figure out stable grasps on an object. The system developed by the research is called *GraspIt!*. It works in two stages, the first being to generate sets of initial grasp locations on the surface of the primitive that represents the object. The second part deals with a qualitative analysis of the generated grasps. Miller et al. (2003) defines certain heuristic grasp strategies to generate grasps, in order to reduce the grasp search space. These strategies act as constraints to narrow down a grasp. The primary constraint being, the start position of a grasp includes the position, orientation in 3D space and a grasp preshape (Miller et al., 2003). Orientation is subdivided into the palm approach direction and thumb orientation. The grasp preshapes are defined for the set of primitives in order to constrain the positions and orientations of the grasp starting locations (Miller et al., 2003). Evaluating the generated grasps is based on an algorithm outlined in Ferrari and Canny (1992). The quality of a grasp is often analysed in terms of the *wrench space*, which is the set of all possible wrenches. A wrench is a vector of vectors consisting of the force vector and the torque vector on the object (Ferrari and Canny, 1992). Building on the use of quantitative Steinitz's Theorem[2] in grasp generation, as detailed in Kirkpatrick et al. (1992), Ferrari and Canny (1992) examines the efficiency of a grasp given as "the value of the radius of the largest closed ball, centered in the origin of the wrench space, contained in the set of all the possible wrenches that can be resisted by applying at most unit forces at contact points" (Ferrari and Canny, 1992, p. 2291). From this set Miller et al. (2003) computes the convex hull of wrenches, which represents a grasp wrench space called L1 wrench space (Ferrari and Canny, 1992). This wrench space has the property that the sum total of all finger contact normal forces is one. If the origin (of the wrench space containing the contact wrenches) is not situated in the convex hull of the contact wrenches then it means that external wrenches (force and torque) exists which the finger forces cannot counteract or balance (Miller et al., 2003).

A much more detailed description of the *GraspIt!* System is given in Miller et al. (2005). Miller et al. (2005) looks deeper into the dynamics simulation as-

[2]Steinitz's Theorem : Classical Steinitz theorem as stated in Kirkpatrick et al. (1992) : If the convex hull of a subset S of Euclidean d-space contains a unit ball centered on the origin then there is a subset of S with at most m points whose convex hull contains a solid ball also centered on the origin and having residual radius,

$$r = 3d(2d^2/m)^{2(d-1)}$$

and m being sufficiently large.

pects of the system and in addition to looking at robotic grippers also looks into simulation of the human hand and tendon actuation. Miller et al. (2005) also discusses the SVM regression used to create a mapping between the shape of the object, grasping parameters and the quality of the grasp. The regression system accepts a vector consisting of shape and grasping parameters and returns a single scalar value indicative of the grasp quality (Miller et al., 2005). The researchers who were a part of Miller et al. (2003)Miller et al. (2005) explains their efforts in performing grasp analysis for human hands through Ciocarlie et al. (2005). In Ciocarlie et al. (2005) contact models are investigated, especially deformable contact models as found on human fingertips. Contact with objects cause exchange of forces between the object and the hand. Rigid contacts, like the contacts of robotic grippers, are easier to process as forces can be calculated using a single contact (Ciocarlie et al., 2005). In the case of human hands, the contact point on the fingers change the shape (becomes a contact area) depending on the amount of force applied (Ciocarlie et al., 2005). The deformable fingertips are modelled using the Finite Element method (Ciocarlie et al., 2005). Using Computed Tomography (CT) scans to create anatomically accurate model of a thumb, Ciocarlie et al. (2005) uses the Finite Element method to simulate the forces applied on the thumb during the contact with a rigid body. Unlike mathematical algorithms that analyses grasps, there are data driven methods also (Li and Pollard, 2005)(Rijpkema and Girard, 1991). Li and Pollard (2005) uses a shape matching algorithm to synthesize grasps. The algorithm used in Li and Pollard (2005) analyses the shape of the palmar surface of the hand during contact to extract the shape property of the object. If a similar distribution of features is found on another object, then the grasp is reused for that particular object. Li and Pollard (2005) describes it as extracting "probabilistic samples of a global shape function from the hand shape". Rijpkema and Girard (1991) adopts a three-phased approach, namely the task initialization phase and the target approach phase and the grasp execution phase.

Grasping is perfected through the experiences that human beings go through in the different stages of growth period (Li and Pollard, 2005). This *knowledge* is stored in a database, in the form of "pre-calculated strategies for different categories of situations" (Li and Pollard, 2005). A knowledge base like this helps in the reduction of the huge search spaces possible in a grasp (Li and Pollard, 2005). The efficiency and quality of a grasp increases with the number of contact points between the hand/gripper and the object (Pollard, 2004). With this in view, Pollard (2004) describes a method that synthesizes many-contact (eight or more frictionless contacts or five or more contacts with friction) grasps. Through the method, force-closure grasps, partial force-closure grasps and grasps that go beyond a given quality threshold are synthesized (Pollard, 2004). Pollard (2004) constructs a family of grasps from a single example grasp. The example grasp is used for grasp optimization where it is used as a template to synthesize grasps that has the same attributes as the example (Pollard, 2004). Rather than implement a global optimization, which requires a high order of complexity when the number of contacts increases, the method described in Pollard (2004) allows the synthesis of many-contact grasps. Optimization methods provide alternatives and extensions to synthesize many-contact grasps (Pollard, 2004). But since the optimization takes place in "space exponential in the number of contacts, each contact can be placed anywhere on the object surface" (Pollard, 2004, p. 599). In order to solve the optimization problem using non-linear equations, N contacts are assumed to be on N surfaces (Pollard, 2004). Thus, according to Pollard (2004), the complexity is $O(N)$ in a force/torque space, being exponential in N. The algorithm in Pollard (2004) converts an example grasp into an equal set or class of grasps, which is projected onto any arbitrary geometry. Given a geometric model of the object to be grasped, a grasp prototype and a task as inputs, Pollard (1996) synthesizes a suitable grasp starting from the grasp prototype. The grasp prototype is represented as contact wrenches and contact point coordinates on the surface of the object (Pollard, 1996).

Most of the research investigated thus far in this section, concentrate on how hands conform to objects and mechanisms by which pre-shaping and planning of grasps can be performed by keeping this view in mind. But the objects subjected to grasps themselves can embody interaction information. These objects are termed as smart objects (Kallmann, 2001)(Francik and Szarowicz, 2005). The three main engineering design features, namely, functional features, design features and manufacturing features given in Parry-Barwick and Bowyer (1993), are used as an inspiration by the research in Kallmann (2001) to develop a new type of feature called interaction features that defines a feature as "all parts, movements and descriptions of an object that have some important role when interacting with an actor" (Kallmann, 2001, p. 51). Interaction features can be classified into four groups: properties that are inherent to the object design, like parts, center of mass (intrinsic properties), properties that aid the virtual hand or character in animation like approach direction, surface points (interaction information), reactive properties that deal with how the object react to interaction (object behaviour), reactive properties for the hand or character (expected character behaviour) (Kallmann, 2001). Thus animation of the hand, when using smart objects, takes on a degree of autonomy and is a derivative of the interaction feature of the object.

2.5 Summary

Having reviewed a substantial amount of previous research, it can be concluded that game systems require computationally lightweight muscle actuator and control systems to maintain the high frame rates. The control methods for muscle dynamics can be broadly classified as in Table 2.2.

Table 2.2: Control methods in Muscle dynamics

Type	**Methods**
Mathematical/Algorithmic (deterministic)	Static optimization (inverse dynamics) Dynamic optimization(forward dynamics)
Machine learning (stochastic)	Artificial Neural Networks, Genetic Algorithms, Genetic Programming

The deterministic methods require intensive computations to model the complexities of muscle dynamics. These methods are not compatible with a game physics environment. The simplistic PD control is ideal for game purposes but are limited in the scope of effects possible and cause tuning problems as the number of articulate joints increase.

Gestural and grasping capability is pre-animated with very little autonomous behaviour (state-based motion blending). This lacking ability hampers the manner by which the game characters interact with game objects. Physics-based and procedural grasping like in Pollard and Zordan (2005) is a step towards better interaction. Grasp psychology defines a set of rules that humans follow subconsciously. The end animation of grasping can be created using a wide variety of techniques like inverse kinematics, forward kinematics, PD controllers or biomechanical muscles. The important point to note is whether the methods are viable within the simulation capabilities of a real-time physics engine. Grasping objects without slippage require the application of opposing torques on the object by the fingers Zatsiorsky et al. (2002b). Control mechanisms borrowing principles from neuroscience adapt to environmental perturbations and perform complex coordination. Such neural models can be used to develop grasping systems with torque control. The methods proposed in this thesis are generic in nature and are suitable for a wide spectrum of character animation including full-body animation.

Research done in fields like physics-based animation, musculoskeletal coordination and neural dynamics is extensive, as is evident in the literature examined. A large amount of progress has been made in biomechanics and computer graphics with a tighter interaction between the two fields. It is evident that better and more unified models with advanced control systems will make way into production pipelines and real-time applications and into completely different real-world fields like robotics. It is only a matter of time.

3

HAND ANATOMY

3.1 Introduction

The human body is a prime example of a compact and highly advanced organic machinery. It constitutes an array of sub-systems endowed with efficient background processing capabilities (like the cardiac and smooth muscle types) that are required for task completion. For simulating the physiological functions of the human body that generate motion, *myological* study is crucial and in addition to that *osteological* study is essential to understand the structure of the bones and joints and how the action of the superficial and deep muscles on them influence articulated motion. The hand being a complex articulated organ, such a study is invaluable in replicating the functionality. This chapter provides a brief look into the anatomical structure of the hand. For detailed anatomical reference, see (Gray, 2006).

3.2 Hand Bones and Joints

The skeletal framework of the human hand is by far the complex segmented dynamic structure (other than the spinal column) in the human body, containing 27 bones (see Figure 3.1). The skeletal structure, in addition to providing rigidity and form, also aids in providing resistance to external forces during physical action. The presence of skeletal joints give rise to the linked structure of the hand. The movements capable and incapable by the individual segments of the hand, due to the constraints imposed by the skeletal structure, arises from the shape and arrangement of the separate bone segments.

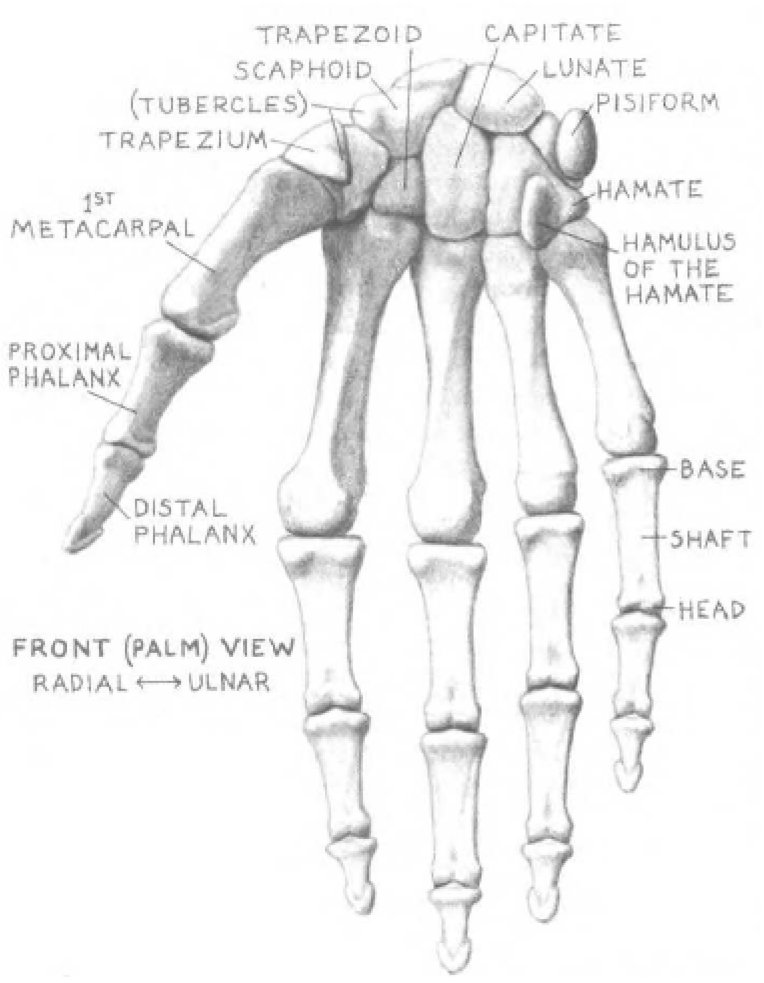


Figure 3.1: Skeletal Structure of the Human Hand. Image courtesy (Goldfinger, 1991)

The bones of the hand are divided into the *carpal* and *metacarpal* bones and the finger bones called as *phalanges* (Figure 3.1). The phalanges are small, short and tubular. There are three phalanges for each finger, with the exception of the thumb, which has two. The *carpus* is a collection of eight bones, which are short, spongy and arranged in two rows having four bones each. The *proximal*, which is the nearest to the *radius* and the *ulna* bones of the forearm (Figure 3.1), consists of the following bone segments, which are named from the thumb: the *scaphoid* bone, the *lunate* bone, the *triquetral* (triangular) bone and the *pisiform* bone. The first three of the above aid in the articulation of the hand at the radius bone by forming an ellipsoid convex surface with the forearm (Prives et al., 1989). The pisiform bone on the other hand articulates only in conjunction with the triqueral and is a sesamoid bone [1]. Sesamoid bones are recurring bones in the

[1]A sesamoid bone is a bone, which is embedded within a tendon, and is commonly found at skeletal joints. The primary purpose of the sesamoid bone is to displace the tendon from the centre of the joint in order to increase the moment arm of the joint (Gray, 2006).

metacarpophalangeal and interphalangeal joints of the thumb and less so with the other finger joints The *distal*, farthest from the radius and ulna, consists of the *trapezium* otherwise known as the larger *multangular* bone, the *trapezoid* (smaller multangular bone), the *capitate* bone and the *hamate* bone (Prives et al., 1989). The bones are shaped in a manner that helps in articulation with the neighbouring bone.

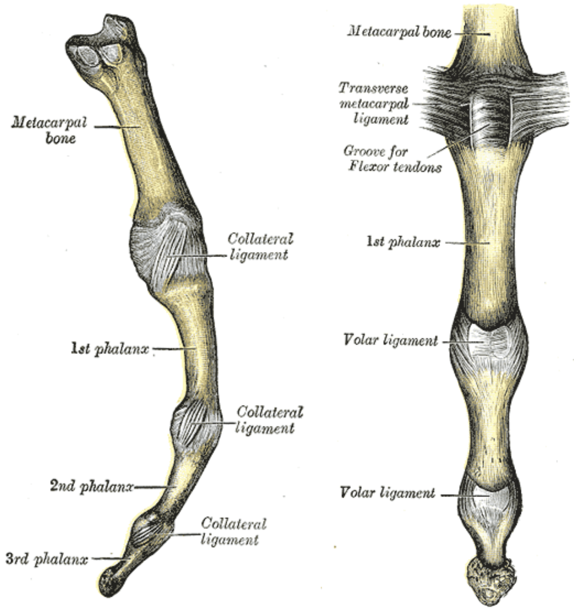


Figure 3.2: The ligaments binding the metacarpal and phalanges, Image courtesy (Gray, 2006).

Joints form the connection between bone segments to help in articulation between the linked segments. The ligaments, in the skeleton of the hand, hold the carpus, metacarpus and phalanges together (see Figure 3.2). Joints, in the simplest terms are a combination of the ligament binding and the concave and convex surfaces of the participating bones. The manner in which the connection surfaces are constructed is of extreme importance to the mobility of the joint.

The wrist, with its eight carpal bones has a complex networking arrangement of ligaments (Figure 3.3) due to its fragmented structure. A *condyloid joint* is a type of synovial joint in which a ball-like articular surface is placed within a concave or elliptical cavity, allowing movement in two planes. This type of joint allows for movements like *flexion*, *extension*, *adduction* and *abduction*. There are various types of synovial joints often classified depending on the type of movement allowed by them. Except for the pivot and ball-and-socket joint, all the other joint types are present in the human hand (Napier and Tuttle, 1993). The phalanges articulate with each other using hinge joints (having a single axis of movement). They articulate with the metacarpals using a bi-axial joint. Adjacent

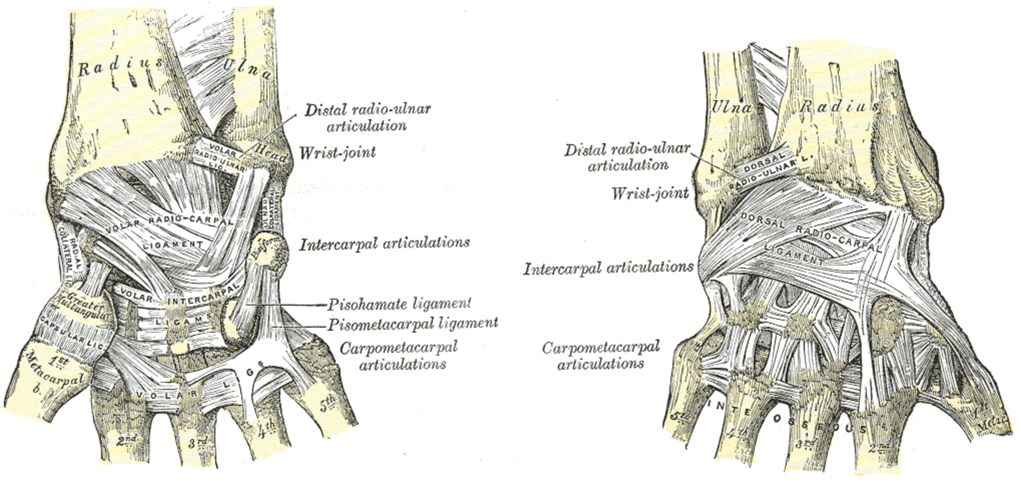

Radius
Ulna
Distal radio-ulnar articulation
Wrist-joint
Intercarpal articulations
Pisohamate ligament
Pisometacarpal ligament
Carpometacarpal articulations
Volar radio-carpal ligament
Volar intercarpal ligament
Volar ligs.
1st Metacarpal b.
2nd
3rd
4th
5th
Ulna
Radius
Distal radio-ulnar articulation
Wrist-joint
Intercarpal articulations
Carpometacarpal articulations
Dorsal radio-ulnar l.
Dorsal radio-carpal ligament
Interosseous

Figure 3.3: Complex ligament network on the wrist Left, Anterior or palmar view Right, Posterior or dorsal view. Image courtesy (Gray, 2006).

carpal bones articulate using the arthrodial joint or plane joint, due to sliding action. The thumb metacarpal articulates with the trapezium using the saddle joint, which combines the properties of a bi-axial joint and a ball-and-socket joint (Napier and Tuttle, 1993). This joint is crucial for the opposability of the thumb. The wrist-joint proper is a condyloid joint (Gray, 2006). All movements with the exception of rotation are possible. A limited amount of ulnar flexion (adduction) and radial flexion (abduction) is also allowed. Rotational effect is made possible by the pronation and supination of the radius on the ulna (Gray, 2006). *Circumduction* is achieved through the consecutive actions of adduction, extension, abduction and flexion (Gray, 2006). For the mid-carpal joint, the main movements are flexion, extension and a low degree of rotation (Gray, 2006). Rotation is made possible with the head of the capitate rotating around a central vertical axis through it along with a small gliding movement laterally and medially. The greater and lesser multangulars on the radial side and the hamate on the ulnar side slide forward and backward on the scaphoid and the triangular creating flexion and extension (Gray, 2006). The metacarpophalangeal joints are made of one volar and two collateral ligaments (see Figure 3.2). The joint allows for bi-axial movement and hence are condyloid joints. The movements possible are flexion, extension, abduction, adduction and circumduction. Adduction and abduction are limited and circumduction is executed through a combination of adduction, flexion, abduction and extension.

3.3 Hand Muscles and Tendons

Combined movements of joints allow for various trajectories of the limb. This is made possible because long muscles cross over multiple joints and create antagonistic behaviour. The movements of the fingers are multifarious. They can be flexed and extended at the metacarpo-phalangeal, inter and distal phalangeal joints. They can be abducted and adducted and combined with the flexion and extension, a certain degree of circumduction is also possible. The movements away from the middle finger are called as *abduction* and movements towards the middle finger are termed as *adduction* and *circumduction* is a rotational movement (Gray, 2006). Circumduction is very important in enabling opposability.

The human hand has most of its muscles situated externally on the forearm (with the exception of interosseius and lumbricalis) and hence are called as extrinsic muscles. The forearm muscles are divided into the *volar* and *dorsal* group (Gray, 2006). The flexors and extensors are mostly classified depending on the type of action. There are flexor and extensor muscles for the fingers as well as the wrist. In addition, there are *pronators* and *supinators* that move the radius bone of the arm. These muscles are separated based on the position into two groups, namely, the *anterior group* consisting of flexors and pronators and the *posterior group* consisting of extensors and supinators (Prives et al., 1989). Flexion and extension is made possible by muscles with proximal attachment near the elbow and also with the help of small muscles within the metacarpophalanges named as the *interroseius* and *lumbricalis* (Napier and Tuttle, 1993).

3.3.1 Hand Muscles and Respective Actions

As stated earlier, most of the muscles instrumental in the complex flexing and extension of the fingers are situated in the fore arm. The action of the muscles is distributed to the fingers through tendons. Below given are the various muscle/tendon groups associated with the hand/finger actions. Note that not all anatomical sub-structures are given as the primary purpose is to give the reader a general idea of tendon layout and related action. In addition to maintaining brevity, providing information on every tendon on each finger would be repetitive due to structural similarity.

Flexor Digitorum Profundus: This is the largest of the forearm muscles with the largest contribution of volume to the flexor muscle mass (see Figure 3.4). It helps in powerfully flexing the distal phalanges of four fingers (excluding thumb) and by continued action, flexes the other phalanges and also the wrist (Goldfinger, 1991).

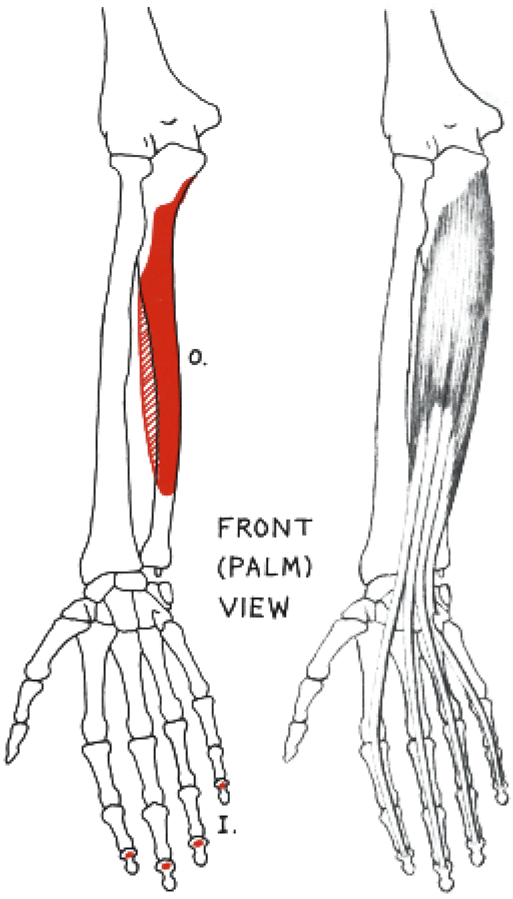


Figure 3.4: The Flexor Digitorum Profundus muscle and branching tendons that flex the distal phalanges of the four fingers of the hand. Image courtesy (Goldfinger, 1991).

Flexor Pollicis Longus: This muscle resides deep in the forearm and helps in the flexion of the distal phalanx of the thumb (see Figure 3.5).

Flexor Pollicis Brevis: One of the two muscles of the thumb that reside in the hand collectively called the *thenar eminence* (see Figure 3.6). This muscle flexes the proximal phalanx of the thumb and assists in the opposition of the thumb with the other fingers and also in the medial rotation of the thumb metacarpal at the carpometacarpal joint (Goldfinger, 1991).

Extensor Digitorum: The extensor digitorum and its tendons extends all the joints of the fingers excluding the thumb (see Figure 3.7). By continued action, it extends the wrist.

Extensor Pollicis Brevis and Abductor Pollicis Longus: The Extensor Pollicis Brevis muscle extends the proximal phalanx of the thumb and through continued action extends the thumb metacarpal bone (see Figure 3.8). The Abductor Pollicis Longus on the other hand abducts and extends the metacarpal of the thumb (see Figure 3.8). It also flexes the hand at the wrist.

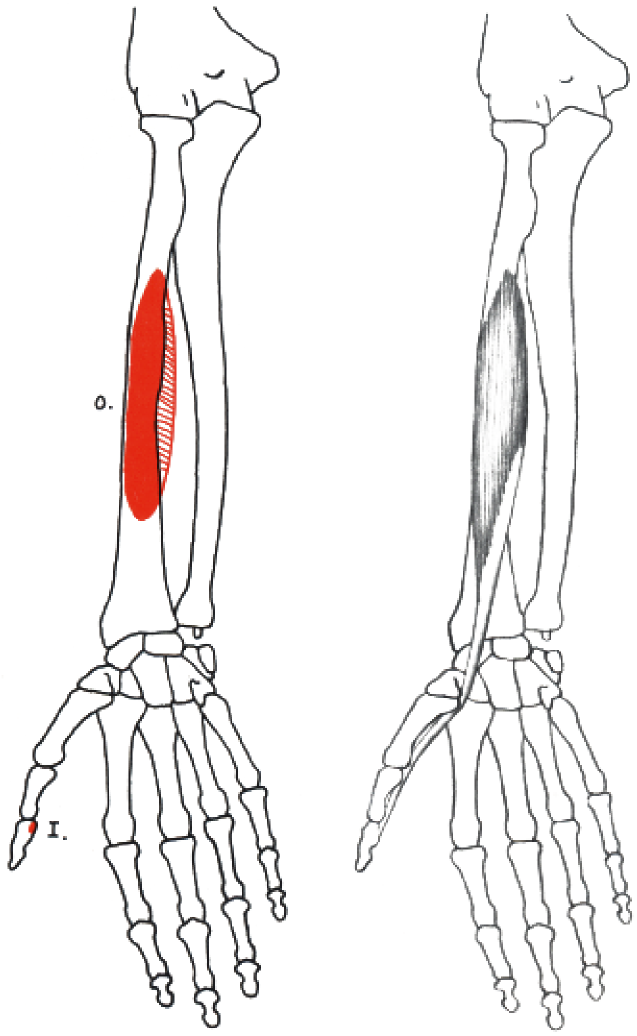


Figure 3.5: The Flexor Pollicis Longus muscle flexes the distal phalanx of the thumb. Image courtesy (Goldfinger, 1991).

Extensor Pollicis Longus: This muscle extends the distal phalanx and through continued action extends its proximal phalanx and metacarpal bone (see Figure 3.9).

Palmar and Dorsal Interossei Muscles:

The muscles that handle abduction and adduction of the three fingers of the hand are the palmar and dorsal interossei. The palmar interosseous muscles *adduct* the fingers towards the middle finger while the dorsal interosseous muscles *abduct* the fingers away from the middle finger.

The thumb has a separate set of muscles that performs the abduction and adduction.

Adductor Pollicis: This muscle adducts and flexes the thumb at the carpometacarpal joint (see Figure 3.10).

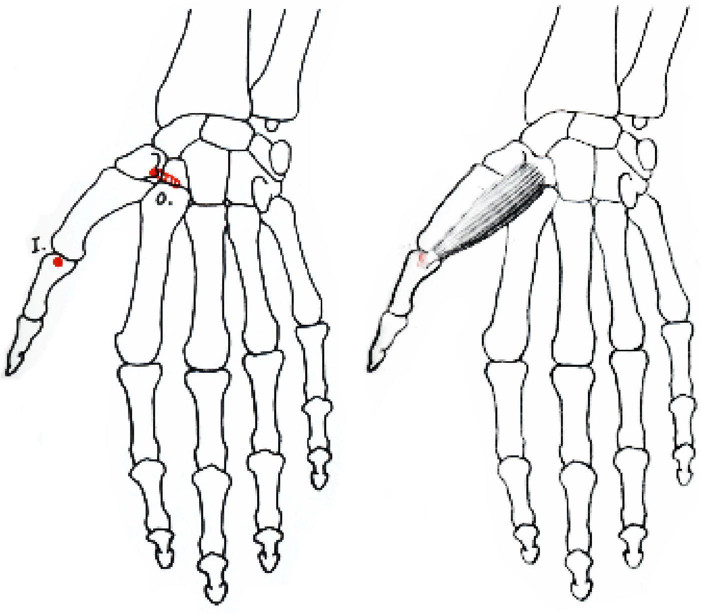


Figure 3.6: The Flexor Pollicis Brevis flexes the proximal phalanx of the thumb. Image courtesy (Goldfinger, 1991).

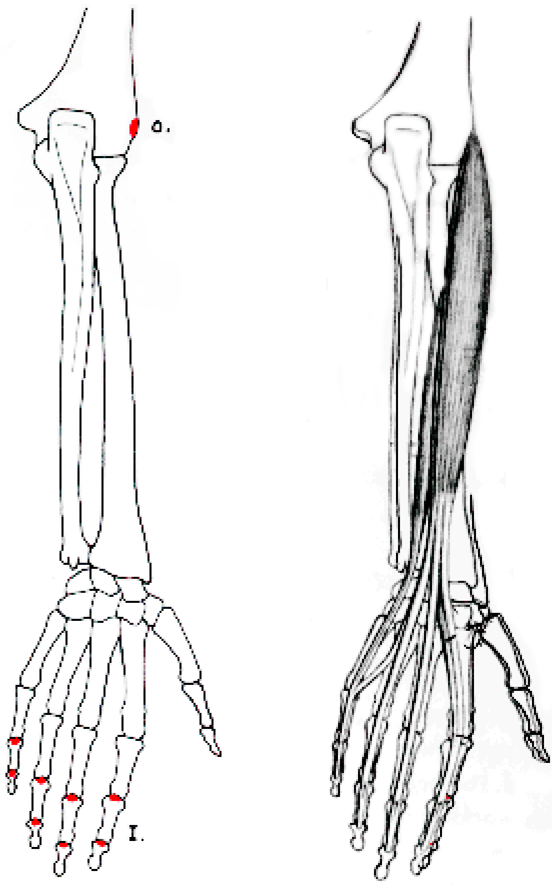


Figure 3.7: The Extensor Digitorum extends all the fingers with the exception of the thumb. Image courtesy (Goldfinger, 1991).

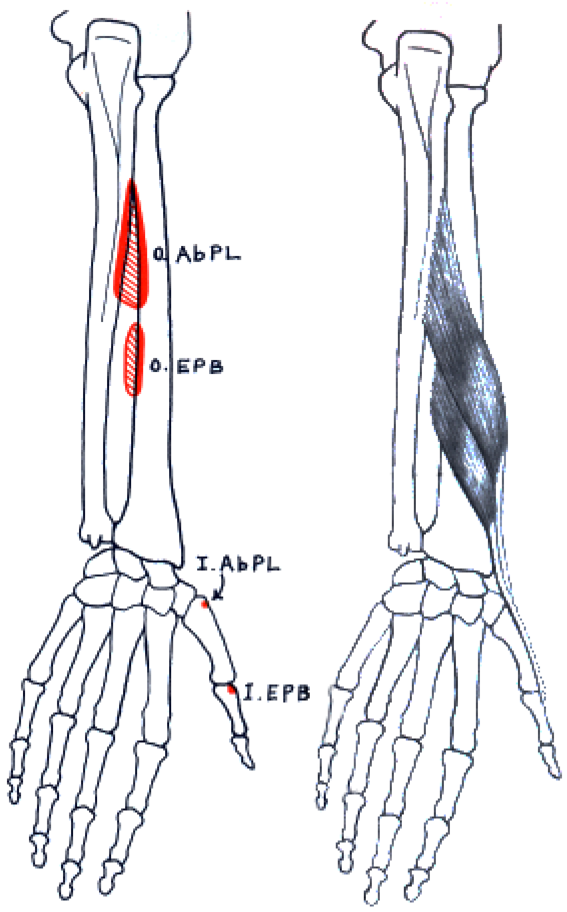


Figure 3.8: The Abductor Pollicis Longus (AbPL) and the Extensor Pollicis Brevis (EPB) of the thumb. The respective origin (O) and insertion (I) points on the bones are also shown. Image courtesy (Goldfinger, 1991).

3.4 Summary

The human body as it exists today is the result of many million years of evolution. The imitation or repair of the versatility and endurance of that evolved product resulted in the exhaustive study of the inner mechanism of the same and that is anatomy. Artists study anatomy to bring realism to poses by way of incorporating the body deformations and torque effects in the character sculpture or painting. Animators study anatomy to get a better estimate of the constraints of the motion possible with the given anatomy and how the physics of the constraints affect the character motion. Irrespective of the tools used, theoretical knowledge of anatomy helps in developing practical models (animation, robots, prosthetics) that are close to the biological counterparts in both form and function. Each muscle when taken individually and its layout (origin and insertion) studied provides a clear picture of the movement possible with that muscle under contraction. Hence, synergestic activity of muscles explores the complex motions produced due to the

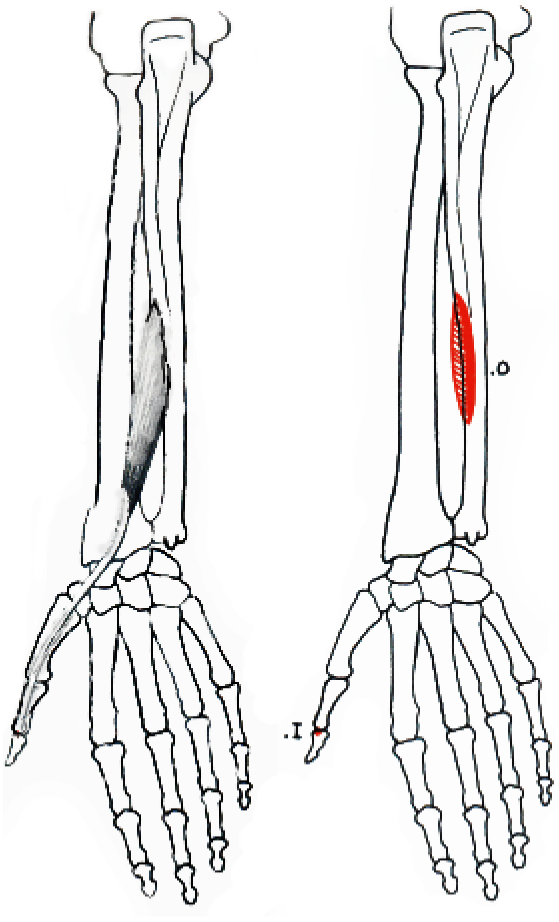


Figure 3.9: The Extensor Pollicis Longus extends the distal phalanx, proximal phalanx and metacarpal bone. Image courtesy (Goldfinger, 1991).

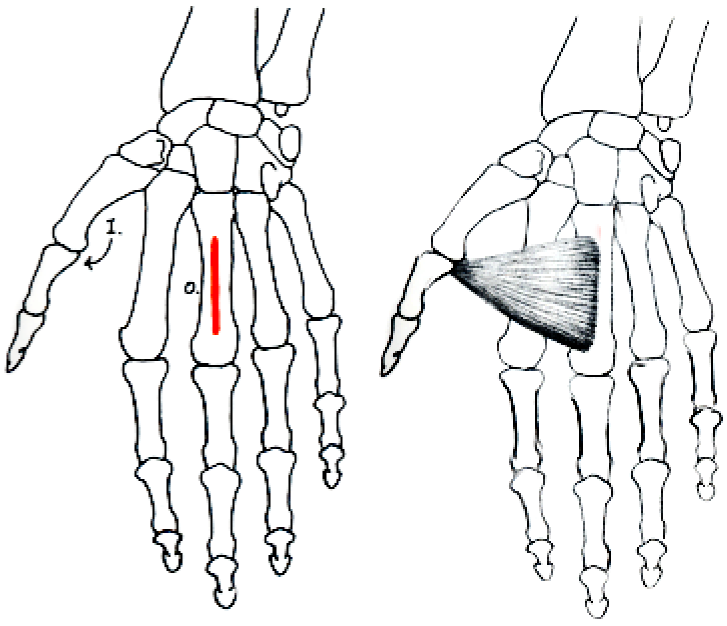


Figure 3.10: The Adductor Pollicis adducts the thumb. Image courtesy (Goldfinger, 1991).

contraction of individual muscles simultaneously. A detailed structural analysis and anatomical description is outside the purpose of this thesis, as such a study would provide the content for an entire book in itself. The purpose of this chapter is to clarify to the reader how the various interactions of bones, ligaments, tendons and muscles play a role in creating the wide variety of movements capable by the human hand, though the anatomical principles remain the same for any other part of the human body.

4

SYSTEM DESIGN

4.1 Introduction

Recreating biological motion using dynamics is non-trivial. Generating muscle excitation patterns or activation patterns for specific recorded motions is computationally intensive and falls under the category of forward dynamic solutions. Though there are dynamic optimization methods that perform this particular task, where normally the objective is to minimize the global error between the simulation output and recorded data, most of them are computationally intensive involving complex differential calculations (Thelen et al., 2003).

The key problem is control and coordination that would keep the motion within realistic limits. In view of this, the two interdependent aims of this research are as follows:

- To create the physical medium by which motion could be generated on dynamic rigid body linkages.
- To create the framework that would solve the coordination problem and achieve the motion using the physical medium.

The two aims constitute a multi-part system and the proper working of a multi-part system, with interdependent modules, depends on a viable architecture. The multi-part system combines three different technologies to produce generalised physical motion - controllers based on evolutionary neural networks, physically-based muscle actuators that actually generate the motion and motion capture training data to teach the controller. The muscle locator-based linear piece-wise tendon unit applies forces according to the layout of the tendon unit on the rigid body skeleton. Thus the proposed animation system consists of three main frameworks, namely, the simulation framework, the AI framework and the modelling and user interface framework (see Figure 4.1).

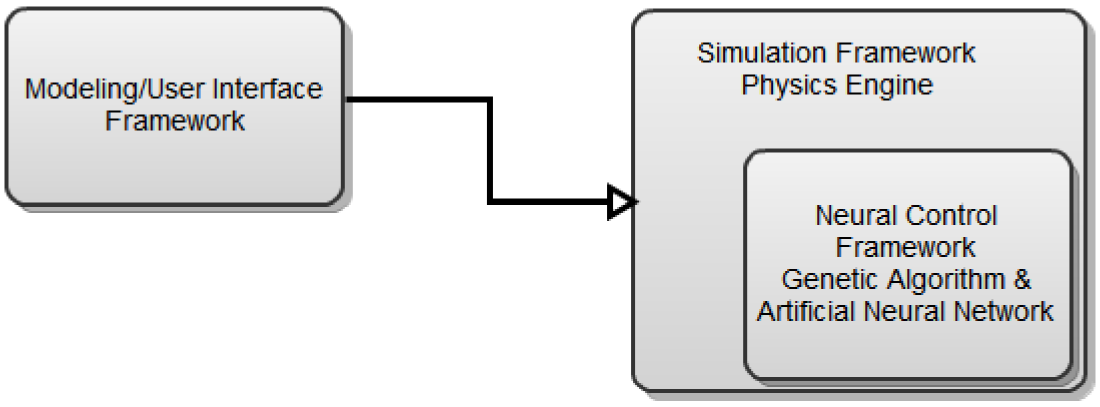

Modeling/User Interface Framework
Simulation Framework Physics Engine
Neural Control Framework Genetic Algorithm & Artificial Neural Network

Figure 4.1: Flow diagram of the system

Development in terms of computer code was done in three stages, namely, Maya related plug-in (custom locator nodes, contexts) development and related MEL scripts for user interfaces and data export functions, PhysX and OpenGL development for the simulation application (constraint specification, muscle generation) and rendering (meshes and rigid bodies) and lastly, the motor control system (an evolutionary neural network using genetic algorithms as a learning model) that controls the muscle activations (see Figure 4.18).

The contribution in this chapter is two fold, which can be termed as follows:

- Theoretical: A simplified real-time friendly representation of a thin muscle structure is formulated that models contact points and force transfer, which also integrates an interface for the artist to specify muscle vectors.
- Developmental: A practical framework that connects together various proven technologies to create a working proof-of-concept.

This chapter examines each of these frameworks and attempts to explain the information flow between them.

4.2 Physics Engines: A Simulation Platform

Physics is a vast subject with a vast array of sub-fields. Physics engines are normally associated with games. And especially for that reason, it is built for efficiency and fast simulation. Physics simulations in games occupied a narrow niche, before the concept of physics engines came into the picture. Simulations were hard coded into the game code. Due to increasing complexity of game design, processing power, game assets and demands in creating complicated effects, game developers found it necessary to create generic code modules that would simulate Newtonian laws in general rather than specific effects of physical laws on game objects. This reusable object oriented code base is collectively called as a *physics engine*.

Most biomechanical and anatomical simulations use customised code bases that are specific to the simulation job at hand (Sueda et al., 2008) (Weinstein et al., 2008). Moreover, these types of scientific simulations, due to the high amount of complex mathematical calculations, are not real-time in the actual sense of the word. Many of them can take anywhere from a few seconds to a few hours to simulate the phenomenon. Hence the secondary goal of this research is to use a fast, efficient and general-purpose game physics engine to create biomechanical simulations and to explore the challenges involved in fine-tuning the engine to the needs of the simulation.

4.2.1 Broad classification of Physics Engines

Physics engines can be broadly classified based on the type of objects being simulated, the method of contact processing and the method of contact resolution (Millington, 2007).

The first classification deals with full rigid body simulation and a system called the mass-aggregate engine (Millington, 2007). Mass-aggregate engines on the other hand, discretize the body into particle masses. A rigid body engine on the other hand takes into account the rotation of bodies (angular velocity of bodies) with the help of a mathematical construct called the *inertia tensor*, allowing an accurate simulation of how solid bodies react to external forces (Millington, 2007). There is also a sub-class of physical simulations called *soft body simulation*, which is distinct in its own right. Soft body simulations deals with the simulation of non-rigid objects like gelatinous masses, fluids, goo like substances etc.

The second classification, deals with contact processing. Contacts form the basic foundation of a physics engine. Contacts are the way forces propagate during various interactions. Contact detection, contact generation and contact processing together form the most exhaustive part of a physics engine. The simplest method is an “iterative approach” (Millington, 2007, p. 6). It is fast and cheap with regards to the processor. Commercial quality physics engines utilize a different approach by processing all contacts simultaneously or by batch processing the contacts (Millington, 2007). This method is called a “Jacobian-based” approach, which calculates the exact interaction between different contacts and calculates a complete set of corrections to all the contacts in a single iteration (Millington, 2007, p. 6). The mathematics involved is often tedious and processor intensive. There are numerous occasions when the system cannot resolve the contacts using the primary collision resolving code and additional code is required to correct the ensuing errors. Another option often used in simulation software is called a "reduced coordinate approach". An entirely new set of motion equations are derived based on which coordinate frame the changes are required. This is a non real-time technique and is seldom used in games where speed is of the essence.

The third classification is based on how the contacts are actually resolved. In the real world, forces are at play with every passing moment of time. And a majority of these forces go unobserved. Forces acting on static objects are a prime example. A glass resting on a table has continuous force acting on it by the table against the force of the gravity, just as the ground is exerting a continuous force on the table opposite to the force of gravity. Simpler physics engines use a variation of this force to keep objects from interpenetrating. These physics engines use force acting for minute fractions of time to keep objects apart. These forces are called "impulses" and hence the physics engines using this technique are called *impulse-based* engines. There are *force-based* engines also. Force-based engines are those that use a Jacobian approach or reduced coordinate approach for contact resolution (Millington, 2007, p. 7). These tend to be more complex than the former type of engines.

4.2.2 The Simulation Loop

Physics deals with quantities that vary with time, like velocities and accelerations. And for that reason, physical simulation often deals with the solving of differential equations. On a computer, differential equations are solved using numerical methods. From a computer code point of view, the simulation loop is the most important part of the physics engine. It is the per-frame processing function that performs the simulation calculations. The simulation loop is the point in the code where numerical solvers are called. There are many numerical solvers at disposal. A comprehensive examination of all the methods is beyond the scope of this thesis and also falls outside the goals of this research. But three methods deserve a mention for the popularity and usage in game physics, namely, Eulers Method, Runge-Kutta Methods and a method that gained fast popularity in the game development arena called Verlet Integration. Refer to Appendix A for a more descriptive piece on these numerical methods.

4.2.3 Choosing the right Physics Engine

Since this research relied heavily on a robust and stable physics engine for implementation of the various concepts described, choosing the correct engine became a highly important task. The following options were available during the initial stages of the research:

(1) Create a custom physics engine

(2) Opt for an open source engine like the Open Dynamics Engine

(3) Opt for commercial level engines like Havok or PhysX from NVIDIA (previously from Ageia)

Serious consideration was given to the first option and the task of creating a custom physics engine was undertaken in the initial stages of the research. It was essential to understand how a physics engine worked in order to expand on it and suit it for the purposes of this research. Study was undertaken to develop a physics engine, which if proved robust, could be used for the simulation tasks necessary. The idea was to create a generic physics engine simulating the basic motion laws and constraints (as that was a secondary goal and instrument of this research) rather than create a specific biomechanical simulator. Care was taken to adhere the design to object oriented programming principles. The exercise proved to be beneficial in unravelling the functional complexities of a physics engine and helped in working around technical roadblocks in the later stages of the research. It also proved instrumental in understanding the complexity and effort required to design a generic physics simulation engine.

The developed physics engine uses Eulers method for numerical integration and supports cuboid, sphere and plane rigid body primitives. The engine used sequential contact resolution. The engine had stability issues with resting contact and was susceptible to vibrations and so objects at rest vibrated. The physics engine developed henceforth was the simplest of its kind and was far from being suitable for a biomechanical simulator.

The second option was to use an open source engine like the Open Dynamics Engine (ODE). ODE was developed by Russell Smith as part of his doctorate research and later made open source (Smith, 1998b). ODE has evolved into a bigger project since then with developers adding features over the years. The biggest attraction was that being released under the open source license, the source code was available for modification. ODE is used by many developers/hobbyists in many projects. ODE is stable and favours speed over accuracy. It is also not multi-threaded (Smith, 1998b).

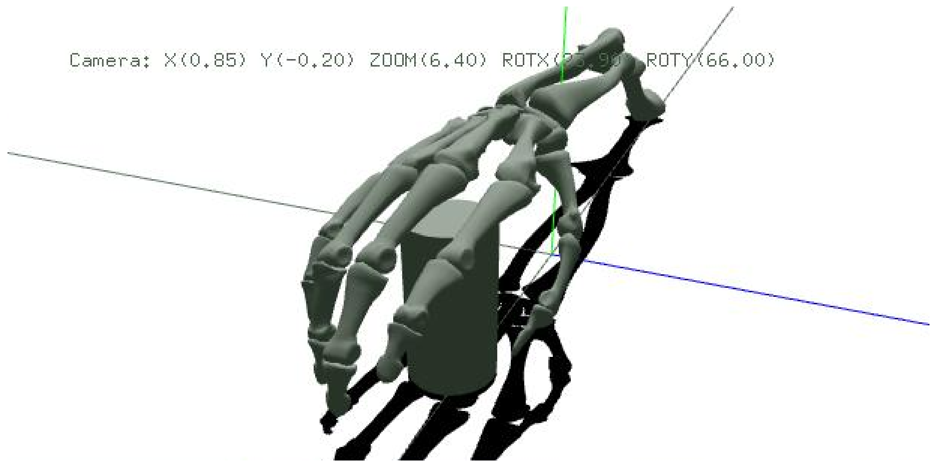
The third option of using a commercial level physics engine proved to be attractive due to several reasons. They have a proven track record in large and successful game projects. Havok is extensively used by a lot of game companies (Havok, 2012). PhysX is also another physics simulation engine that has seen incredible successful in commercial games (NVidia, 2012). At the time of initiating this research, PhysX was the only commercial quality physics engine freely available. PhysX was a product bought by Ageia Technologies from Novodex and was popularly known as Ageia PhysX (Ageia, 2006). Since then, Nvidia acquired Ageia and PhysX is now released under the Nvidia banner. Ageia was the only company to bring forth a microprocessor called a Physics Processing Unit (PPU), which was dedicated to performing physics calculations directly on hardware as required by the PhysX engine (Ageia, 2006). Currently the PPU support is discontinued and hardware physics acceleration is performed directly on the Graphics Processing Unit (GPU). The non-commercial version of PhysX is available for hobbyists and academic research and is free of cost. Havok was available only to game developing companies. After Intel later acquired Havok a free non-commercial version was released for academic and hobbyist purposes (Havok, 2011). In comparing the PhysX and Havok engines, PhysX simulations were far more accurate and stable for the task undertaken, with a comprehensive documentation and sample code. Also the API structuring was clear to understand and better organized. An important aspect of using a commercial level physics engine was conforming to standards allowing easier compatibility with legacy game engine code. Thus the PhysX physics engine was chosen for implementing the ideas in this research.

4.2.4 The Physical Hand Model

The human hand is a highly articulated model having 27 degrees of freedom (Lin et al., 2000). To emulate the motion of the hand, in a physically realistic manner, it is necessary to create a physical model, which can function very close to the biological counterpart.

The 3D model (see Figure 4.14) is polygon-based with every single bone segment modelled as a polygonal mesh. It can be rigid skinned or *parented* to an underlying joint skeleton to be animated using forward or inverse kinematics, if the need arises. The model is the skeletal base for the muscle path laying and for constraint (joint) specification between the bone segments that define a linked structure, specifically, the phalanges and the metacarpals.

As stated in Section 4.2.1, collision detection is by far the most complicated part of physics engine. This part of the physics engine provides maximum load on the processor. And for this reason, real-time game engines provide basic prim-



```
Camera: X(0.85) Y(-0.20) ZOOM(6.40) ROTX(25.90) ROTY(66.00)
```

Figure 4.2: PhysX dynamic model of the hand. Model imported from Maya.

itives like cuboids, spheres, cylinders, capsules as collision objects. These object primitives due to fixed topological definition have pre-calculated inertia tensors and due to simplified geometry allows for linear algebraic intersection tests like the Separation Axis Theorem. There are methods to calculate the inertia tensors of arbitrary meshes. One such algorithm is detailed in Appendix B along with inertia tensors of standard primitives. Arbitrary meshes are therefore simplified to perform collision tests. The most common mechanism used in real-time physics engines for achieving this is through the use of convex hulls. There are various algorithms to calculate the convex hull of a point cloud. The *Quickhull* algorithm and *Andrews* algorithms are two such algorithms (Barber et al., 1996)(Ericson, 2005). Andrews algorithm is a 2D algorithm while the Quickhull algorithm is extended to 3D (Ericson, 2005). PhysX creates convex hulls from a given point cloud. If the point cloud is the vertex array of a polygonal mesh, then it calculates a convex hull for the polygonal mesh. The object exporter from Maya exports the vertices of the bone meshes (see Appendix C). For this purpose, a utility function created specifically for this project performs the convex hull generation operation, which reads the object file and uses the vertex array to generate convex hulls (see Figure 4.3) for the corresponding bone surface meshes.

4.2.4.1 Types of Joints

Game physics engines like Havok and Nvidia PhysX support a wide variety of joint types, which are highly suitable for modelling a physically-based hand avatar with the required DOFs. The PhysX engine employs joints of various types to create articulated rigid bodies (some of which is listed in Table 4.2). The skeletal human hand consists of twenty-seven elements. Fourteen of them are phalanges or finger bones, five are metacarpal or palm bones, and eight are carpals or wrist bones. The

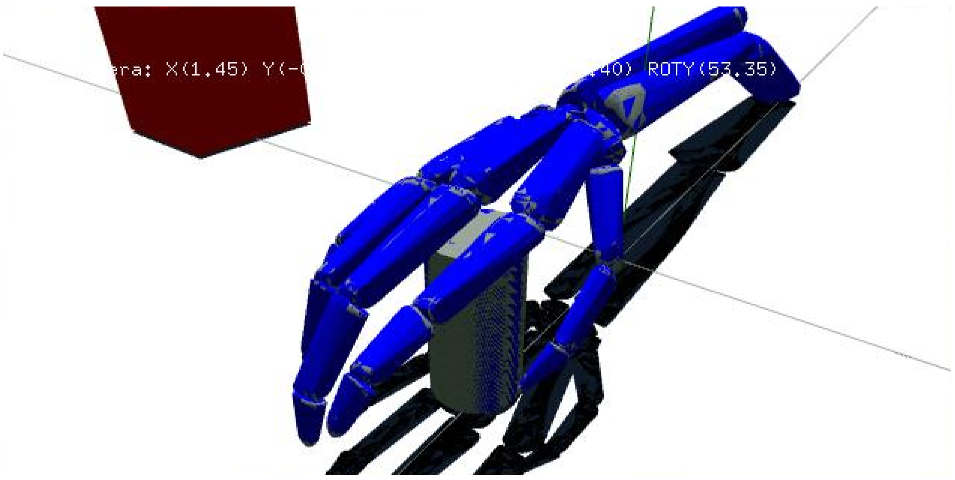


Figure 4.3: The convex hull rigid body overlay of the mesh model. The red cube is another PhysX rigid body. The cylinder is another convex hull rigid body placed for scale reference. The convex hull rigid bodies are the collision objects.

Table 4.1: Anatomical joints in the hand and associated DoFs

Degrees of Freedom (DoFs)	**Joint Type**	**Anatomical Features using the Joint Type**
One	Hinge/Pivot Joints	Phalanges articulating with each other
Two	Bi-axial/Condyloid joints	Phalanges acrticulating with metacarpals
Multiple	Poly-axial joints, saddle joints	Thumb metacarpal with carpal bone (wrist)
One	Plane joints	Carpal bones (wrist)

articulation mechanism for all these bones is a synovial joint (Napier and Tuttle, 1993, p. 45-46). The synovial joint, which has mobility as its primary feature, is the most common joint in the body. There are a total of 15 joints (including the wrist) in each hand. The types of synovial joints are classified according to the type of movement provided (see Table 4.1) (Napier and Tuttle, 1993).

By comparing the anatomical joints with the PhysX joints available, it is possible to create a mapping between the two so as to functionally mirror the joints in the PhysX-based hand model (see Table 4.2). This user interface options for joint types is based on this mapping (see Appendix D). The joints are automatically created (based on export data from Maya) between the rigid bodies (see Figure 4.4).

The 6 DOF joint is an advanced joint model capable of emulating the behaviours of all the other standard PhysX joints (revolute, spherical and fixed joints). It is used in creating realistic rag dolls for games. The key to setting vary-

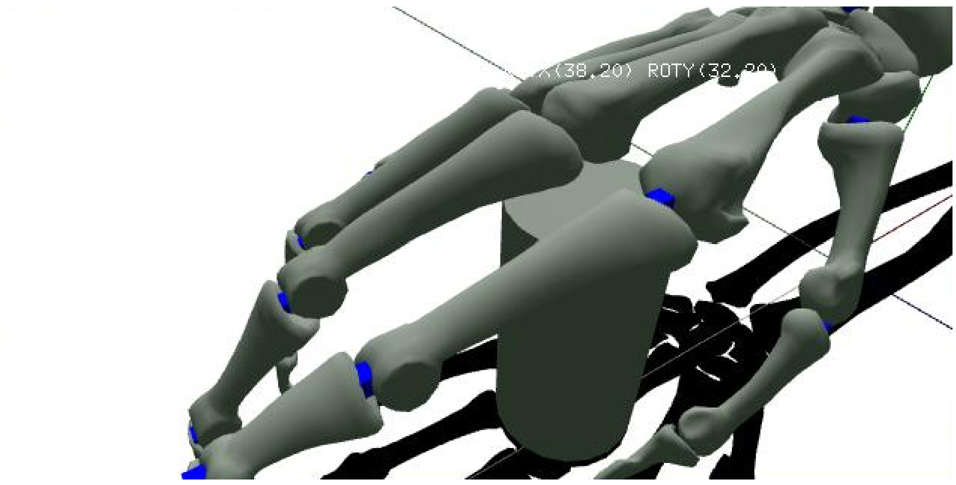


Figure 4.4: The blue cubes depict the joints between the rigid body bones in the hand.

Table 4.2: Joint Mapping

Synovial Joints	**NVIDIA PhysX joints**	**Corresponding PhysX class**
Hinge joint	Revolute joint	NxRevoluteJoint
Bi-axial joint	Spherical joint	NxSphericalJoint
Saddle joint	6 DoF joint	NxD6Joint

ing behaviours is specifying limits for the DoFs. This is made possible through the *NxD6JointDesc* structure and its member variables (Ageia, 2006). The revolute and spherical joints also implements limit structures to specify angle limits to restrict movements. This is explored in the next section.

4.2.4.2 PhysX Joint Limits

Being part of the real world and forming a major anatomical feature, the human hand is bound to certain constraints, which prevent movements that could be perceived as unnatural. Lin et al. (2000) states the importance of constraint modelling, as it helps in the reduction of search spaces (which in the case of hand animation is large considering the large number of the degrees of freedom) and hence crucial in creating realistic animation. At the same time, Lin et al. (2000) states that: “Even though constraints help reduce the size of the search space, too many or too complicated constraints would also add to computational complexity. Which constraints to adopt becomes an important issue”. Lin et al. (2000) divides the constraints into 3 types, namely, Type I definition quote : “limits of finger motions as a result of hand anatomy which is usually referred to as static constraints", Type II definition quote: "limits imposed on joints during motion, which is usually referred to as dynamic constraints in previous work" and Type III definition quote: "constraints which are applied in performing natural motion, which has not yet been explored". Practically, it is Type I and Type II constraints that are important from an implementation point-of-view.

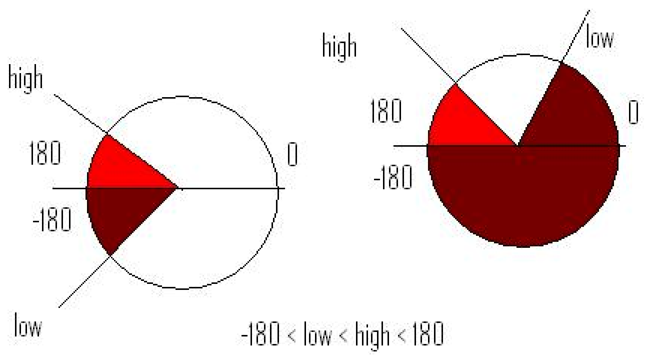


Figure 4.5: Revolute joint limit schematic (Ageia, 2006)

PhysX imposes joint limits through the use of data structures. Figure 4.5 shows the visualisation of the limits of the revolute joint. In Figure 4.5, the white region is the swing limit of the joint. As mentioned in Lin et al. (2000), imposing constraints would reduce the search space of available motion. Using the constraint specifying structures available in the PhysX engine, the physical hand model can be configured to behave like a biological hand. The angular constraints given below are reproduced from Lin et al. (2000).

$$0^\circ \leq \theta_{MCP} \leq 90^\circ \tag{4.1}$$

$$0^\circ \leq \theta_{PIP} \leq 110^\circ \tag{4.2}$$

$$0^\circ \leq \theta_{DIP} \leq 90^\circ \tag{4.3}$$

$$-15^\circ \leq \theta_{MCP\text{-}\mu} \leq 15^\circ \tag{4.4}$$

$$\theta_{MCP\text{-}\mu} = 0 \tag{4.5}$$

$$\theta_{TM\text{-}\mu} = 0 \tag{4.6}$$

$$\theta_{DIP} = \frac{2}{3}\theta_{PIP} \tag{4.7}$$

θ denotes the angle of the various joints on the phalanges and metacarpals and the subscripts MCP, PIP, DIP, μ and TM stands for the metacarpo phalangeal joint, the proximal inter-phalangeal joint, the distal inter-phalangeal joint, the abduction/adduction angle and the trapezio metacarpal joint respectively. Equations 4.2 and 4.3 are the abduction/adduction states for the metacarpo phalangeal joint of the middle finger and the trapezio metacarpal joint. Setting these constraints reduces the DoFs. Equation 4.4 is a Type II constraint imposed on the DIP joint that relates the angle of the DIP to the angle of the PIP, because in order for the DIP joint to bend, the PIP joint must also bend.

4.3 Muscle actuators

Thin muscle structures in the human body help in the transmission of forces to that part of the body where it is impractical for the placement of the full volume of skeletal muscle due to space constraints. Tendons are prime examples of thin muscle structures.

A large part of the motion space, specifically, flexion and extension, of the digits of the hands is attributed to the tendons of the *digitorum superficialis* and *digitorum profundus* (Prives et al., 1989). Muscles of the forearm, namely, *flexor digitorum superficialis* and *flexor digitorum profundus*, transmit contraction forces along their tendons enabling the flexion and extension of the digits of the hand. Tendons are non-rigid, non-elastic and are highly flexible allowing routing over bones and joints and can transmit muscle forces even though there is no direct line of action between the origin and insertion points. Tendon routing is an important factor in the functioning of human hands (Sueda et al., 2008).

4.3.1 Nature of the Problem

In the context of real-time interaction in games, performance is an absolute requirement. Herein lies a two fold problem with respect to physically-based animation in games. They are:

- **Simulation complexity** - Biomechanical simulation dealing with muscles is mathematically complex and extremely processor intensive. Controlling the movement of rigid linkages through muscles requires the calculation of the motion state equations using Newton's Laws of Motion. The motion equations are dependent on the number of muscles and on the forces produced by the muscles in aggregate (Thelen et al., 2003). Simulations like in Sueda et al. (2008), and Lee and Terzopoulos (2006) take anywhere from a few seconds to minutes to solve the complex calculations involved. These long simulation times make it unusable for games which normally run between 30-60 frames per second.

- **Reliance on physics packages** - Middleware physics engines are widely used in current games for a wide range of physical effects, but extremely rarely used in character animation other than ragdolls. This is possibly due to the requirement of physically-based character animation for a fine level force control of rigid body linkages, which generic physics engines do not provide.

Thus a solution that allows for easy application of forces at multiple points on a linkage is required.

The physics engine encapsulates the rigid bodies and the force applicators. The inherent problem arising in creating an entirely new dynamic primitive is that it would be difficult or near impossible to achieve it without core modification of the engine. With the use of a commercially available physics engine, this becomes impractical, as a source level access is granted only to commercially licensed developers. An alternative is to create the required primitive using the higher-level functionalities of the physics engine. It is imperative to look into previously done research on dynamic primitives and anatomical knowledge of muscles in order to create an abstraction that would suit the needs of this implementation.

4.3.2 A Comparison Study of Actuators

Linear force actuators or line of action-based actuators work perfectly on a concave arrangement of rigid links. Concavity in this context means that there has to be a "line of sight" between the insertion and origin points of the muscle on the rigid body. Any break in the line of action due to the presence of joints or other links disrupts the functioning of these types of actuators (see Figure 4.6).

With implicit actuators (Chapter 2, Section 2.2.1), it is possible to generate the kind of motion required as seen in Figure 4.7. The joint connecting the links applies the force. Proportional-Derivative (PD) controllers for dynamic motion have been the subject of various research papers over the years (Van de Panne et al., 1994) (Fattal and Lischinski, 2006) (Van de Panne and Fiume, 1993) (Weinstein et al., 2008). PD control is a feedback system used in physical animation, where feedback is used to achieve the target joint angle. But from a force generation perspective this type of joint-based torque generation has no basis in anatomy and precisely for that reason lacks the kind of flexibility seen in the muscles and

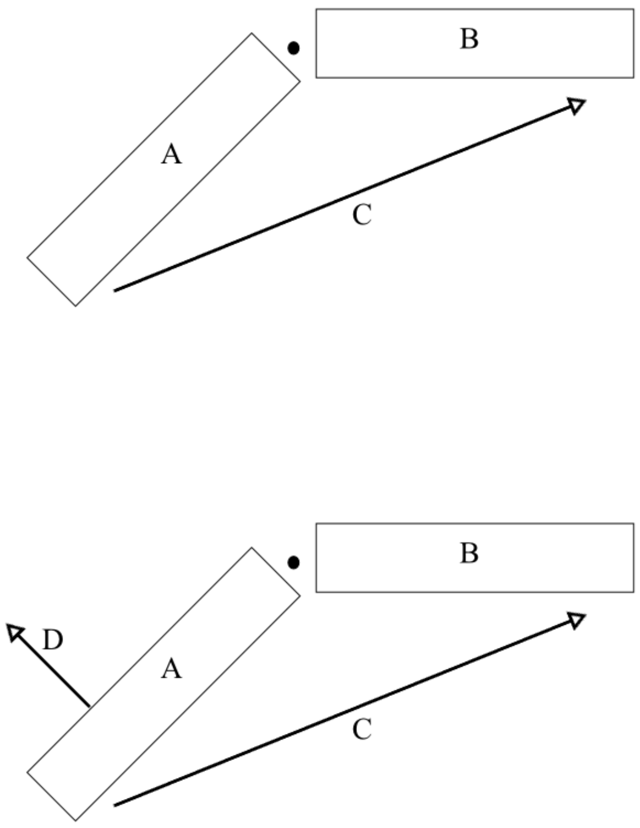


Figure 4.6: : In the top image, the directional line (C) joining the rigid links (A and B) is the line of action force actuator. The arrow shows the direction in which force acts, bringing it closer to the horizontal link on the right. The dot between the links denotes the joint constraint with a single degree of freedom. In the bottom image, the rigid link A has to move away from the horizontal link B, but the line of action actuator is incapable of producing the force to achieve the task (unless its property deviates from anatomical functioning).

tendon systems of the human body. Also PD control normally works best with a single degree of freedom (DOF) (Weinstein et al., 2008).

But while using explicit actuators (Chapter 2, Section 2.2.2), this creates difficulties. An explicit actuator requires a dynamic muscle vector capable of wrapping around joints, so that force application does not require line of action between insertion and origin points of the muscle (see Figure 4.8).

In the case of skeletal muscles, mechanical impedance is defined as the resistance to mechanical motion, due to antagonistic muscle coactivation (Hogan, 1984). This impedance is controlled by the brain (Hogan, 1984). Torsional stiffness required to counteract gravitational destabilization can be attributed to feedback

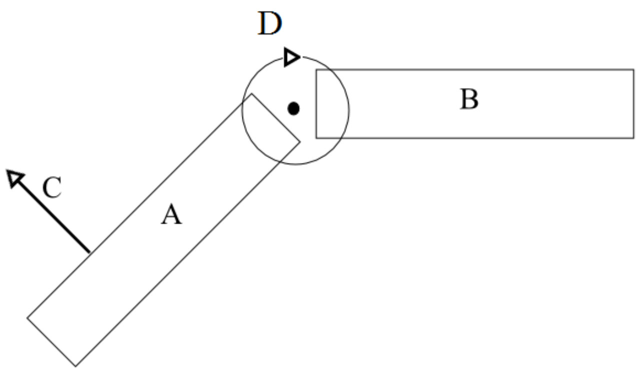


Figure 4.7: The angular force in the given direction D by the joint provides the required movement for the rigid link A to move in the direction C.

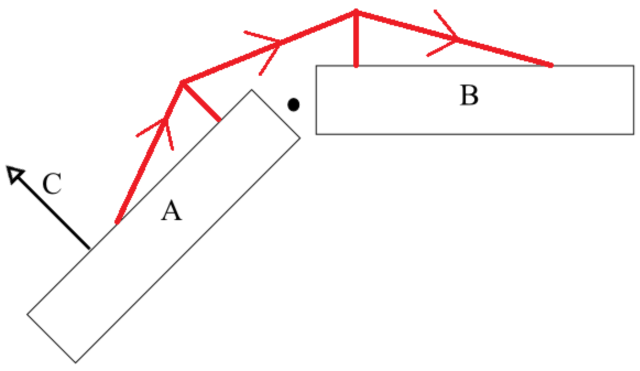


Figure 4.8: An explicit actuator, which wraps around the joint bringing it closer to real-world anatomical mechanics. The pole (in red) features act as *via points*. The red arrows show the direction in which muscle force acts. This is an example of an extensor muscle.

systems in the body or an effect of the coactivation of antagonistic muscles or a combination of both. According to Hogan (1984) there are marked differences in their limitations and this difference would prefer one method to the other. Feedback control suffers from neural transmission delays reducing the gain and lowering stiffness while coactivation suffers from consumption of energy as the opposing muscles expend energy without doing mechanical work. In spite of that, coactivation is not impeded by neural transmission delays Hogan (1984). Hogan (1984) uses dynamic optimisation to predict the neural activations to modulate the impedance via coactivation. A key limitation of motion capture is that the captured motion fails to capture the underlying physical processes that generates the motion. Kry and Pai (2006) tries to solve this limitation and preserving the essence of motion by capturing the compliance of the joints. According to Kry and Pai (2006) compliant joints are essential to model the behaviour of tendons muscles and activation. In the human musculo-skeletal system the compliance is a direct effect of contractile elements of muscle creating spring-like properties. Kry and Pai (2006) models compliance using a set of torsion springs to produce joint torques.

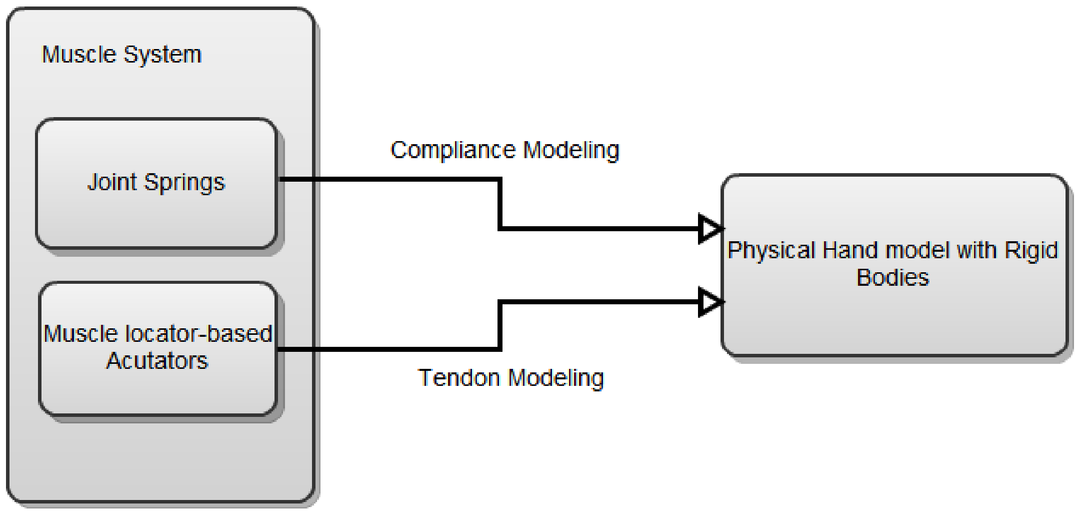

Muscle System
Joint Springs
Muscle locator-based Acutators
Compliance Modeling
Tendon Modeling
Physical Hand model with Rigid Bodies

Figure 4.9: Schematic of the combinatorial muscle system in the physics engine.

Compliance can be modelled in the physics engine through the use of joint springs. The model in this research uses a combinatorial approach to create the muscle system (see Figure 4.9).

4.3.3 Use of Muscle layouts

The muscles and tendon layout of the hand is very complex. Simulating every muscle fibre would be very heavy on the system resources, particularly for performances required in games. Hence it is logical to simplify the layout for physical or virtual functional emulation. Through an examination of the anatomy and muscle layout in a real hand, it is possible to develop a simplified representation of the musculotendon mechanics. Robotics literature also provides functional simplifications that work very well (Pollard and Gilbert, 2002). The origin and insertion points of the muscle models are *points*. In real muscles the origins are *patches* (see Figure 4.10).

So in the model used, the muscle volume is represented using a single strand rather than a set of strands. Using multiple strands would help in getting more

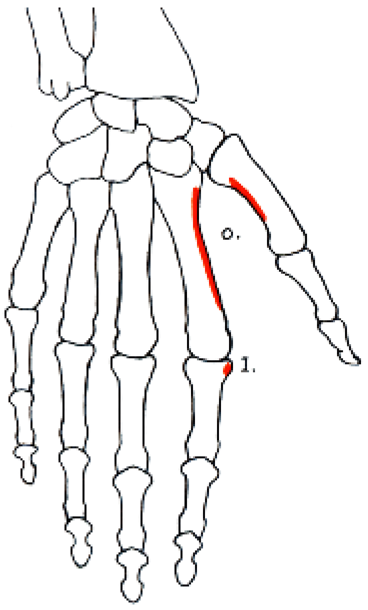


Figure 4.10: The origin (O) and the insertion (I) of the Abductor Indicis that abducts the index finger away from the middle finger. The origin is a set of two patches on the metacarpals of the thumb and index fingers. Image courtesy (Goldfinger, 1991).

accurate muscle behaviour (see Figure 4.11) and also shows a closer approximation of the muscle volume . Internally during muscle contraction, the muscle fibres contract in different amounts at different locations within the muscle volume. Therefore, representing the muscle volume using multiple strands would simulate differential contraction within the strand group.

As seen in Figure 3.4 in Chapter 3, there are muscles that branch into multiple tendons. One way to achieve this effect is to use a dynamic weighting factor (see Figure 4.12). The weighting factor depends on the angle between the primary muscle fibre and the branching fibres that can be described mathematically,

$$W = \left(\frac{1}{Ang}\right) S_f \tag{4.8}$$

$$a_f = W a_o \tag{4.9}$$

where W is the weighting factor, Ang is the angle between the primary muscle fibre and the branching fibres (angle of deviation) and S_f is a scaling factor. The final activation, a_f is a product of the weighting factor and the original activation, a_o. a_o is the activation for the entire muscle which is distributed among the muscle fibres based on the weighting factors.

The branching muscle model is not implemented in the current system.

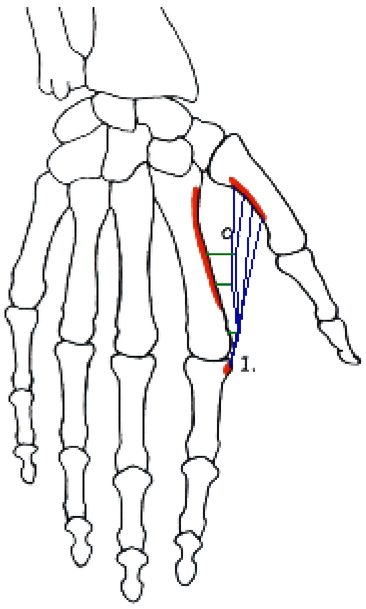


Figure 4.11: Simplified muscle model of the Abductor Indicis, represented by lines (blue) and intermediate origin points/locators (green) between the Origin (O) and Insertion (I). Image from (Goldfinger, 1991) modified by the author.

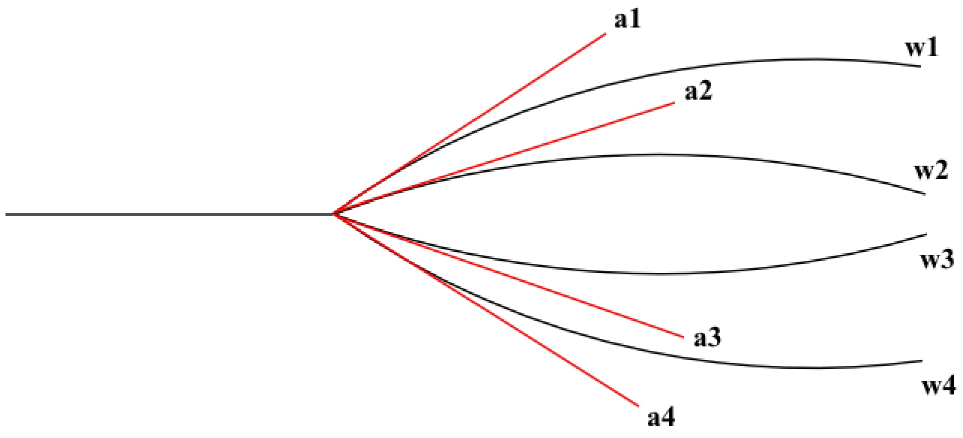


Figure 4.12: A branched muscle model with four branching fibres and associated dynamic weights $w1$, $w2$, $w3$, $w4$ and angles of deviation, $a1$, $a1$, $a3$, $a4$

4.3.4 Force Scaling Muscle Model

Pai and Sueda, in Sueda et al. (2008), have described a simulation technique they named as *strands*. Strands are a form of spline curves that are dynamic and responsive to collision and other external forces. Musculo-tendon dynamics is simulated using strands by incorporating the control points of the strand (being spline curve) into the motion equations. The system in Sueda et al. (2008) uses an algorithmic controller. Though accurate, the results take a few minutes to compute and every new target motion requires computation of activation levels for the strands. In Sueda et al. (2008), the simulator is implemented in Java. Additionally, even though the algorithmic controller is portable to any end simulation engine, the dynamic strand primitive is not easily suitable for conversion to physics engine formats, which means games cannot take advantage of the functionality. In order to achieve game friendliness, a *kinematic tendon* is created.

Tendons, enclosed in the sheaths, are in constant contact with the underlying bones. And the motion of the rigid linkage is governed by these contact interactions. The algorithm for a physics-engine compliant thin structure muscle actuator is based on the above fact. The feature set for the model are as follows:

- The muscle should behave in a manner befitting anatomical muscles. Anatomical muscles produce forces only upon contraction.
- It should have a suitable representation of muscle activation levels for easy activation.
- Robust to function within a real-time simulation environment irrepsective of the number of muscles.

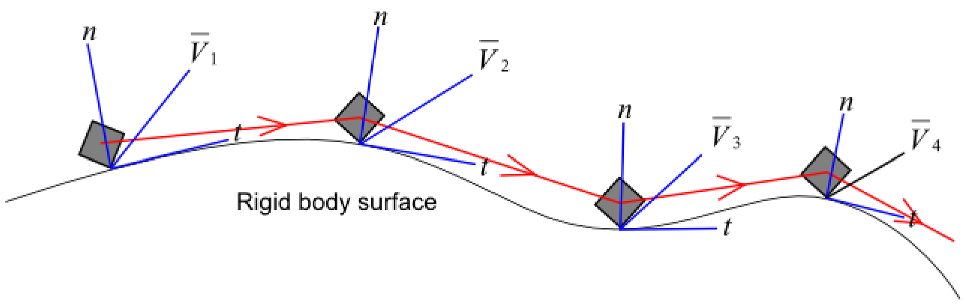


Figure 4.13: Muscle thin structure schematic showing specific contact points, force vectors and components. The grey boxes denote the muscle locators placed on the rigid body bone surface. The muscle locators define the tendon path along which the forces act.

The force application points and direction are calculated from the attachment points, surface normals and the order and arrangement of the attachment points (see Figure 4.13). The path of the tendon structure is defined by the most important component of the muscle system, the muscle locator.

During collision between two objects, forces act on the two objects in two directions: force along the collision normal and force along the opposite direction of approach. The tendons in the human body constantly interact with the bones through contact. And hence there are forces along the normal of contact and along the tendon towards the origin. The model constructed in this project adheres to this principle and applies normal forces at all the muscle locator positions and a tangential force from a muscle locator to the preceding locator all the way to the origin.

Standard vector mathematics decomposes a vector into the normal and tangential components. In Fig. 4.13, the normal and tangential components are shown, along with the actual force vectors. Thus, generalising for all the vectors,

$$\overline{V} = \widehat{t} + \widehat{n} \tag{4.10}$$

These components are essential to model the contact forces prevalent at each muscle locator along the thin structure, on the rigid body linkage.

Deviating from existing controllers that solve the inverse dynamics problem by calculating the torque around the joints, this implementation accepts the magnitude of force as a user-specified quantity, and the direction of the force vector is dependent on the layout of the muscle thin structure on the rigid body linkage. By making force an input quantity, control is not completely abstracted from the animator.

Each muscle strand is associated with an activation level, a, where $0 \leq a \leq 1$

A simplified interpretation of the D'Alembert's Principle states that a set of forces or torques acting on an object can be replaced by the sum of the same forces or torques Millington (2007, p. 78, 220). In a physics engine, at every frame or time step, the forces acting on an object is accumulated and applied to the object at the end of the frame. Every muscle group has a force associated with it. This force is modulated using the activation level of each structure, thereby producing different scaled forces per thin muscle structure. Thus the magnitude of the force on the rigid linkage is distributed among the involved muscles. The total force magnitude $\left|\overline{F}_m\right|$ of the muscle group is defined as,

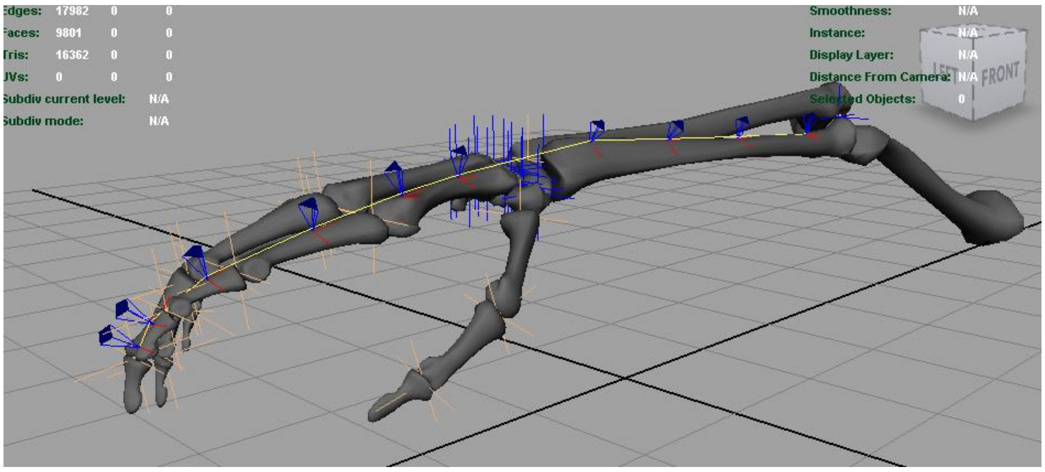
$$\left|\overline{F}_m\right| = a_1F + a_2F + a_3F + a_4F + ...a_nF \tag{4.11}$$

where F is the user defined scalar magnitude of the force input by the user and $a_1, a_2, a_3...a_n$ are the respective activations for each thin muscle structure. Thus, using Equation 4.10, we can represent force scaling as

$$\overline{F}_s = a \left|\overline{F}_m\right| (\hat{n} + \hat{t}) \tag{4.12}$$

where $\overline{F}_s$ is the scaled force vector.

In neuroscience, muscle synergy is a well-accepted theory that helps in solving the redundancy problem in the human motor system (Gazzaniga, 2004). In order to reduce control signals, the central nervous system groups the muscles and dynamically reconfigures the set of muscle groups depending on the task, providing for a wide range of motion behaviours (Gazzaniga, 2004). End-point or end-effector coordinate representation of a motion plan is performed by approximating it as force fields corresponding to muscle synergies (Gazzaniga, 2004). The important fact arising out of this methodology of motor processing, as stated in (Gazzaniga, 2004), is: "When multiple synergies are induced by a pattern of motor commands, their net effect is a field of torque vectors: for each configuration and state of motion of the joints there is one and only one corresponding torque vector." The linear muscle vectors generated from the multiple muscle locators placed on the bone segment act as a linear vector field along which muscle force is applied. Thus, a set of muscle vectors collectively becomes a force field affecting the insertion point of the muscles, which is at the end-effector.



```
dges:     17982   0   0
aces:     9801    0   0
ris:      16362   0   0
Vs:       0       0   0
ubdiv current level:  N/A
ubdiv mode:           N/A
```

```
Smoothness:                N/A
Instance:                  N/A
Display Layer:             N/A
Distance From Camera:      N/A
Selected Objects:          0
```

Figure 4.14: The skeletal hand model with joint locators, muscle locators (blue) and the resulting musculo-tendon unit that extends from the distal phalanx of the forefinger down to the arm bone (yellow). The joint locators are in orange and blue, between the bones of the phalanges and between the carpal bones at the wrist. The blue coloured joint locators represent fixed joints with zero DoF and the orange joint locators represent revolute joints with single DoF.

In Maya, the muscle locators are placed on the bone surface meshes (see Figure 4.14). So when the locators are exported, the world space position of the locator is a point on the mesh. But in the PhysX engine, the meshes are merely render objects. The actual rigid body is the underlying convex mesh (which is generated from the surface mesh) and the surface meshes are bound to the convex rigid bodies via the rigid body transformation matrix (see Figure 4.3). Convex meshes, otherwise known as convex hulls, are simplified polygonal shells that encompass the surface mesh, discarding any concavity (see Section 4.2.4). In the simulation environment, forces are applied directly on the rigid bodies; therefore it is essential to shift the point of locator attachment to the convex rigid body hulls of the corresponding bone surface mesh. A computer graphics method called *ray-casting* can be used to find the updated position of the locator on the convex mesh. The ray would need a point of origin and a direction to propagate, both of which can be calculated easily.

If M_loc is the transformation matrix of the muscle locator, $\overline{V_y}$ is the y-axis vector and F_p is the global placement position (footer) of the muscle locator, then

$$\overline{V_y} = (0.0, y, 0.0) \tag{4.13}$$

$$H_p = M_{loc}\overline{V_y} \tag{4.14}$$

$$\overline{V}_{dir} = H_p - F_p \tag{4.15}$$

where H_p is the calculated head or tip of the muscle locator. Construct a ray from normalized $\overline{V}_{dir}$ and locator placement position, F_p. Since the convex hull is created from the underlying mesh and it always encloses the mesh, the ray would need to always radiate towards the exterior of the mesh. Thus,

$$P_{new} = F_p + P_{rayhit} \tag{4.16}$$

Here P_{new} is the offset position of the muscle locator and P_{rayhit} is the ray intersection point on the convex hull.

The ray-intersection algorithm is a standard algorithm in geometry. PhysX provides the functionality to create and shoot rays into a simulation scene and collect information regarding the point of contact of the ray and any objects that lie in its path.

The direction and the movement are dependent on the muscle attachment points, and the arrangement of the thin fibre (see Figure 4.16). Multiple muscles are entirely possible and the type of movement, in which case, would be governed by the amount of activation of each muscle and its layout on the rigid body linkage. The linear piece-wise locator-based muscle system is configurable so that each segment can have a different activation (if the need arises). Also, by increasing

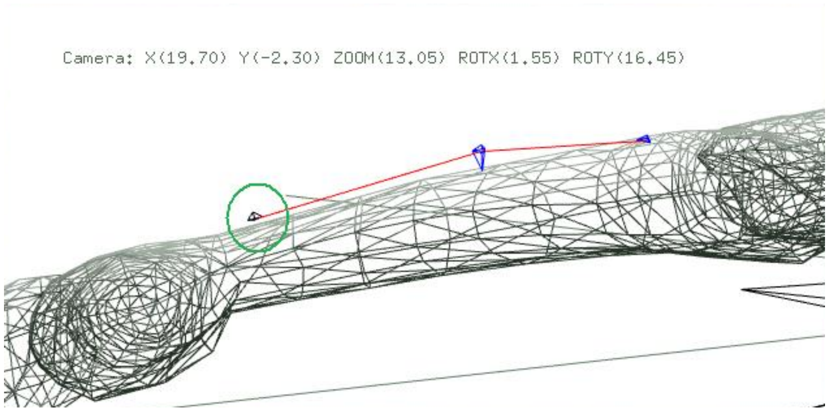

Camera: X(19.70) Y(-2.30) ZOOM(13.05) ROTX(1.55) ROTY(16.45)

Figure 4.15: Due to the offset, the locator (encircled in green) appears to be floating above the bone surface mesh (shown in wire frame here). The offset places them on the convex hull (not shown here) that encompasses the mesh. The remaining two locators are also floating above the bone surface mesh, but the viewing angle makes it appear otherwise.

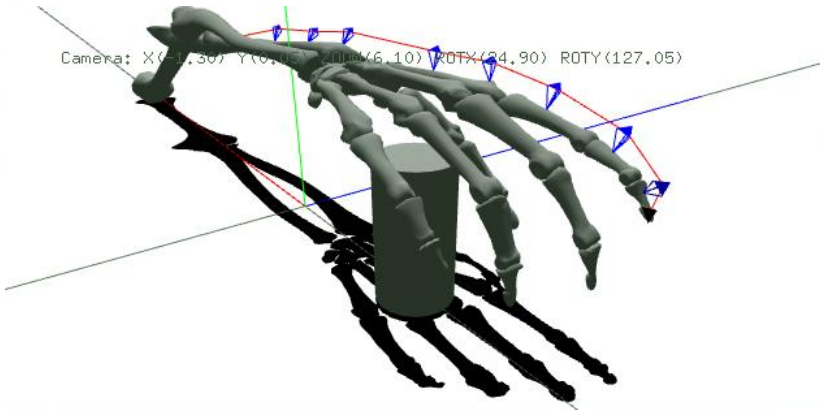


Figure 4.16: The exported muscle locators from Maya and the muscle fibre visualization in the PhysX simulation application.

the number of intermediate muscle locators, the muscle would attain a curve like form very much like the tendons. Functionally, this would mean additional force application points to produce smoother motion at the expense of simulation speed.

4.4 Neural Control Structures

The human body has an extremely complex neural system, which controls the many muscles that performs different actions depending on tasks at hand. This neural system is at a nascent state during human infancy. Neural development is crucial in all stages of a human life, especially where cognition and motor skills are concerned. The exact process of neural development is a popular subject of research. Self-organizing systems form a part of this ongoing research (Camazine, 2003). What is self-organization? According to (Camazine, 2003):

> "Self-organization is a process whereby pattern at the global level of a system emerges solely from interactions among the lower-level components of the system. The rules specifying the interactions among the systems components are executed using only local information, without reference to the global pattern."

The last part in the quote highlights the reason why self-organization is an important factor in neurobiology. The human brain constitutes about 10 billion neurons with approximately 60 trillion synapses (neural connections) (Negnevitsky, 2005, p. 166). This connectivity attributes to the enormous complexity of the brain, but taken singularly the biological computing element is just a *neuron*, which has certain behavioural properties. Each neuron processes only its own local information. But the dense connections between the large amount of neurons gives rise to self-organizing behaviour and contributes to the evolution of temporally bound systems like pattern generators (for locomotion) , cognition and behaviour (Kelso, 1995, p. 239-246). Camazine (2003) defines *emergence* as "a process by which a system of interacting elements acquires qualitatively new pattern and structure that cannot be understood simply as the superposition of the individual contributions". Even though self-organizing patterns can "emerge" in internal neurology, emergence is often governed by external stimulus (from the containing environment) and tendency for goal acquisition (Konczak, 2004). This is clearly apparent in human infants. Voluntary motor behaviour does not emerge by itself, but arises as a consequence of constant interaction of the infant with its environment (Konczak, 2004). There is a high dependency on sensory inputs for advancing neural development, though it varies with organisms. Due to high neural complexity in humans and certain primates, sensory stimulation is imperative for activating the neural development processes which in turn leads to the development of motor control (Konczak, 2004). This is evident from comparisons of different organisms at infant stages to humans at their infancy. The human brain has various neurological sub-systems to take care of the many functions like motor capability, memory, vision, and most importantly the high level of intelligence and association. Due to high complexity, humans are rarely born with locomotion skills or speech capability. All these higher order skills are acquired skills, taking a certain amount of training time leading to neural development that perfects motor behaviour (Konczak, 2004). Animals have rudimentary brains with base level intelligence with the rest of the processing space utilised for survival qualities like motor skills and sensory skills. All these skills are more or less in-built. Hence in the case of humans, environmental feedback plays a more important role in learning motor behaviour than in other organisms, largely due to the need for neural development for learning motor skills (Konczak, 2004). Through an inference from observational analysis, it is safe to say that pre-conceived motion templates handed down generations form the basis of human training for tool manipulation and other motor skills. Examples range from holding a pencil or tea cup to attaining skill in dancing or sports.

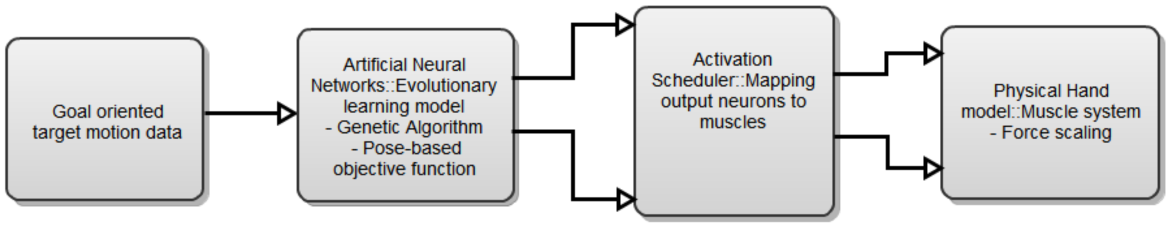

Goal oriented target motion data
Artificial Neural Networks::Evolutionary learning model - Genetic Algorithm - Pose-based objective function
Activation Scheduler::Mapping output neurons to muscles
Physical Hand model::Muscle system - Force scaling

Figure 4.17: Control flow schematic of the neural motor controller

Targeted motion, in physical terms requires precise control. In the human body, this is achieved by a complex motor neuron system. Temporally coordinated action, like locomotion and precision tasks, is generated by sophisticated timing and activation mechanisms. The timing of activation is an important aspect in performing a specific movement. Neural signals (activating muscle synergies) are transmitted by the central nervous system (CNS) to the corresponding muscle units, which contract creating the required motion, thereby converting a perceived goal into mechanical motion. In the system described as part of this thesis, muscle actuators are the motion generators, similar in function to the biological muscles. The muscle actuator system is described in section 4.3.4.

In order to create goal-oriented motion, the muscle actuator system requires a control system to be plugged into. The neural control system consists of two main sub-components, namely, the artificial neural network comprising of “motor” neurons that uses an evolutionary learning model and an activation scheduler, which acts as the bridge between the neurons and the physics engine-based muscle system (See Figure 4.17). Below is given the class diagram of the controller architecture.

4.4.1 Artificial Neural Network

There are many successful implementations of neural networks for a wide variety of tasks ranging from locomotion to neck and arm movements (Sims, 1994)(Afshar and Matsuoka, 2004)(Oztop et al., 2004)(Lee and Terzopoulos, 2006). Karl Sims in Sims (1994) uses genetic algorithms to evolve neural network models that output muscle forces and also to evolve the morphology of the digital organisms. Sims’s research is important in the artificial life field and also has implication in evolu-

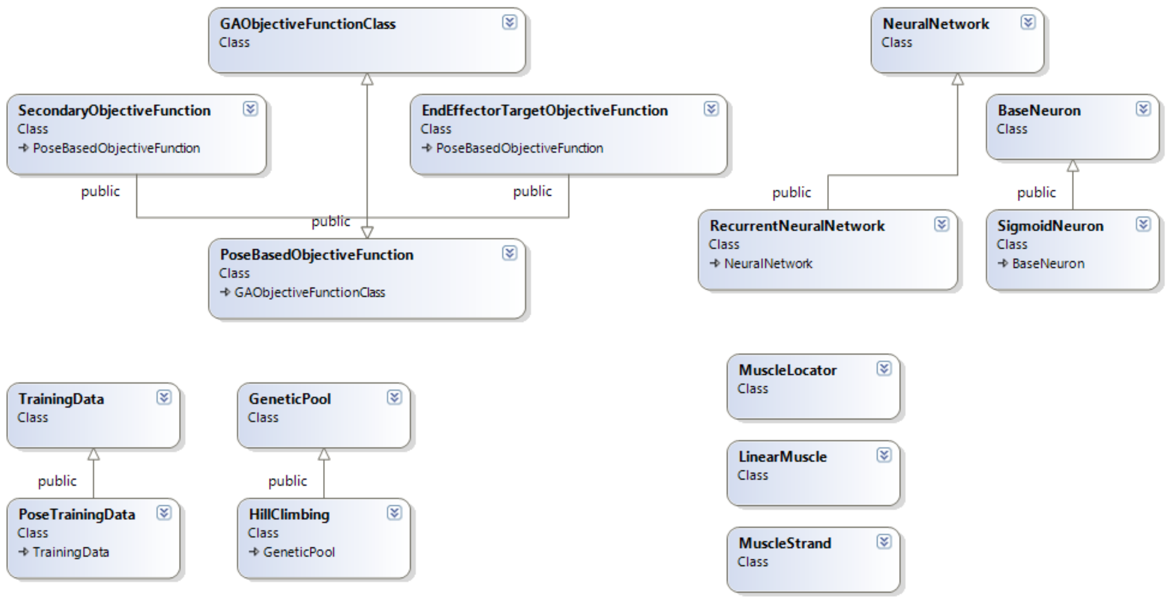


Figure 4.18: Class diagram of the various object classes in the virtual motor control system.

tionary biology and species propagation. But from an actual production pipeline point of view, Sims work is of little value. Character designs in production are driven by the needs of the story and the manner of movement of the characters are locked down only after a thorough iterative study that examines the anatomy and limitations of the character morphology. Also, there is no autonomous behaviour attributed to the character motion. They are animated specific to a shot or generalised within limits in the case of games. This research differentiates itself from Sims's work in the following ways:

- The goal of Sims's work is to evolve artificial life creatures that commute in a simulated environment, whereas this research concentrates on creating neural controllers capable of mimicking captured motion.
- Morphology of the character is fixed.
- Objective functions are of higher complexity than the simple target and distance objective of Sims's work.
- Architecture of the neural network is fixed.
- A detailed muscle system in contrast to Sims's simple PD actuators.

Robotics has many applications that use neural networks to learn adaptive control. Holmes et al. (2006) is a comprehensive paper that explores the dynamics of legged locomotion and the neural bases in the sensing, planning and learning evolved in the process. Central pattern generators (CPG), which are key components in locomotion generation, is also neural based, as is proprioception. As mentioned in the previous section, a complex interaction between the environment and the physiology of the organism is essential for goal directed active force generation and control. A large part of the control is through feedback and the ability of the neural models of the motor system to adapt through synaptic learning (Holmes et al., 2006).

A light weight neural network library catered specifically for the task at hand was developed. A few key points that were considered during the creation of the library were as follows:

- Easy-to-use, lightweight library and rapid prototyping of a neural network using minimal configuration steps is a primary aim.
- Use object oriented programming and design principles to embed scalable architecture.
- The library should provide support for loading and saving a neural network from and to a file.
- A generic learning model class with which different models can be implemented depending on requirements.
- Plugs in to the physical simulation framework smoothly.

There are three main base class structures in the network model that encapsulates the neural network properties. They are: *BaseNeuron*, *Layer* and *Network*. Object oriented design allows for extending the functionality of the base classes by having a class derive from the base class and have extra functions written into the derived classes. The base classes have functions or methods whose functionality can be overwritten in the derived classes. These functions are called as *virtual functions*. The BaseNeuron class simulates the single computing unit in a neural network, the neuron and has a re-programmable virtual activation function. The default behaviour of the function is the step function that output only binary values. Muscle activations are analogue in nature with a range of floating point values. The function suited for analogue output is the *sigmoid* function. The Layer base class represents the different layers in a neural network. After a layer is initialised, neurons can be added to it. Though representing the network internally with a layer abstraction brings with it a storage redundancy, the layer class helps in bringing neural network concepts to the library in a structured manner. For efficiency, it is entirely possible to bypass the layer storage structure and perform direct neural connections, but for this project, the structured approach is utilised. Adding the layer structure also brings a complexity (from an implementation point of view) to the neuron connection representation. Finally, the Network class organises all the sub-components into the semblance of a networked model. The Network class has a *Connect* method, which is a virtual function that can be overridden in the derived classes. The Connect method specifies how the neurons between two consecutive layers connect to each other. It basically defines the network architecture or topology.

4.4.2 Activation Scheduler

Muscle contraction occurs due to the action of motor nerve impulses on muscle fibres and generally, as a principle, one motor neuron cell activates many muscle fibres. The collection of muscle fibres innervated is called as a *motor unit* (Chaitow and DeLany, 2000, p. 33). The number of muscle fibres innervated can range from 10 to the several hundreds (like in the major limbs used for locomotion and grasping, and other manipulation tasks) (Chaitow and DeLany, 2000, p. 33). So it can be seen that a single nerve cell has a varying influence on the muscle.

Similarly in the virtual neural control system, the activation scheduler maintains the relation between the neural network output and the simulated "muscle fibres" (locator-based muscles). It acts as an interface (between the artificial neural network and the muscles) that transmits the activations produced by the network to the muscles. The scheduler was created with future expansion in mind in order to support muscle fibre-based volume muscles.

4.5 Modelling and User Interface

The difficulty in getting accurate results in a graphical simulation is compounded by two factors, namely, proper models and the lack of a good user interface (UI) to integrate the modelling and simulation framework. Schniderman in Schneiderman and Plaisant (1998) defines three principles or pillars of user interface design, namely:

(1) Guideline documents

(2) User interface software tools

(3) Expert review and usability testing

These pillars of design are followed in the design of user interfaces of most conventional software, Autodesk Maya included. Therefore extending the functionality for the Maya workflow, using Mayas own intrinsic API, ensures the usage of Mayas design features.

It is always wise to keep in mind that no matter how intuitive the UI design of the animation system, the artist ultimately using it is restricted to work on a two-dimensional screen interacting with another two-dimensional hardware tool like the mouse that lacks the ability to determine its spatial position in 3D space. The loss in dimensionality can be compensated through the use of unconventional hardware interfaces like those used in animatronic effects. This would allow the a closer mapping between the interface and 3D space and allow the artist a better navigation of 3D space. But such interfaces are experimental and not cost effective due to electronic component overheads and training time. And hence mainstream animation houses refrain from using such interfaces.

This makes it imperative for functional friendliness of software user interfaces. Even complex interfaces like in Pixologic's Zbrush (3D sculpting software) provide advanced intuitiveness and configurability, but still allow the artist to create a wide range of effects with the default settings (Pixologic, 2010). Similarly, with the exception of interfaces using hardware add-ons (such as motion detection sensors or Microsoft Kinect-based software hacks), a good animation software interface should work within the restriction posed by the most conventional hardware interaction tools the mouse and the keyboard.

Good user interface design is highly subjective to the software in question and the intended use of the software and often requires multiple iterations to lock down on a practical design. For 3D animation, it often means creating a mapping interface between 3D transforms and 2D user controls, eg: facial animation rigs and the slider controls that drive the rig. Full body animation, on the other hand, requires animation rigs that exist in the same 3D space as the character.

Simulation set-up is made easier if the actual graphical elements involved in the simulation are constructed in the proper manner and also properly spatially arranged in the simulation environment. Alternatively the graphical elements could be created in a professional 3D modelling and animation package like Maya, as exemplified in Tsang et al. (2005), Sueda et al. (2008). In the case of human musculo-skeletal modelling and animation described in this thesis, this attributes to a good skeletal model, easy interface to specify constraints in the model and an intuitive interface to layout muscle paths. Providing the artist with the feature to lay muscle paths, is crucial for the simulation aspects because the animation is dependent on the muscle vectors.

Autodesk Maya is becoming the de-facto graphics industry standard. It is used primarily in the visual effects industry and is capable of dealing with heavy duty graphical requirements in modelling, animation, simulation and rendering.

Therefore, as a manner of future proofing the concept to fit into conventional animation pipelines in the games or visual effects industry, Maya was chosen as an integral part of the system framework.

It was decided during the initial stages of the research to include Maya as the primary test bed for tool development. But as the research evolved and the system re-designed due to technical reasons, the decision was made to separate simulation and content generation. Thus Maya became primarily a content generation platform for the external simulation application.

Maya implements a graphical three-dimensional place-holder element called *locator* (see Figure 4.19) for various tasks. The standard locators shipped with Maya are typically visualised in Maya like a three-dimensional cross hair which can be placed anywhere in 3D space. Locators basically specify a location in space. But they can also be used intuitively for manipulation of objects in 3D space.

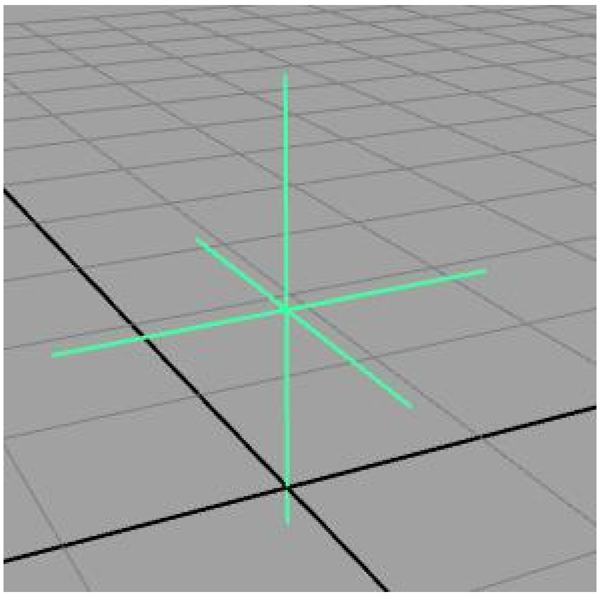


Figure 4.19: Maya locator in the 3D view port.

Custom locators, manipulators and interactive contexts can be easily developed through the Maya C++ API.

The ideal interface for laying muscle paths and origin and insertion points on skeletal segments is direct interaction with the segments in the 3D view port where the artist would use the mouse to click on the surface of the segments to specify the attachment points (see Figure 4.20).

As given in Section 2.2.2 of Chapter 2, the obstacle set method in muscle simulation uses *via points* placed on rigid bones and obstacle objects to denote the wrapping of muscles around joints. The inclusion of obstacle objects to hold the via points and facilitate collision detection burdens the processor. The alternative approach taken in this research is to use specialised locators having some intrinsic

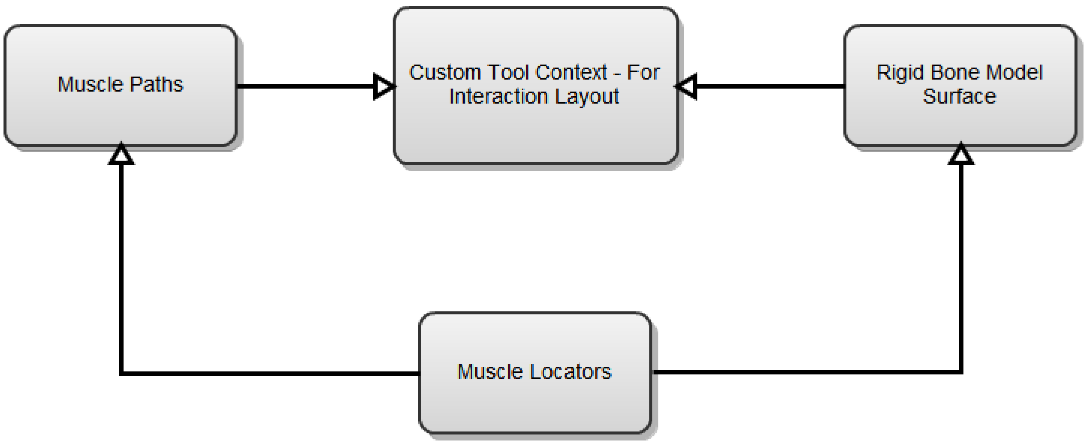


Figure 4.20: Schematic showing custom context and interactive elements tied to it.

properties that would specify points of contact of muscular tendons and thereby define the path of the tendons (see Figure 4.21).

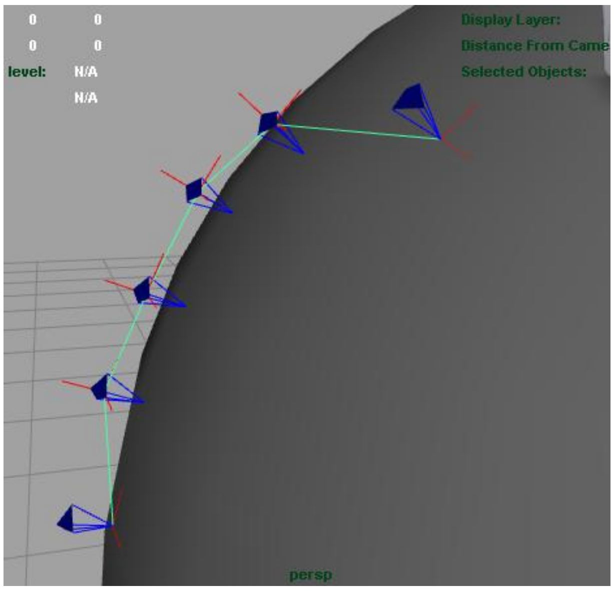


Figure 4.21: The muscle locators in blue and the path defined by them in green, laid on the surface of a sphere.

Each muscle locator along with the attributes (see Table 4.3) is created automatically as the user clicks the mouse cursor at the point of insertion on the bone surface. The attributes are essential in encapsulating certain muscle properties, which are essential in recreating the physical muscle during simulation. The context responds to user clicks only when a bone mesh surface is selected. Using a Maya C++ API method, the context fires a ray into the scene from the mouse pointer and finds the point of intersection of the ray with the mesh surface. The normal at that point is retrieved and using the transformation matrix of the

Table 4.3: Muscle locator attributes

Attribute Name	**Description**
Locator	1 or 0 depending on whether to use the head or the foot of the locator as the control point of a linear curve.
Height	A floating point value representing the height of the locator. A single muscle group can have a constant height for all its locators or have varying height.
Locator Type	0,1 or 2 depending on whether the locator is an origin, insertion or intermediate point.
Muscle Grp Id	A unique identifier for the muscle group or tendon.
Muscle order	A consecutive integer number to denote the locator order in the muscle group.
Normal X	Normal at the locators position X component
Normal Y	Y component
Normal Z	Z component
Measure X	Stores the actual calculated point (head or foot) after performing the necessary transforms X component.
Measure Y	Y component
Measure Z	Z component

bone mesh, the locator is transformed and drawn at the point of intersection. Each subsequent muscle locator in the muscle group is stored into a list. The user action of switching off the context utilises the stored locators to act as input to control points of a linear Nurbs curve. The generated curve is the visualization of the muscle tendon path (Figure 4.21). The muscle tendon path is important because each linear segment of the path (the part between two consecutive muscle locators) is used to construct the vector linkage chain in the simulation application. The vector chain forms the guideline for the application of contraction forces for the muscle. The height attribute of the muscle locator is of extreme importance as it affects the amount of moment arm or torque generation in the simulated muscle. It also affects on how effectively the muscle segment wraps over a joint.

The displacement of the tendon from the joint center is essential to increase the moment arm. The sesamoid bones near the joints achieve this in the human musculo-skeletal system (Gray, 2006). They protect the tendons as well as help in wrapping the tendons over the joints. In obstacle set methods, joint wrapping is done with the help of geometric collision primitives, whereas in the system under discussion, this is achieved through the height attribute of the muscle locator.

Once the muscle locators and path are laid out on the hand model and the joint locators placed between bone segments (see Figure 4.14), exporting process is performed “under the hood” using custom MEL scripts (see Figure 4.22).

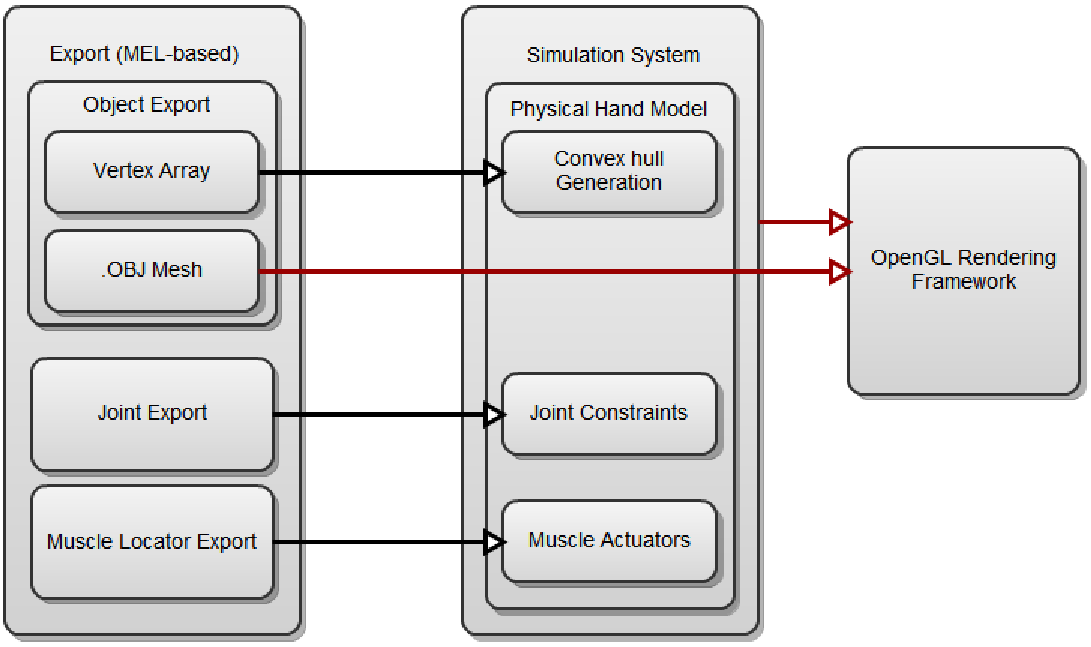


Figure 4.22: Schematic that shows the export system and how it ties in with the simulation and rendering framework.

The screen shots of the MEL interfaces are given in Appendix D. The exporting functions write out the data required in a text format, which loosely follows an XML style (details of which are given in Appendix C). In order to maintain co-relation between the models generated in Maya and the PhysX, the Maya names of the models and joint locators are set into the information structures of the physics rigid bodies and joints. So there is direct one-to-one mapping between the names in Maya and the names in PhysX. The mapping is important for the working of the system. The design choice of a locator-based muscle system with a point and click interface that interacts directly with 3D meshes allows for easily editable muscle system. Laying out the muscle on a surface is as easy as selecting the muscle locator context and clicking and creating the locators directly on the mesh.

4.6 Summary

The system outlined in this chapter combines three different technologies to create a usable interface for generalised physical motion using controllers based on evolutionary neural networks, physically-based muscle actuators that actually generate the motion and motion capture training data to teach the controller. The muscle locator-based linear piece-wise tendon unit applies forces according to the layout of the tendon unit on the rigid body skeleton.

Physical system modelling requires intuitive modelling tools for the end user to set up the simulation environment. The most difficult aspect of the system is the controller design due to lack of a priori information related to the number of neurons and network architecture (details of which are given in Chapter 5).

5

SIMULATING BIOLOGICAL COMPUTING ELEMENTS IN DYNAMICAL SYSTEMS

5.1 Introduction

The previous chapter examined the physics engine compliant musculotendon model that generates skeletal motion. But in order to generate goal oriented motion, the muscles must activate in a specific sequence . The redundancy problem in the musculo-skeletal system makes this coordination sequence a complex problem. Since the research emphasises on a real-time application of biomechanical motion generation using muscles and also due to computational overhead, deterministic solutions are not ideal for the required performance. Instead, a machine learning approach based on evolutionary neural networks is adopted and understanding the control system in the human body is crucial to simulate a similar system.

In the human body, motion is the result of coordinated effort of various synergistic activations of muscle groups. Similarly, in order to generate simulated motion, the simulated muscles have to activate in a coordinated fashion. In humans, the motor learning process involved in training this coordination is complex and motion behaviours that are instinctive collect over many generations genetically even contributing to biological *fitness* (Shadmehr and Wise, 2005). It is also dependent on key factors like prediction, error signals signifying prediction accuracy and also error correction (Shadmehr and Wise, 2005). The motor control system of the central nervous system (CNS) that helps produce coordinated motion continues to be a field of active research. The results or interpretations of the research can be categorised under the following two approaches of predicting muscle activations, namely,

- **Algorithmic or mathematical techniques**: Optimization techniques are used to calculate muscle activations for goal-oriented articulate movements. These techniques try to solve the inverse dynamics and forward dynamics problems and are part of biomechanical muscle simulation. These mathematical techniques often integrate biomechanical simulation of muscles along with activation coordination. Examples include biomechanical research works in Tsang et al. (2005), Thelen et al. (2003), Sueda et al. (2008), Sifakis et al. (2005).

- **Machine learning techniques**: Previous research on generating muscle activations used neural networks with gradient learning models that used electromyographic (EMG) data recorded for calculating the error (Murai et al., 2008), (Prentice et al., 1998), (Adamczyk and Crago, 2000).

This chapter delves deeper into the hybrid machine learning techniques implemented as part of the system developed for the research and also the nature and construction of the objective functions that enable the machine learning algorithms to narrow down the correct solution. Section 5.2 examines the movement encoding in the human central nervous system and how the CNS uses basis functions to generalize and adapt to motion. Section 5.3 presents the algorithm used in creating the control system. Section 5.4 is an introduction to genetic algorithms and the various objective functions developed. The genetic algorithm is used as a learning model by evolving the weights of the artificial neural network. Section 5.5 looks in-depth into the core of the neural controller. The section also examines the time series prediction technique and evolutionary neural networks. Finally Section 5.7 provides the standalone test results of the objective functions used by the genetic algorithm.

5.2 Movement Encoding in the Central Nervous System

The problem of mapping a motion shape to an internal muscle activation state is complex mathematically in the deterministic domain. It requires finding aggregate joint torques taking into account all the muscles that are responsible. The complexity arises due to the force computation for the number of muscles participating in the desired motion. There is another influencing factor which is the force generation properties of the muscle which is dependent on the muscle length and the rate of change of length. Also, in the CNS, there is no inverse mapping between an end-point force (muscle contractions) and the neural firing in the primary motor cortex that is responsible for motor control (Todorov, 2000). Research suggests that there are numerous neurons firing for the generation of single force outcome. This redundancy causes a many-to-one mapping which makes the determination of properties at the individual neural cell, very difficult or impossible (Todorov, 2000).

Experimental results in Shadmehr and Wise (2005, p. 436) confirms that training the motor system to perform motions result in adaptation that is encoded in long term memory. Basically, in order to perform motor learning and adapt to changes, the CNS relies on error correction based on previous motions (Shadmehr and Wise, 2005). The CNS requires an internal model for the dynamics involved in controlling limbs. The internal model is actually a mapping between estimates of limb configuration and forces required for the movement.

The variables for muscle control is embedded within the neurons. This idea of using a population of neurons to encode required variables is called *population coding* which is a widely used model in neural computation (Shadmehr and Wise, 2005). A population code generally maps an input variable to an output variable providing an estimate of how the input relates to the output. This is called *identity mapping*. So, each neuron should encode limb configuration and velocity within a function g_i with a force vector f_i. A neuron discharge rate for the duration of a movement in a particular direction is defined as a function of the direction of movement, $g_i(\phi)$. This function describes the neuron's *directional tuning curve*. The internal models use certain *basis functions* to represent the dynamics and the shape of the bases indicate the pattern of generalization. The tuning curve of the participating neurons become the basis functions and through a combination of these functions any linear and non-linear function can be approximated (Shadmehr and Wise, 2005). This basically describes the process happening in an artificial neural network also where the activation functions of each neuron is a basis function and in combination with other neurons, the network is able to approximate any linear or non-linear function. The neural discharge in an artificial neural network is dependent on the type of activation function used. The neuron cells in the CNS have perturbations or noise during discharge which has to be accounted for in the discharge function. Thus, in (Shadmehr and Wise, 2005),

$$r_i{}^{(n)} = g_i(\phi) + \eta_i{}^{(n)} \tag{5.1}$$

where ${r_i}^{(n)}$ is the neural discharge function, $g_i(\phi)$ is the function of the direction of movement ϕ and ${\eta_i}^{(n)}$ is the noise term.

A particular limb configuration is used to map to a particular velocity field defining a particular motion. The cerebellum, the parietal cortex and the motor cortex broadly encode limb configurations using approximate linear functions (Shadmehr and Wise, 2005). Understanding these functions are key in unlocking how an adaptive system stores the maps it learns and also in learning the patterns of generalization used in the adaptation.

A motion of rigid linkage can be defined as a sequence of configuration states in time, which can be expressed mathematically as an equation of forces for achieving those configurations. Thus,

$$\hat{\tau}^{(n)}(t) = \sum_i w_i g_i(\theta^{(n)}(t), \dot{\theta}^{(n)}(t)) + w_i \eta_i^{(n)}(t) \tag{5.2}$$

where n is a given trial or iteration, θ and $\dot{\theta}$ denote the pose configuration and velocity of the linkages, w_i is the preferred force or direction vector. In the artificial neural network controller used in this research, the direction of movement is input as time sequence of pose vectors. Velocity encoding is not directly performed in this network. But in Shadmehr and Wise (2005), it was shown that velocity encoding works better than direction encoding.

As (Shadmehr and Wise, 2005) states, relating tuning curves to generalization patterns of motor learning requires the following set of assumptions:

- Temporal states of the limb during a movement is simplified to direction of movement.
- Temporal activity (discharge) of the neural component during the movement is denoted by the average discharge.
- Noise is absent.

Taking the above assumptions into consideration, (Shadmehr and Wise, 2005) presents the final derived force function as

$$\widehat{\tau}^{(n+1)} = \widehat{\tau}^{(n)} g(\phi^{(n)}) + \alpha g(\phi^{(n)})^T g(\phi^{(n+1)}) \widetilde{\tau}^{(n)} \tag{5.3}$$

where $\widehat{\tau}^{(n+1)}$ is the force at a future time $n+1$, $\widehat{\tau}^{(n)}$ is the current force, $g(\phi^{(n)})$ is the function of the current direction of motion, $g(\phi^{(n+1)})$ is the function of the direction of motion at time $n+1$, $\widetilde{\tau}^{(n)}$ is the error between the estimated and actual forces. For a complete derivation, refer to Shadmehr and Wise (2005, p. 415).

And the generalisation function, $b(\phi^{(n)}, \phi^{(n+1)})$ is the second term and given as

$$b(\phi^{(n)}, \phi^{(n+1)}) = \alpha g(\phi^{(n)})^T g(\phi^{(n+1)})\widetilde{\tau}^{(n)} \tag{5.4}$$

It is a function of the current and subsequent direction of movement, $\phi^{(n)}$, $\phi^{(n+1)}$. α is a scaling factor for scaling the error, $\widetilde{\tau}^{(n)}$. The error is the differential between estimated torques/forces and actual torques/forces. Equation 5.4 says that the error in n trials times the tuning function (directional tuning curve) describes the force predictions of the internal model from movement n to movement $n + 1$ (Shadmehr and Wise, 2005, p. 416). The generalization function is the key to adapting the dynamics to new movements. In the absence of quantifiable data comparison between the inputs and the outputs, stochastic optimization method is combined with an artificial neural network to accomplish motor learning. The artificial neural network (with its generalization capability) encapsulates the generalization function.

There are a few advantages in using machine learning methods to solve the control problem over deterministic solutions, especially when the target application is games.

- Using machine learning methods that adopt parallel processing nodes, like the artificial neurons of ANNs, with simple computational functions.
- Machine learning methods using stochastic optimization are efficient in exploring vast search spaces that result from a large number of control variables. The muscle activation space is explored indirectly using GAs.
- Generalising ability of machine learning methods allow for re-usability and solving for unknown inputs.

The presumption that paves way for the method detailed in this thesis is that medical data relating to muscle activation is not practical in a computer graphics pipeline used in the industry. Recording the EMG data often requires specialised equipment and the expertise of medically trained professionals which is not feasible economically. The computer graphics, animation and games industries have access to low-cost alternatives like motion capture facilities which is widely used in generating quick and realistic animation for characters. So the aim is to design a system that can generate muscle activations from recorded surface motion and to find out if the system can generalise to an unknown motion sample. Since there is no direct correlation between input pose data (from motion capture) and output activations, the only method to explore the space of activations is through the means of stochastic optimisation methods.

5.3 The Algorithm

The functional mapping between a pose state and corresponding muscle forces (denoted by activations) that produce the pose is derived through training an evolutionary neural network. The algorithm for evolving the weights of the neural network is given as follows:

```
(1) Initialize GA. Create initial weight population vector.
(2) Loop through each member of the population vector.
    (i)   Set weights into the Artificial Neural Network.
    (ii)  Activate the network.
    (iii) Set output activations to the physical muscles on the rigid
          linkage.
    (iv)  Run simulation.
    (v)   Calculate fitness.
(3) If exit criteria is reached (optimized) then exit loop and save
    network configuration.
(4) Perform Genetic Optimization.
(5) Goto Step 2.
```

There are two sub-algorithms implemented as part of the genetic optimization.

```
(1) Calculate selection probability ratio for each population member.
  (i) Sum the fitness of all the members in the population.
  (ii) Calculate the ratio for each member ie the percentage of the
       member fitness in the fitness landscape for the population.
(2) Perform Selection using the ratio, Crossover (single point
    crossover) and Mutation based on mutation probability.
(3) Replace the old population with the new one.
```

Genetic algorithms usually represent the problem space as binary strings. But real-valued chromosomes are also used as is in this project. The genetic operations are emulating natural evolution and exist to provide variations and to guarantee crossover of genetic information in subsequent populations. In a binary representation, changing even a single bit can produce drastic changes in the fitness landscape. This can be achieved mostly using selection and crossover alone with a low probability of mutation. But in real-valued chromosomes, variety is better introduced through mutation, as crossover alone would be just recycling the existing population without variation. The following algorithm is for creating mutation depending on a variance vector that mutates based on the performance (Fogel, 2006). This is known as *Gaussian* mutation, explained in sub-section 5.5.3.

```
(1) For every gene in the chromosome create a random number between 0
    and 1.
(2) For every chromosome in the population create a random number
    between 0 and 1.
(3) Calculate new variance vector for the population member.
(4) Mutate the member based on the variance vector using a Gaussian
    scheme.
(5) Replace the old population with the new mutated population.
```

5.4 Evolution, Genes and Chromosomes: the Genetic Algorithm

To optimize is to find the best solution to a certain designated problem. Optimization methods can be broadly classified into two: deterministic and stochastic methods (Garcia et al., 2006). Deterministic methods have a direct calculation approach, usually calculating the derivatives of a function or approximations (Garcia et al., 2006). They are very useful in solving problems where functional representations are known a priori. Stochastic methods implement an "oriented random" method. They are successful in tackling highly non-linear problems where the problems do not have good functional approximations. These methods require the change of the input parameters (usually a number of them) depending on the output of the functional representation of the objective of the solution. The objective function is the only guide stochastic methods use to converge on a solution. The situations where the stochastic methods work well are in large solution spaces and these methods gained popularity mainly because of the increasing power of the computers (Garcia et al., 2006). This section and the following sections and sub-sections provide a brief overview of the various parts of a Genetic Algorithm. The understanding of the biological world changed after Charles Darwin presented his theory of evolution, which comprised of the neo-Darwinian paradigm (which in turn is actually a sum total of the concept of genetics and theory of natural selection) (Negnevitsky, 2005). Neo-Darwinism lays its foundation on the processes of reproduction, mutation, competition and selection (Negnevitsky, 2005). What forms the objective function in nature? One key factor is the environment. Once environmental factors lay the foundation, natural selection takes over. The population's ability to counter environmental disasters and survive is called evolutionary fitness (Negnevitsky, 2005). Genetic Algorithm (GA) is an evolutionary approach to machine learning and is based on computational models of neo-Darwinism.

Evolutionary computation is an encompassing term that includes GAs along with evolution strategies and genetic programming. All these techniques simulate evolution by using the processes of selection, mutation and reproduction (Negnevitsky, 2005). Put forward by John Holland in 1975, GAs operates on a population of artificial chromosomes by selectively reproducing the chromosomes of individuals with higher performance and applying random changes. It is an iterative process spanning over several cycles or generations. Each chromosome is decoded, one at a time, its fitness evaluated and three genetic operators - *selection*, *crossover* and *mutation* - are applied to generate a new population. The following two subsections explore the genetic operators used and modified as a part of this project and also explores the design of the most crucial part of a genetic algorithm that helps in solution convergence - the objective function.

5.4.1 Methods of Representation

In nature the inherited genetic code is called the *genotype* and the resulting organism is called the *phenotype* (Nolfi and Parisi, 2002). The genotype is handed down each subsequent generation and embodies the parent genetic information and a new phenotype is created from those instructions. This is known as the *genotype-to-phenotype* mapping (Nolfi and Parisi, 2002). Adaptation requires evolvability which strongly depends on how the genetic variation maps onto phenotypic variation representation. This is known as the *representation problem* (Nolfi and Parisi, 2002). The two mechanisms that allow GAs to find solutions are *encoding* and *evaluation* (Negnevitsky, 2005, p. 221). The solution space needs to be encoded in a form the GA can process and apply evolutionary operations on it. An artificial chromosome (genotype) is a string that encodes the characteristics of an individual (phenotype) (Negnevitsky, 2005). The string may encode in binary representation the value of a variable of a function that must be optimized. Chromosome representation ultimately depends on the target type for optimization. Thus, the chromosome, for instance, might encode the connection weights of a neural network instead of function variables. In order to use GA as a learning model for the artificial neural network, the network needs to be encoded as chromosomes. In the case of a learning model, the architecture of the network is fixed and synaptic weights are evolved using the GA. So the synaptic weights are encoded as chromosomes.

5.4.2 Genetic Operators Used

Nature selects the fittest individual to propagate a species. Often adaptation to environmental conditions occurs through successive generations and subtle changes in the organism's genetic make up. Similarly GAs use selection, reproduction and mutation in order to create the fittest solutions for a given problem.

5.4.2.1 Selection

Selecting the right individuals out of a population of individuals is of utmost importance to guarantee that the fitness is transferred to the next generation.

There are various types of selection in the genetic algorithm like *tournament selection*, *rank selection* and *steady state selection*. The selection used in this project is called *roulette wheel selection*. It is a commonly used selection routine (Goldberg, 1989).

In roulette wheel selection, a circle is subdivided into as many sections as there are chromosomes in a population (see Figure 5.1).

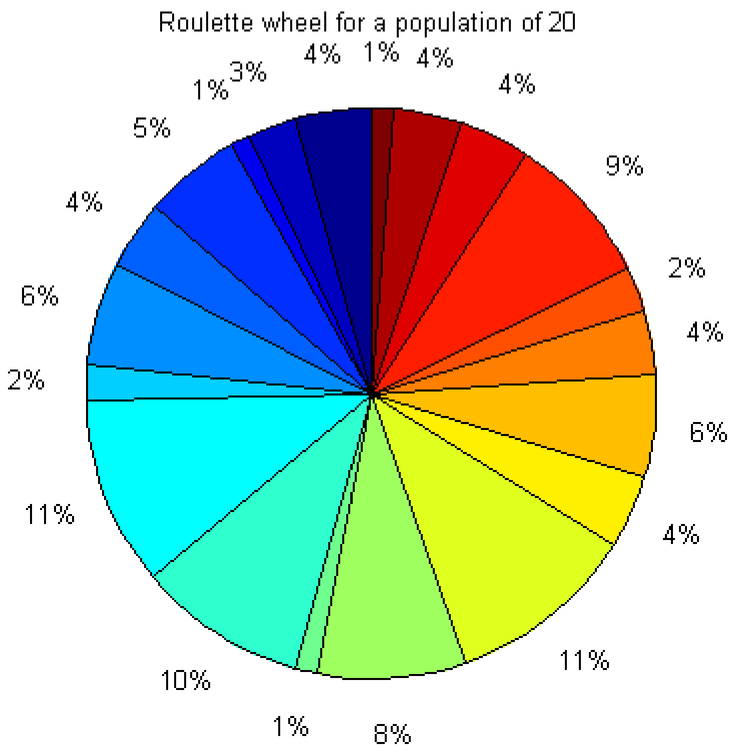


Figure 5.1: A Roulette wheel visualisation for the neural network given in Figure 5.8. Population size is 20 while chromosome size is 182.

The size of each section depends on the fitness of each chromosome and is measured as a percentage. The spinning of the wheel is simulated by generating a random number between 1 and 100 or a preferred upper limit. Then,

$$R = \left(\frac{rand()}{RAND\_MAX}\right) * 100 \tag{5.5}$$

Equation 5.5 normalizes the random number generated and converts it into a percentage

$$h = l + P_{r(i)} \tag{5.6}$$

The selection probability ratio, $P_{r(i)}$ is calculated for each individual chromosome in the population. The size of the subdivision in the roulette wheel is proportional to the selection probability ratio. The chromosome which satisfies the condition, $l \leq R \leq h$, is selected for crossover.

$$P_{r(i)} = \frac{F_i}{\sum\limits_{i=1}^{N} F_i} * 100 \tag{5.7}$$

where F_i is the fitness per chromosome in a population of size N.

There are two situations when the roulette wheel selection breaks. They are:

- When all the individuals in the population have the same fitness, then all the individuals have the same probability of being selected.
- When one or two individuals have a very high fitness value than the rest of the population, then the fitness spans for those individuals occupy a larger area of the wheel. Thus with every spin of the wheel, those same individuals have a higher probability of being selected.

  Scaling the fitness value prior to selection can help in resolving the above issues.

5.4.2.2 Crossover

Crossover is the process of transferring genetic material from both parents to the offspring. A pair of chromosomes is selected for crossover. Crossover is achieved by splitting the parent chromosomes at a random location and swapping the parts to generate two child chromosomes which form the part of the new population (see Figure 5.2). Single point crossover is implemented for this project. The other types of crossover are *two point crossover* and *multi-point crossover*. There are also crossover operations that use multiple parents.

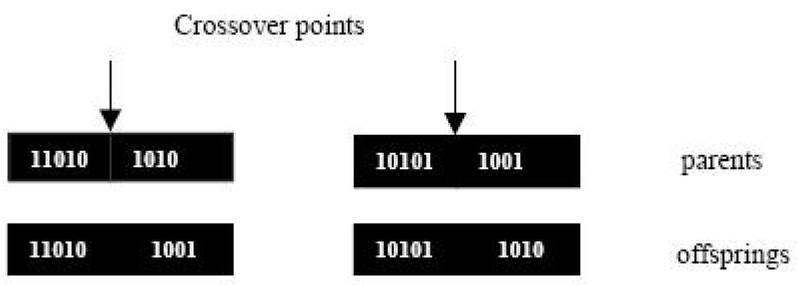


Figure 5.2: Genetic crossover between parent chromosomes to generate offsprings

5.4.2.3 Mutation

If genetic diversity does not exist in subsequent generations, then the GA can get trapped in a local optimum. Crossover and selection can stagnate at some point. This is avoided through the use of mutation. Simple mutation is performed by randomly selecting a gene in the chromosome and then modifying the value. Excessive usage of mutation can be detrimental to convergence. Hence normally mutation is assigned a very low probability.

Since this project uses GA as a learning model for an ANN, real-valued chromosomes are used. As mentioned earlier, normal binary string chromosomes can change drastically even when a single bit is flipped. For real value representations, the value of a selected location is substituted with another value randomly extracted from the same range, or is incremented by a small number randomly generated from a distribution centred on/around zero. But in real-valued chromosomes change is gradual. In such a case, GAs can use mutation operator alone without crossover (see sub-section 5.5.3).

Even though, standard GA can be adapted to work with evolving real-valued vectors, it was discovered during implementation that unlike binary operators which depended on crossover to provide variety, real-valued chromosomes preserved variety better through mutation than crossover alone (with a low mutation probability).

Previous research, like in Reil and Husbands (2002) and Watson et al. (1998), suggests that cross-over lacks efficiency in real-valued evolutionary neural network problems involving dynamic learning. So mutation along with an *elitist* selection, was preferred over crossover. Elitist selection was implemented in this research to ensure that high standards are met while filtering the networks from the population. A specified percentage of the fittest individuals from a population are selected and transferred without mutation to the next generation. The remaining individuals are mutated and then evaluated. This ensures that fittest individuals are always maintained and are replaced only if a new individual with a higher fitness evolves from the remaining percentage of the population.

The evolutionary computation method of using the mutation operator alone is termed as an *evolution strategy* (Negnevitsky, 2005). Mutating all the genes in a chromosome mimics nature. Natural selection usually acts on a collection of genes instead of a singular gene and the fitness of an organism can be governed by either a single gene characteristic or multiple gene characteristics (Negnevitsky, 2005). The original evolution strategy was proposed by two students in 1963, Ingo Rechenberg and Hans-Paul Schwefel and was a single parent-single offspring strategy. Rechenberg later modified the algorithm to introduce a self-adaptive parameter that controlled the distribution of new trials or offspring from each parent (Fogel, 2006).

In order to achieve an evolution strategy based search, the mutation operator implemented is called *Gaussian mutation.* In Gaussian mutation, the mutation is applied to all the individuals of the population based on a *variance* or *perturbation* vector.

$$\sigma_i^{'} = \sigma_i \cdot e^{(\tau^{'} \cdot N(0,1) + \tau \cdot N_i(0,1))} \tag{5.8}$$

where σ is the variance vector, $N(0,1)$ is a standard Gaussian random variable and $N_i(0,1)$ is an independent standard Gaussian random variable for every individual in the population $(i = 1, 2, 3......n)$.

$$x_i^{'} = x_i + N(0, \sigma_i^{'}) \tag{5.9}$$

where x_i is the gene per chromosome.

$$\tau = \frac{1}{\sqrt{2\sqrt{n}}} \tag{5.10}$$

$$\tau^{'} = \frac{1}{\sqrt{2n}} \tag{5.11}$$

τ and $\tau^{'}$ affect global and chromosome mutation step sizes.

5.4.3 The Objective Function

In GAs, an objective function or fitness function govern the fitness of a chromosome. The GA uses the measure of fitness of the chromosome to aid in reproduction (genetic operator) to propagate the fitter genes. The fitness function plays a crucial role in the convergence of the system to the solution because it is the only deciding factor in selecting a chromosome for an evolutionary process producing the next generation of population. Fitness function formulation is often difficult and not always possible in some situations. This is generally due to complexity of the problem under consideration or due to the large interdependence of variables in the problem or due to the inseparability of the low-level implementation from the behaviours exhibited. Fitness functions are often closely dependent on the end-behaviour targeted. This dependency makes it extremely difficult to create generalised fitness functions that suit multiple purposes. The fitness functions implemented here are part of a class of dynamic learning fitness functions that are often used in Evolutionary Robotics (ER) and Artificial Life (AL) and are called *behavioural fitness functions* (Nelson et al., 2009). The *behavioural terms* are mostly extracted from the result of motion generated in a specific time period in a dynamic environment. These type of fitness functions are used in ER to measure how a robot is behaving through the entire training period rather than examining the end result. There are *aggregate terms* that measure the end result. An example of an aggregate term is used in this thesis is detailed in the section 5.4.3.7. The objective functions developed in this research are explained in the following sections.

5.4.3.1 Pose-based Objective Function

The sample spaces provided consists of three dimensional motion samples, which acts as a guideline for evolution to follow. Mathematically, the guideline is expressed in the form of an optimizing function, which can provide a measure of the fitness of the network chromosome that generated the motion. The differing degrees of freedom of the kinematic linkages make the error calculation a multi-dimensional problem. By discretizing the continuous looped motion sample, it is possible to create specific instances of stationary motion manifolds or key frames. Common objective functions found in evolutionary robotics and artificial life use distance functions to teach the robot or the artificial organism to navigate its surroundings. Similarly, the pose-based objective function uses the distance between orientation of target and source vectors generated from the 3D pose in the motion sample and the generated pose of the dynamic skeletal bodies.

5.4.3.2 A Special Case Vector Field

The research under discussion relies on certain factors to simplify the creation of an objective function for the GA. The factors are as given below:

(1) The hand is an articulate organ. Topologically it is the same in all humans, irrespective of the race, unless severe congenital or accidental deformation occurs.

(2) The number of skeletal segments remains constant. Also skeletal landmarks (anatomical points that are in the same position over a range of human bodies) exist, which can be used to transform the hand to a neutral alignment.

(3) The similarity metric is used only in the training phase of then network.

(4) The motion samples exist in the Autodesk FBX format, which is a 3D skeletal-based format.

(5) The phalanges of the hand are rotation invariant along their central parallel axis. This is important as it allows the representation of the skeleton orientation as a collection of directional vector chains.

(6) The magnitude of the phalanx vector is not taken into consideration, as lengths of phalanges changes with different hands, but orientation can be the same. For instance, an adult hand and a child hand vary in their respective dimensions, but they can mimic each other maintaining the same orientation of the skeletal segments.

(7) Unlike standard vector fields where vectors exist independently in space, the vectors in this case are arranged in a chain, in order to function as a pose descriptor. Hence a more accurate terminology is vector chain, though in the course of this chapter both terms are used interchangeably.

Taking into consideration the above factors, a simplified vector chain can be extracted from the 3D skeletal key frame. Since the hand (or any articulate body) is segmented with joints constraining the rigid bodies (phalanges), the result is a vector chain and not just any arbitrary set of vectors. It has to be noted that for the purposes of this research, true shape matching is not required since it is a pose identification using the vector chain generated by the skeleton of the articulated body.

The use of vector chains for the comparison of a linked structure also ensures that the corresponding vectors are given as inputs also. There are 19 vectors for the human hand. The musculotendon layout of the fingers are identical. Therefore, by using individual neural networks for each finger simplifies network training. A single trained finger network can act as the template for all the other networks. Even though a one-to-one mapping exists between the two vector sets, representing the source and target articulate bodies, the respective links can vary in length. This case is shown in Section 5.7. Here, only the pose is considered and not structural properties like size. This allows for the possibility of rigging a completely non-human proportioned hand and have it trained using the motion capture data of a human hand, provided the musculature and skeletal structure remains the same. Performing local space vector chain comparison allows for differing global space orientations for the source (motion capture skeletal chain) and target (rigid body linkage) chains. A vector chain representation allows for segmentation of the chain prior to training. The user has the choice of using only a sub-chain from the original chain, for training.

5.4.3.3 Directional Cosines of the Vector Field

Vectors are mathematical structures that encompass a direction in space with a magnitude. They are used to represent any quantity, which has both a direction and a magnitude. They have proved to be an invaluable tool in the physical and mathematical sciences to such an extent that an entire branch of mathematics is devoted to them called vector analysis.

If $\overline{V}$ denotes a vector in Cartesian space,

$$\overline{V} = x\widehat{i} + y\widehat{j} + z\widehat{k} \tag{5.12}$$

The direction cosines, as the name suggests, are the cosines of the angle subtended by the projection of the vector on the three coordinate axes. They are usually denoted by the symbols, α, β and γ. So mathematically,

$$\alpha = \cos(\theta) = \frac{x}{\left|\overline{V}\right|} = \frac{x}{\sqrt{x^2 + y^2 + z^2}} \tag{5.13}$$

$$\beta = \cos(\omega) = \frac{y}{\left|\overline{V}\right|} = \frac{y}{\sqrt{x^2 + y^2 + z^2}} \tag{5.14}$$

$$\gamma = \cos(\phi) = \frac{z}{\left|\overline{V}\right|} = \frac{z}{\sqrt{x^2 + y^2 + z^2}} \tag{5.15}$$

where θ, ω and ϕ are the angles subtended by the vector on x, y and z axes, respectively. $\left|\overline{V}\right|$ denotes the magnitude of the vector.

5.4.3.4 Application to the problem

The directional cosines are unique to each vector in the vector field. In order to compare the orientation of two vectors, assuming one of them as a source and the other as a target, it is a good measure to and the difference between its directional cosines. The difference is a measure of the quantity by which the orientation of a vector differs from the other. So by constructing vector chains from each frame of generated skeletal motion, it is possible to compare it to a stored motion clip.

If $\overline{V}_1$ and $\overline{V}_2$ are two vectors, with directional cosines, $d_1 = (\alpha_1, \beta_1, \gamma_1)$ and $d_2 = (\alpha_2, \beta_2, \gamma_2)$ respectively, then,

$$d_2 - d_1 = (\alpha_2, \beta_2, \gamma_2) - (\alpha_1, \beta_1, \gamma_1) \tag{5.16}$$

Using equations 5.13, 5.14 and 5.15 in their native form creates a problem. Due to lesser variations in $cos(\theta)$ as it tends to zero and to retrieve a better estimate of the variation, it is beneficial to work with the angles, θ, ω and ϕ (Van Verth and Bishop, 2008, p. 49). So the above three equations can be re-written as

$$\theta = \cos^{-1}(\alpha) \tag{5.17}$$

$$\omega = \cos^{-1}(\beta) \tag{5.18}$$

$$\phi = \cos^{-1}(\gamma) \tag{5.19}$$

Thus Equation 5.16 can be changed to the difference of the corresponding angles,

$$(\theta_2, \omega_2, \phi_2) - (\theta_1, \omega_1, \phi_1) = (\varepsilon_\theta, \varepsilon_\omega, \varepsilon_\phi) \tag{5.20}$$

where ε_θ, ε_ω, ε_ϕ denotes the respective small differentials. Hence if there are n vectors in the vector chain F_1 , denoted by $(\overline{V}_{F11}, \overline{V}_{F12}, \overline{V}_{F13}..........\overline{V}_{F1n})$, the corresponding angles are (θ_{F11}, θ_{F12}, θ_{F13},.......θ_{F1n}, ω_{F11}, ω_{F12}, ω_{F13},......ω_{F1n}, ϕ_{F11}, ϕ_{F12}, ϕ_{F13},......ϕ_{F1n}). Similarly it can be defined for another vector chain F_2 consisting of p vectors with corresponding angles, (θ_{F21}, $\theta F22$, θ_{F23},.......θ_{F2p}, ω_{F21}, ω_{F22}, ω_{F23},......ω_{F2p}, ϕ_{F21}, ϕ_{F22}, ϕ_{F23},......ϕ_{F2p}) The constraint imposed on the system of vector chain is that n=p, so that there is a one-to-one mapping between the vectors in the two chains. Subtracting the corresponding angles of the corresponding vectors from the two vector chains, we get

$$(\theta_{F2p}, \omega_{F2p}, \phi_{F2p}) - (\theta_{F1n}, \omega_{F1n}, \phi_{F1n}) = (\varepsilon_{\theta n}, \varepsilon_{\omega n}, \varepsilon_{\phi n}), \{n = p\} \tag{5.21}$$

A representative $\varepsilon_p rime$ of ε_θ, ε_ω, ε_ϕ is required, which is retrieved using the following

$$\varepsilon_{prime} = \max(\varepsilon_\theta, \varepsilon_\omega, \varepsilon_\phi) \tag{5.22}$$

Processing all the vectors in a similar fashion produces a set of $\varepsilon_p rime$, which is $(n = p)$ dimensional. The root mean square (RMS), otherwise known as a quadratic mean, is a well used statistical measuring tool used in engineering, which is ideal in providing a representative measure of the deviation of one vector set from the other. The RMS is defined as

$$RMS = \sqrt{\frac{\sum_{i=1}^{n} (x_i)^2}{n}} \tag{5.23}$$

, where $i = 0....n$. Thus, $\varepsilon_p rimeRMS$ can be defined as

$$\varepsilon_{primeRMS} = \sqrt{\frac{\sum_{i=1}^{n} (\varepsilon_{prime_i})^2}{n}} \tag{5.24}$$

A threshold δ, which is user specified and having a very small floating-point value, is used as a comparison target for $\varepsilon_{primeRMS}$. Thus, a multi-dimensional comparison problem is reduced into a solution with a single variable comparison. So it is safe to assume that a target and a source pose are the same if $\varepsilon_{primeRMS} \leq \delta$.

5.4.3.5 Rotation Invariance in Local Space

The algorithm given above is applicable only in neutral alignment in global space. In Figure 5.12, the first set of images where the poses are identical, it can be seen that with respect to the global coordinate system, the first link (the parent link) in both source and target linkages maintain the same orientation. In the present state of the algorithm, congruence in orientation is a required constraint for it to provide the correct results. Below is given a screen-shot of a situation where the algorithm fails, because the above stated constraint is not satisfied.

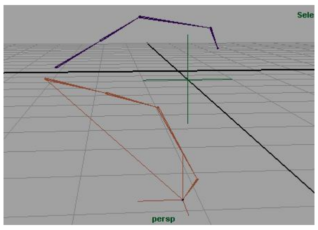


Figure 5.3: In global space, both linkages are different. And yet, in the local space of the parent link, they have the same pose.

In Figure 5.3, the linkages maintain the same pose in local space, but in a common global space the poses are different. It is vital to note that the pose is determined using the root node of the linkages as the origin, which in the above images is the extreme left end of the linkages. So, for pure pose comparisons global orientation must be discounted, which leads to a comparison invariant under rotation. The directional cosines of each of the vectors generated from the linkages define the cosines of the angles subtended by the vectors with each of the basis vectors. The basis vectors define a coordinate space. Therefore, in order to transform the respective vectors into local coordinate spaces, it is necessary to define new sets of basis vectors that define the spaces. The required steps that modifies the original algorithm to work in arbitrary root node orientation is:

(i) Transform the respective vectors into the local coordinate space of the respective linkages.

(ii) Calculate the RMS differential of the directional cosine angles as a post transform process.

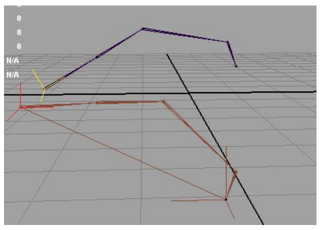


Figure 5.4: The hypothetical local basis vectors (in red and yellow), at the root nodes of the two linkages.

Let $\overline{V}_1$,$\overline{V}_2$, $\overline{V}_3$ be the three basis vectors that define a coordinate system. Arranging the components of the vectors in the columns of a matrix M_b, creates a basis matrix.

$$M_b = \begin{bmatrix} V_{1x} & V_{2x} & V_{3x} \\ V_{1y} & V_{2y} & V_{3y} \\ V_{1z} & V_{2z} & V_{3z} \end{bmatrix} \tag{5.25}$$

The basis matrix M_b is instrumental in transforming the vector chains from global space to the local space of the linked chain. Transforming any vector $\overline{V}$ by M_b transforms $\overline{V}$ into the global space. But the reverse transformation (transform $\overline{V}$ from global space to the coordinate space represented by the basis vectors $\overline{V}_1$, $\overline{V}_2$, $\overline{V}_3$) is performed by taking the inverse of M_b. For a square matrix, the inverse is equivalent to its transpose. Therefore if ${M_b}^T$ is the transpose of M_b, then

$$\overline{V}_{local} = {M_b}^T \overline{V}_{global} \tag{5.26}$$

The basis vectors are orthonormal vectors. The initial two vectors used to create the first basis vector is the root node vector of the linkage and any one of the global coordinate axis (x, y or z). The resultant vector of the cross product of these two vectors forms the first basis vector. Subsequent cross products with the new generated vector and the initial two vectors provide the second and third basis vector. The complication arises when the initially chosen two vectors are parallel, in which case the cross product will be zero. In such a case, the chosen coordinate axis has to be swapped with any of the other coordinate axes. For example, if y was chosen initially, then it has to be swapped with x or z.

If $\overline{X}$, $\overline{Y}$ and $\overline{Z}$ are global axis vectors, $\overline{V}_R$ is the root link vector and $\overline{V}_1$, $\overline{V}_2$ and $\overline{V}_3$ are the basis vectors to be calculated then

$$\overline{V}_1 = \overline{V}_R \times \overline{X} \tag{5.27}$$

$$\overline{V}_2 = \overline{V}_R \times \overline{Y} \tag{5.28}$$

$$\overline{V}_3 = \overline{V}_1 \times \overline{V}_2 \tag{5.29}$$

5.4.3.6 End-effector based Objective Function

The pose-based objective function measures the difference in orientation of the vectors calculated from each skeletal segment, in the input frame and also the corresponding output frame. In order to simplify calculations and reduce iterative complexity the number of vectors compared required reduction.

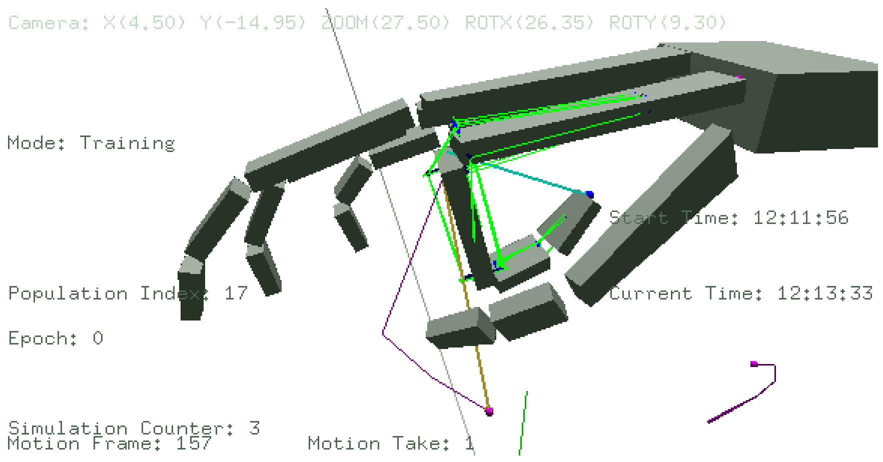


Figure 5.5: The cyan line from the end of the finger to the MCP joint is the end effector vector of the dynamic motion generated and the brown line is the end effector vector of the motion sample superimposed. The orientation of these two vectors provides an indication of differing poses in the input and output.

The error calculation is the same as the pose-based objective function equation, Equation 5.24. The only difference is the number of vectors compared. While the pose-based objective function utilised all the segment vectors, the end effector based objective function uses a single vector generated from the end point of the skeletal finger and the MCP joint (see Figure 5.5). In the pose-based objective function, the magnitude of the vectors are not considered because the vectors formed a chain and orientation alone defined a pose. But in the case of end effector vectors, both magnitude and direction are important.

$$\xi_{mag} = abs(|\overline{E}_m| - |\overline{E}_d|) \tag{5.30}$$

$$\xi_{dir} = O(\overline{E}_m, \overline{E}_d) \tag{5.31}$$

where O is the vector comparison function and $\overline{E}_m$ and $\overline{E}_d$ are the end effector vectors of the motion sample and the dynamic finger respectively.

$$\xi_{tot} = \xi_{mag} + \xi_{dir} \tag{5.32}$$

The total error is the sum total of magnitude difference and directional difference of the end effector vectors.

$$f = \frac{1}{\xi_{tot}} \tag{5.33}$$

where f is the final fitness.

5.4.3.7 Motion Variety Penalty Function

The method of traversing all the possible neural clusters capable of generating the coordinated activation requires examining the population of neural clusters/networks and selecting those that satisfy the fitness criteria. Network encoding in chromosomes enables the GA to generate a population set of possible neural networks that can produce physical motion from kinematic input motion. Initial test runs used only the pose-based objective function for calculating the fitness of each network. But it was observed that some of the generated networks created better continuous variation in activation than other networks which output a constant activation. Constant activation caused the rigid bodies to reach a constraint maximum and maintain that pose for the duration of simulation. If the input motion also had similar poses repeating frequently, then the pose-based objective function marked these networks as having a high fitness, without taking into consideration the overall motion generated. This caused deterioration in network performance in subsequent generations of evolution because of poor genetic transmission.

The primary aim of the research is to replicate motion physically, the key word being "motion" and not merely static poses retained over an extended period of time. The solution to the above stated problem is to insert a *penalty* term into the primary objective function that penalises a network depending on the motion generated by it during the training simulation.

The penalty is calculated by comparing subsequent frames in all the stored poses of the generated motion by the network.

$$\xi_{accum} = \sum_{f=0}^{N} (P_{f+d} - P_f) \tag{5.34}$$

where $d = 1, 2, 3....N$ is the frame offset.

The basis of the calculation is the same as in the pose-based objective function, specifically as in Equations 5.24, 5.27, 5.28 and 5.29. Thus, the sequence of poses is examined by measuring the amount of deviation between poses. This is indicative of the amount of movement generated by the network.

Due to the segmented nature of skeletal bodies, it is perfectly possible for parent segments to be static while the child segments move. In order to get a better estimate of the error, calculation is performed for each segment by converting it to the local coordinate space of the segment parent (see Figure 5.6).

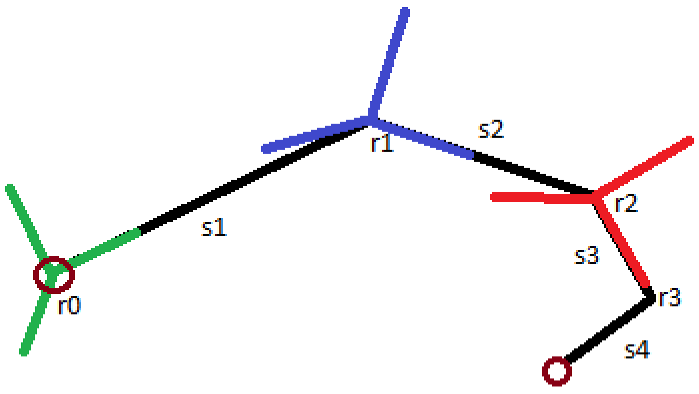


Figure 5.6: The local coordinate axes for each segment root are shown in red, blue and green. The segments are s1, s2, s3 and s4 and the roots are denoted by r0, r1, r2 and r3.

Equation 5.34 is subtracted from Equation 5.24. Conventional selection operators requires that the fitness is always positive. Subtracting the penalty has the drawback that fitness becomes signed. But since the genetic operation uses only mutation without the conventional selection, the sign of the fitness is immaterial. Thus,

$$\varepsilon = \varepsilon_{primeRMS} - \xi_{accum} \tag{5.35}$$

$$f = \frac{1}{\varepsilon} \tag{5.36}$$

where f is the final fitness. The inclusion of the penalty term produced considerable difference in network performance in later generations.

5.5 Artificial Neural Networks

Artificial neural networks (ANN) are mathematical abstractions in interconnected form, of the singular computing element in the human brain, the neuron. They are used in a wide of variety of classifying tasks as they have shown good performance in identifying patterns from large sample datasets and generalising to unknown datasets. They are useful in modelling non-linear relationships between variables where analytical approach fails. Muscle control is a computational neuroscience problem. Owing to high non-linearity and complex interactions between various biological elements, mathematical methods tend to contain numerous variables that require tuning. Manually tuning the variables becomes a cumbersome task.

The ANNs modify the synaptic weights between neurons through learning models. The most common learning model is the error Back-propagation (BP) and variations of it that try to overcome the drawbacks of the original model. The success of the back-propagation depends on the network architecture, the activation function of the neurons and the learning algorithm (Negnevitsky, 2005, p. 176). It is also dependent on the actual output and desired output at each neuron in the output layer. The difference between the desired output and actual output defines the error at that neuron.

With regard to neural control learning from surface motion, this is a presupposition and it prevents the use of BP as a learning model in a neural network that acts as a control system for coordinated muscle activations. The training samples (extracted from motion capture) provide a "style" of motion that informs the trajectory the limbs have to follow, which is the desired objective. But the objective itself is not an indication of the desired muscle activations. Desired muscle activation is an unknown quantity unless EMG readings of the real muscles are recorded and used in training. Therefore, in the absence of EMG readings, error calculation is not feasible.

5.5.1 Evolutionary Neural Network

In view of the situation detailed in Section 5.5 weight modifications can be performed using a hybrid approach combining the features of an evolutionary optimization method like the Genetic Algorithm and Artificial Neural Networks. An artificial neural network that uses evolutionary optimization techniques as a learning model is called an *Evolutionary Neural Network* (ENN) (Yao, 1999). There are at least four distinct ways a GA can be used in conjunction with an ANN. All of them require the modifications of the network parameters. They are:

(1) **Evolving synaptic weights** - The connection weights are modified iteratively through genetic operations, guided by a suitable chosen objective function.

(2) **Evolving network architecture** - The architecture of network topology can evolve over time leading to an optimized topology.

(3) **Evolving the Learning rules** - It evolves the learning rules to create adaptive and probably better efficient learning algorithms.

(4) **Evolving the transfer functions** - Evolution can be used to determine optimal mixture of transfer functions in neurons. Normally, neurons in a layer have the same activation function like the sigmoid function. But experimental research like in Annunziato et al. (2003) attempts to use multiple transfer functions between neurons of the same layer.

Using a hybrid model like an ENN has certain features which makes it advantageous than other approaches. Nolfi and Floreano (2000) lists them as follows:

- Motor functions are smooth and continuous. Simulating a continuous control system would be best achieved through a model that is analog in nature. In neural networks, gradual changes in the network parameters produce gradual changes in behaviour.

- Evolutionary networks have different levels of evolutionary flexibility ranging from the low level specification like connection weights to higher level specification like network architecture. Nolfi and Floreano (2000) defines three evolutionary stages that can be applied to ANNs: *phylogenetic* (evolution), *developmental* (maturation) and *ontogenetic* (life learning). These stages pertain to architecture evolution starting from a base structure and evolving through time and adapting connection weights to learn from experience.

- The generalising capability of neural networks encodes information learned over trials and hence makes it ideal to represent an adaptive system like the motor control system in the human body.

5.5.2 Time Series Predictor Network

Dynamical systems where deterministic rules describe the state of the system can be classified as a *time series*. Connectionist architectures like ANNs are ideal for predicting future states in the time series (Frank et al., 2001). Neural networks are commonly used in the time series prediction of network traffic forecasting, traffic forecasting, weather forecasting, market analysis and more. A time series is highly suitable for understanding patterns in a dynamic process. By understanding the time series pattern in one system, it is possible to simulate the relation of that system to another. And so in the system described in this thesis, the recorded motion acts as a template of the underlying neural and muscular dynamics that produced the motion. Thus, ANNs are used as time series predictors of muscle activations for a given motion sample. The neural networks used in time series prediction normally has standard feed-forward architecture. The network accepts an N-tuple set as inputs that describe the sequence d steps back in time. This method is often termed as a *sliding window technique.*

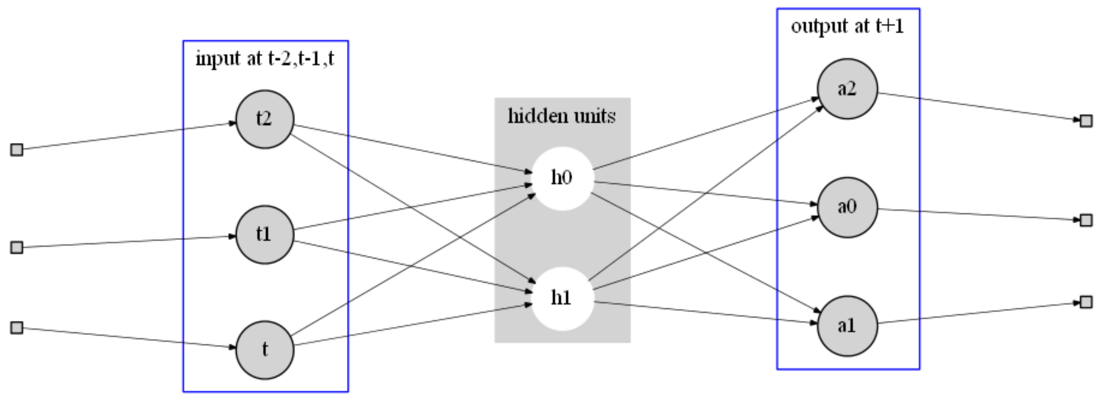


Figure 5.7: Vectors at three preceding time steps are fed into the network. The output is an extrapolated value for time t+1.

Animation is a sequence of static frames, instances in time, which when combined in the right order produces coherent action. The input to the ANN is a motion sample with key poses at each frame of the sample. The motion sample is a recording of a dynamic system (the neuro-musculo-skeletal system). Therefore, by treating the motion sample as a time series problem, the nature of the dynamic system can be recreated. Since the neural network exists within the same dynamic environment as the rigid bodies and the muscles, the outputs (activations) always generate continuous motion for any input irrespective of coherence.. The correlation between the input vectors and the output activation is unclear and there is the possibility that continuous motion is regarded as the correct output.

The activations that generate an arm or finger motion take the arm or finger from an initial state at time t to another state at time t+1. In the animation sequence given as input to the network, the time relation is maintained. It is imperative that the network learns the time relation between the input sequence.

In the dynamic system, the space of muscle activations that generated the original motion can be termed as the *phase space*.

It is up to the neural network to capture the nature of this phase space, so that each subsequent activation takes the dynamic system from one state to the other in time. The neural network, by using the sliding window technique, "slides" over the training set. So for every $t + 1$ step, the network outputs the activations that allow the muscle to contract so that the dynamic linkages mimic the orientation of the input skeletal set.

Mathematically, the predictor function can be defined as,

$$x(t + d) = f(x(t), x(t - 1), ..., x(t - N + 1)) \tag{5.37}$$

where f is a real valued function that calculates the value of the observed system at a future time $t + d$.

The size of the window is a crucial factor in pattern identification. If the window is very large, then the relation between subsequent time samples becomes less coherent. This prevents the ANN from identifying the pattern in the time series. Although, the correct window size is normally found through trial and error, the probability for the correct identification of the pattern is higher if the sizes between the samples are close to each other.

5.5.3 Architecture and Activation

Artificial Neural Networks are classified based on the architecture and also sometimes based on the activation functions used (such as Radial Basis function networks and Wavelet neural networks). Some of the different architectures are feed-forward networks, recurrent networks, adaptive resonance maps etc. The type of architecture is often dependent on the task at hand. So following the well researched notion that an artificial neural network with a single hidden layer can approximate any function that contains a continuous mapping from one finite space to another, the control system in this thesis used a fully connected feed-forward time series predictor network with three layers (see Figure 5.8). There are 27 inputs corresponding to 3 pose vectors for time $t - 2$, $t - 1$ and t. There are 8 hidden neurons in this version. The output layer consists of 8 neurons that give activations for the 8 muscles controlling the finger. A *Discrete Time Recurrent Neural Network* (see Figure 5.9) was also created in order to map the input motion to temporally correlated muscle activations. The recurrent neural network incorporates feedback neurons that preserves the previous state of the network which is fed back into the hidden layer. This functions as a memory retention layer and is perfectly suited for learning in dynamic systems. In the neural network controller, the activations output at a previous time step are input to the hidden layer in order to provide additional information regarding the state of system.

The three layers are: *input*, *hidden* and *output*. The number of neurons in the hidden layer varies with the type of problem and deciding the correct number is a matter of trial and error. The heuristic followed as a starting point in this thesis is as follows (adapted from Heaton (2005)):

- The number of hidden neurons is between the number of input and output neurons.
- The number of hidden neurons is two-third the number of input neurons plus the number of output neurons.
- The number of hidden neurons is less than twice the number of neurons in the input layer.

The first rule was followed by taking the average of the number of input and output neurons.

Unlike first generation neural networks that can compute only binary functions (the activation function produces digital output), the neurons in the evolutionary neural network implement continuous activation functions (analogue) capable of representing a wider range of behaviour.

Activation functions like the *step* and *sign* functions are binary. Such functions are called as *hard limit* functions (Negnevitsky, 2005). Networks using these functions are called *linear separators* (Negnevitsky, 2005). In an active control process during task execution in the human body, the force output by the muscles vary in accordance with neural stimulation. In other words activation can be termed as continuous. And hence, the activation function used in the neurons of the hidden and output layers of the network is the *sigmoid* function.

$$S = \frac{1}{1 + e^{-X}} \tag{5.38}$$

where X is the net weighted input, given by

$$X = \sum_{j=1}^{n} w_j x_j(t) \tag{5.39}$$

Unlike hard limit functions, the sigmoid function converts any value between plus and minus infinity into a value ranged between 0 and 1 (see Figure 5.10).

This property makes it an ideal choice to use it as an activation function for the neurons in the hidden and output layers. It can be seen later in Section 5.6 that muscle activation is represented by a value between 0 and 1.

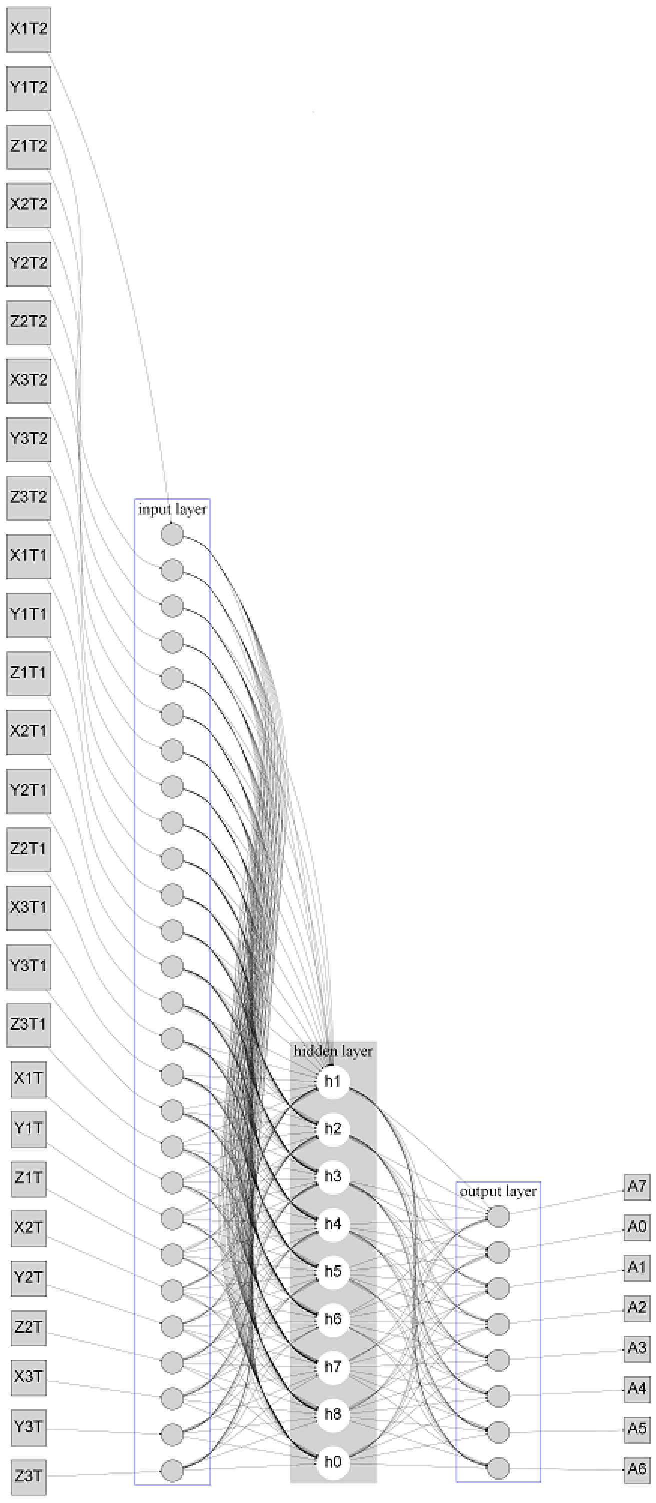


Figure 5.8: The fully connected time series predictor artificial neural network. The input is the pose vector of the forefinger at time states $t-2$, $t-1$ and t, which is an input vector containing 27 elements. In this particular network, there are 9 hidden nodes. $A0..A7$ are the activations for the eight simulated muscles of the forefinger.

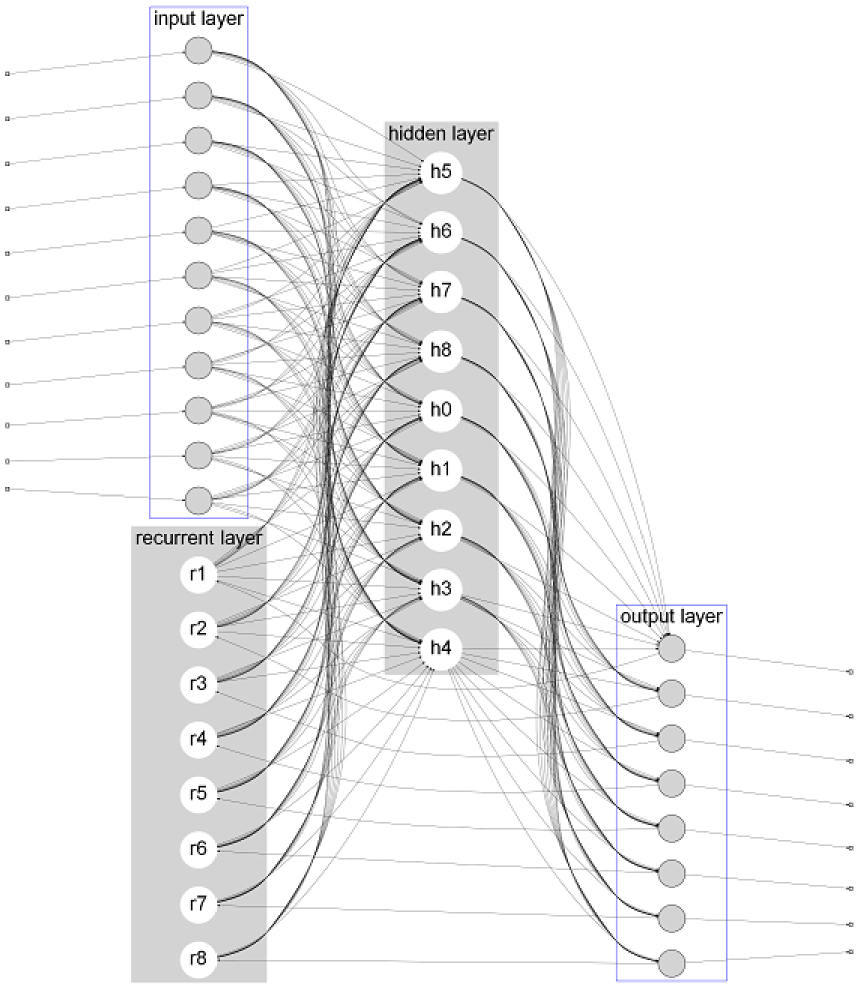


Figure 5.9: Recurrent architecture.

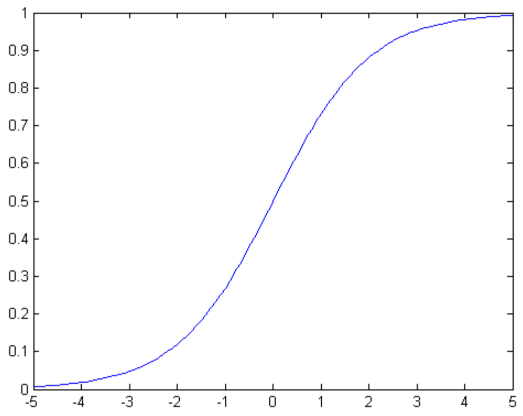


Figure 5.10: The continuous sigmoid function with output between 0 and 1.

5.6 Controlling Muscle Activations

According to Adamczyk and Crago (2000), "Activation and deactivation dynamics are the processes that describe the delay between muscle force development (i.e., the delay between the neural excitation arriving at the muscle and the muscle developing force) and relaxation (i.e., the delay between the neural excitation ceasing and the muscle force falling to zero) that is a characteristic of the excitation-contraction coupling." Muscle activation is a complex time dependent process that gradually generates the force. In short, a muscle activates when the motor units fire and the actin-myosin cross-bridges start to form (IvyRose, 2010). Force generation in a muscle is gradual and is dependent on both the activation strength and duration. *OpenSim* is a biomechanical simulation toolkit used in the analysis of muscle activation dynamics (OpenSim, 1990). Similar biomechanical systems implement a Hill-type muscle model that models activation and deactivation dynamics as first order differential equations. Activation is represented by a real value in the interval (0, 1) making it a real-valued continuous optimization problem. Unlike PD controllers where target angles can be negative, limiting activations to a positive range makes it suitable for a neural network output using sigmoid activation functions. This also allows the muscle behaviour to have a singular function (contract) and mimic the biological counterpart where antagonistic muscles are used to generate opposing motion. The timing of activations during a movement is too complex to recreate manually, especially when multiple muscles are involved. The dynamic calculations involved are often complicated and time consuming computationally.

5.6.1 Activation Modulation per frame

Time dependent functions, like the motion equation in dynamic simulations, modulate function variables at each time step. Dynamic simulation of an object requires the calculation of the forces acting on it at every time step. Thus, in an active control scenario like muscle actuated motion, forces produced by each muscle must be known ahead to apply on the rigid body. This is achieved through forward dynamics methods. In a normal rigid body simulation, the force value is not modulated *actively*. The system computes the forces acting on the object in the simulation environment, like gravity for example, and performs the Newtonian calculation for each time step using time as a parameter. Examples of the said *passive* simulation are an object falling under the force of gravity and bouncing to rest, a ball rolling after a force is applied. On the contrary, constraining a rolling ball on a curved trajectory requires modulation of the forces acting on it at every time step. Thus, it can be summarised that force and time are important quantities in generating motion through dynamics. Since the force is a vector, the modulation is two fold:

- **Magnitude**: The magnitude requires modulation at each step in order to prevent the ball from off-shooting the trajectory. So corrective forces need to be applied.
- **Direction**: Similarly the direction of force application is modulated to keep in synch with the magnitude modulation and maintain the trajectory.

In the human body, the modulation is performed by the CNS through a complex neural process (Shadmehr and Wise, 2005). Equation 5.4 in subsection 5.2 describes the force changes or force predictions in the model residing in the CNS during a transition from one movement to another at every time step.

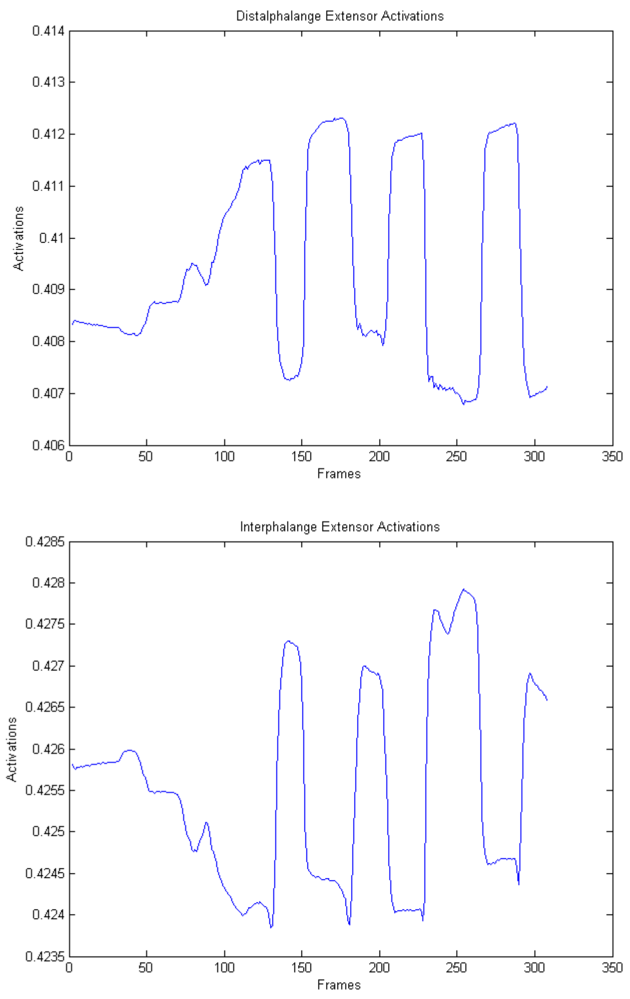


Figure 5.11: The untrained time activation graphs for the distal and inter phalanges for a period of 308 frames.

Through constant neural signalling, the central nervous system maintains sustained activation in muscles to maintain positions or trajectories (MussaIvaldi and Bizzi, 2000). This is necessary because neural stimulation in the CNS that pertains to a muscle movement creates a dynamic (as opposed to static) force field. The sustained activation affects the muscle lengths which in turn affects or produces the force generation of the muscle (MussaIvaldi and Bizzi, 2000). Normally differential equations are used to express the complex dynamics involved in the simulation of muscle actuated control. Solving the differential equations always require numerical solvers and depending on the number of variables involved, solving can be time inefficient. The number of force variables is dependent on the number of muscles involved in the action. This generates a layer of complexity with regard to activation prediction. If the force is not considered as a computational factor, the problem can be simplified through the use of neural computing elements and the concept of a scaling representation of force.

The force acting on the body can be expressed linearly as a fixed force magnitude (that is user specified) scaled by the muscle activations output by the neural network (see Equation 5.40).

$$\tau_i(t) = a_i(t)|\overline{F}| \tag{5.40}$$

where $\tau_i(t)$ is the force of the muscle at time t, $a_i(t)$ is the activation of the neuron innervating the muscle at time t and $|\overline{F}|$ is the force magnitude input by the user.

With the muscle activations changing at every time step, scaling the force magnitude using activations produce a changing force every time step. The force magnitude is embedded with a directional vector by way of the muscle system described in Chapter 4. Combining Eqn. 5.38 and Eqn. 5.39, the activation of the output neuron i at time t is given by,

$$a_i(t) = \frac{1}{1 + e^{-\sum\limits_{j=1}^{n} w_j x_j(t)}} \tag{5.41}$$

where n is the total number of incoming connections at neuron i and w_j is the weight and x_j is the input of the j^{th} connection.

The unknown activation function is highly non-linear and dynamic with a vast search space. The neural network is used to approximate this function and produce coordinated motion. (see Figure 5.11).

In a simulation, a typical output of the neural network would have activation values per frame (see Table 5.1).

The neural network has to recognise a mapping between the changing pose of the limbs, input as pose vectors (see Table 5.2) and probable corresponding muscle activations. In the course of a motion, there are some muscles that activate and others that do not or all muscles activate at varying levels. The activations are dependent on the type of motion, the direction of motion, the external forces

Table 5.1: Activation modulation table for 8 muscles at time ranging from t_0 to t_n

Time	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**
t_0	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}
t_1	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}
....	....	....	....	....	....	....	....	....
t_n	a_{n1}	a_{n2}	a_{n3}	a_{n4}	a_{n5}	a_{n6}	a_{n7}	a_{n8}

involved that the motion is playing against. Since the neural network is receiving constant input at each frame of simulation, there is a constant stream of output. This is indicative of co-activation of the muscles. Thus every muscle is active at every point of time. The determining factor for the shape or envelope of the motion is the amount of activation for each muscle.

Table 5.2: The first 6 frames of the motion sample, converted into vectors for the phalanges of the forefinger. The first training set is a motion sample of 308 frames. The columns are in sets of x,y and z components.

X	**Y**	**Z**	**X**	**Y**	**Z**	**X**	**Y**	**Z**
4.8043	0.18845	0	2.6021	-0.0834	0	2.4028	-0.0777	0
4.8026	0.22652	0	2.6027	-0.0629	0	2.4033	-0.0590	0
4.8023	0.23661	0	2.603	-0.0573	0	2.4036	-0.0537	0
4.8031	0.22428	0	2.6029	-0.06534	0	2.4034	-0.0609	0
4.8028	0.22626	0	2.6028	-0.0644	0	2.4034	-0.0599	0
4.8034	0.22065	0	2.6029	-0.0675	0	2.4034	-0.0627	0

5.7 Standalone Outputs of Genetic Objective Function and Neural Activation

Below are given screen captures of kinematic linkages first created in Maya, and then later exported into the custom C++ application. The linkages consist of four links in each chain. The linkage labeled A in the left image is the target chain and chain B is the source. The application compares linkage B with linkage A to calculate by what amount linkage B differs in pose from linkage A. The output of the external comparison application (threshold, RMS differentials, source/target linkage sizes and comparison result) is given below each set of images.

For all the cases in Table 5.3, the threshold (δ) is 0.15.

Below are given the new results after implementing the rotational invariance comparison. The image on the left is screen captured from Maya and the

Table 5.3: Objective Function Results

Case	Description	No: of links	RMS Differential	Result
1	Identical in pose and proportion (see Figure 5.12)	4	0.0601041	Source and target chains are in the same pose
2	Proportionally same, different pose (see Figure 5.13)	4	0.201456	Source and target chains are in different poses
3	Proportionally different, identical pose (see Figure 5.14)	4	0.0619632	Source and target chains are in the same pose
4	Different both in proportion and pose (see Figure 5.15)	4	0.219443	Source and target chains are in different poses

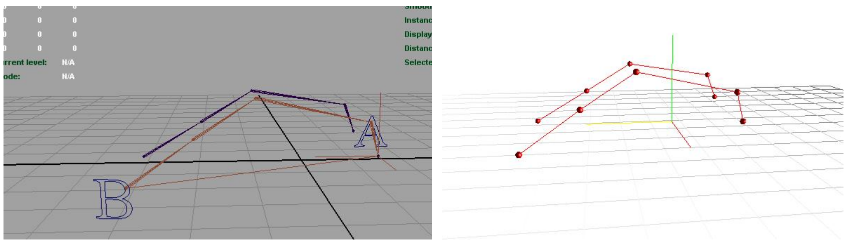


Figure 5.12: The two sets of linkage chains (Maya joint tool) created in Maya. The brown coloured linkage labelled B is the source chain and the chain labelled A is the target chain. On the right, the same two chains are exported from Maya into the external application, which performs the comparison test. Both links are proportionally same and maintain the same pose.

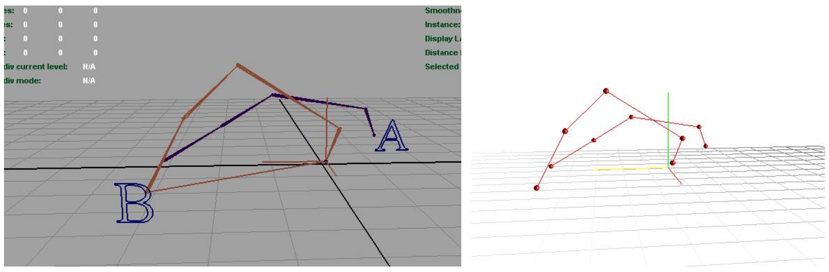


Figure 5.13: On the left, Maya scene and on the right the external application. The source chain is in a different pose from the target chain.

image on the right is the screen capture of the application implementing the algorithm.

In addition to acting as an objective function for the genetic algorithm,

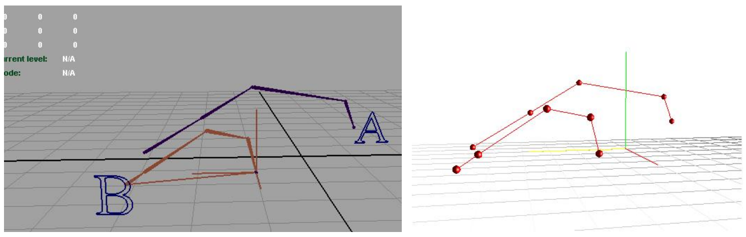


Figure 5.14: On the left, Maya scene and on the right the external application. Here the source chain is of different proportions, but have the same pose and the objective function can still compare the two poses.

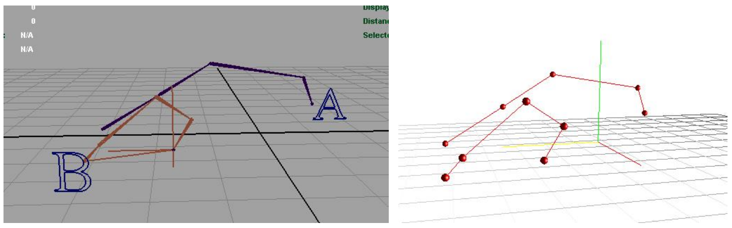


Figure 5.15: On the left, Maya scene and on the right the external application. The links are proportionally different and the poses are different.

Table 5.4: Objective Function Results after Local Space Transformation

Case	**Description**	**No: of links**	**RMS Differential**	**Result**
1	Identical in proportion with same local pose, different global orientation (see Figure 5.16)	4	0.118765	Source and target chains are in the same pose
2	Proportionally different, same local pose, different global orientation (see Figure 5.17)	4	0.118765	Source and target chains are in the same pose

the algorithm can be used to retrieve animation frames from a motion sample of 4 segment skeletal linkage. In the following Table 5.5, the frame to be retrieved is the frame where the RMS differential is zero. But the threshold given was 0.015 in order to retrieve the closest frames around the required frame (frame no: 46). The time taken for retrieval is also shown. The motion sample had 300 frames in total.

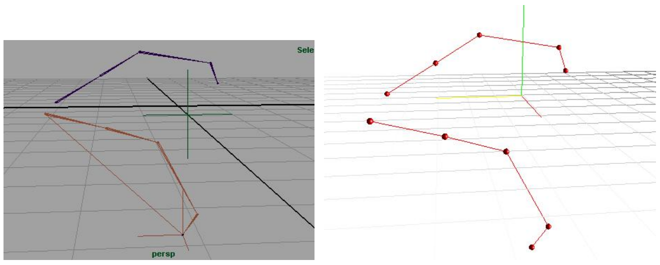


Figure 5.16: On the left Maya and on the right, the output of the external application. With the modified algorithm, the comparison works by identifying the two linkages as having the same pose.

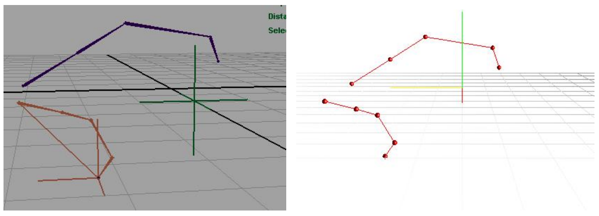


Figure 5.17: On the left Maya and on the right the output of the external application. The modified algorithm recognizes the pose even when the source links are in different proportions.

Table 5.5: Objective Function Results for Frame Retrieval

Frame No:	**RMS Differential**	**Time taken (seconds)**
42	0.00203514	0.002
43	0.00126628	0.001
44	0.000650087	0.001
45	0.000217639	0.002
46	0	0.001
47	2.83306e-005	0.035
48	0.000334082	0.002
49	0.000948799	0.002
50	0.00190461	0.001

5.8 Summary

Biomechanical muscle models for generating human motion have sub-models like activation dynamics, contraction dynamics and skeleton dynamics that involve solving complex differential equations. Muscle dynamics, particularly coordination of limbs through synergistic co-activations is a complex problem with a huge solution space. The human central nervous system traverses this space in a non-trivial manner. For locomotor functions, the central nervous system contains neural models (called CPGs) that generate cyclic patterns in the motor neural activation. But movements that are non-repetitive and requiring an active modulation of motor faculties to execute are difficult to represent using periodic functions. As mentioned in section 5.2, the CNS performs generalizations from previously executed motions. The CNS uses the basis functions in the neurons participating in the movement, which when combined can approximate linear or non-linear functions. The objective of this research is to use the biological model of CNS motor control as a basis to generate controlled muscle activations for given input motions. Artificial neural networks are used as a simplified facsimile of the biological neural networks.

Thus, this chapter explained the control and computational system that simulated the motor functions of the CNS using artificial neural networks. Due to the difficulty of obtaining muscle activation data, the simulated motor control system is trained using genetic algorithms. Genetic algorithms are very useful in finding solutions where the search space is large. The network weights are encoded as chromosomes and subjected to artificial evolution and selected based on the fitness functions explained in sections 5.4.3.1 and 5.4.3.7.

In order to recreate the muscle activations that produced the sample motion, a technique called *time series prediction* is used with the artificial neural networks where previous poses of the sample animation is used to generate predicted values of activations.

Thus ANNs, in conjunction with GAs are used to sample the large muscle activation space to isolate the correct activations that correspond to recorded surface motion.

6

RESULTS AND ANALYSIS

6.1 Introduction

Muscle dynamics is computationally complex to perform manually, and hence require a controller system to automatically manage the coordinated activations that produce a given motion. This chapter applies the controller system concepts detailed in the previous chapter to a dynamic hand model in a real-time game physics environment and discusses the results of the generated animation. This chapter also presents the results of a very early version of the system that used PD controllers as actuators.

Section 6.2 gives a short description of the two broad groups of hand animation. Section 6.3 presents a the very early test results of a PD controller-based dynamic hand model, an early physical model driven by an XBox360 game controller, grasp contact determination tests (6.3.2) and the implemented Maya plug-in. Starting from Section 6.4, the chapter presents the results and analysis of the neural controlled muscle dynamics-based model. It presents motion tests of the dynamic finger model rigged with the locator-based muscle system. It looks into the time series predictor network format and effect of window sizes on convergence, It also looks into the effects of scaling forces of the muscles and changing the original motion pattern by using non-uniform force values. Sub-section 6.4.1 analyses the motion produced using the system, the effect of noise addition in neural output and activation graphs both prior to and after training. Sub-section 6.4.2 explains a modification of the neural system to enable a procedural control system similar to inverse kinematics. And finally, in Section 6.5 the current problems and short comings of the system like accuracy of the physics engine are examined.

6.2 Types of Hand Animation

In the real world, since we use our hands for a variety of purposes, many of which are purely based on the context of usage. This research classifies the hand movements into two categories:

- **Hand gesture**: Hands are used for communicating through gestures. For the hearing impaired, a gestural language itself is developed called sign language and is widely used in conversing. An example is American Sign Language (ASL) (Xu et al., 2009). Gestural interfaces are a hot bed for research in Human-Computer Interaction (HCI). In gestural animation, interaction is devoid of contact with the environment. From a physical point of view, the environment has little (gravitational forces) or no effect on the internal forces generated by the muscles and tendons of the hand.

- **Hand grasping**: Grasps form the base type of interactive function that hands evolved to perform. There are different categories of types of grasps (Napier, 1956). Grasps require our hands to have physical contact with the environment with an exchange of forces. Complexity of grasps lies in the balancing of forces to maintain a stable grasp.

Observational analysis makes it clear that all hand animation, either falls into or into subsets of these categories. Having stated that, there are also types of hand animation, which fall into neither of the above classification, namely, push, touch, rub etc. And a few of those are difficult to be isolated from a full body motion and are the direct result of movement initiated by muscles higher up the fore limbs. These types of motion are not the subject of this research.

6.3 Results during the Initial Phases

A review of existing research proved that considerable work was already done in the area of grasping animation like the work done by Nancy S. Pollard ((Pollard and Zordan, 2005), (Li and Pollard, 2005), (Pollard, 2004), (Pollard, 1996)). While grasping research examined computational problems, grasp animations were implemented through traditional inverse kinematics methods and PD controllers (Pollard and Zordan, 2005). The composite problem of hand animation in grasping was divided into two layers, namely, the computational layer (mapping points on the object using both object and hand properties) and the animation layer (that actually moves the fingers to the mapped regions computed on the object in the previous layer). The animation layer is further sub-divided into two groups, which are traditional key-frame animation and physics-based animation. So it was evident that the area for research had to be narrowed. Muscle-based physical motion proved ideal in view of the techniques intended for use.

Before present iteration of the physical system, the early prototype used for experimentation used hand models (proportional to a real hand skeleton), which relied on a deprecated concave mesh support in the PhysX engine (Ageia, 2006). The concave mesh (as opposed to convex mesh) allowed for converting polyhedral meshes directly into a rigid body. And so, hand models from Maya were directly imported into the PhysX engine. Due to drastic reduction in performance while using high-resolution meshes as rigid bodies, the researcher opted for a "box" model of the hand, created in Maya using the high-resolution mesh as a reference and directly imported into PhysX as a concave mesh. Concave mesh support was later removed by the developers. Also, collision detection between concave meshes was performed by another deprecated API structure called *PMap* (Ageia, 2006). By redesigning the system later and combining convex meshed rigid bodies with high-resolution mesh models proved to be extremely efficient and aesthetically pleasing. The muscle system was also very rudimentary. Tests were done with implicit Proportional Derivative joint springs. The explicit actuator tested was an ordinary line-of-action based muscle actuator with joint wrapping absent. The following sub-sections present the results obtained during the initial phases.

6.3.1 Early Physical Model

In the real world, muscle contractions and expansions drive the skeletons in contrast to the virtual world where the reverse takes place. In a musculo-tendon ensemble, the force is developed in the muscle and transmitted to the bones through the tendons. Some filtering occurs in the true contraction force due to the non-linear passive properties of the connective tissues (Maurel et al., 1998). It also depends non-linearly on the current length, velocity and activation of the muscle spindle. Biomechanical models help in measuring musculo-tendon force generation. A basic model is the linear spring-damper, which views muscles as spring-like, exhibiting linear forces in the direction determined by an origin and an insertion point (Maurel et al., 1998).

$$f = k_s(L_c - L_r) + k_d v_s \tag{6.1}$$

where f is the spring force, k_s is the spring constant, L_c and L_r the current and rest spring length respectively, k_d is the dampening constant and v_s is the spring velocity. Another type of virtual spring popularly called the *Proportional Derivative Controller* is used to keep a control variable and its derivative within the neighbourhood of desired values (Parent, 2007, p. 240-241). A PD controller is a continuous controller; generating target poses automatically from the current state of the system. PD controllers are useful for biasing a model toward a given motion and allowing it to react to system forces.

$$\tau_i = k_p(\theta_d - \theta) - k_d\,\dot{\theta} \tag{6.2}$$

where τ_i is the torque, k_p and k_d are the proportional and derivative constants, θ and $\dot{\theta}$ are the angle and angular velocity respectively.

The proportional gain controls the strength of the spring while the derivative gain adjusts how smoothly the joint reaches the desired value. Without muscles to apply passive forces to the linkage, the rigid body linkage would behave like a loose chain (Figure 6.1).

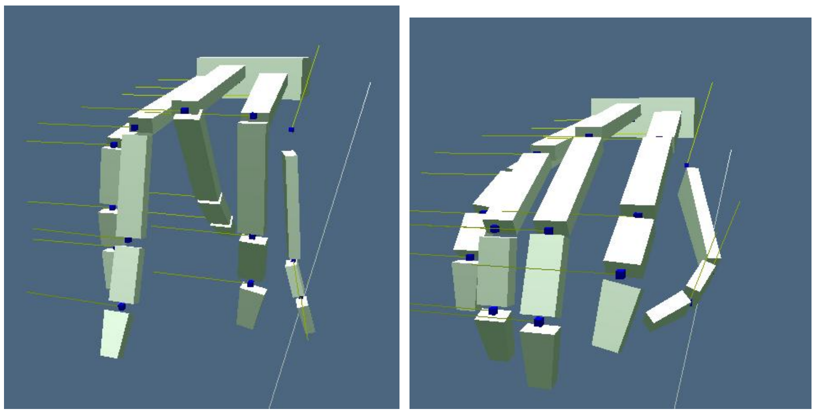


Figure 6.1: On the left, shows the effect of the absence of PD spring muscles. On the right, shows the effect of passive forces on the linkages from the joint springs functioning as muscles (PD). The pale green lines at the joints (blue boxes) denote the axis of rotation.

In order to observe the real-time motion behaviour of the fingers, the PD actuator was hooked up to a standard XBox360 controller. Support for the XBox360 controller was provided through the unified framework classes designed by the researcher. These classes adhere to strict object oriented design practices giving it flexibility and ease of use along with extensibility to use in other applications using the framework. Flexing of the fingers is controlled using the analogue triggers on the controller. The analogue output ranges between 0-255, which is normalized and clamped. This enables a precise and smooth control of the animation in real-time (Figure 6.2).

The PD controller has a couple of advantages. The motion produced looked organic. Maintaining static poses was stable. The disadvantage was that, the spring constants of the joint required a lot of fine-tuning to get the correct amount

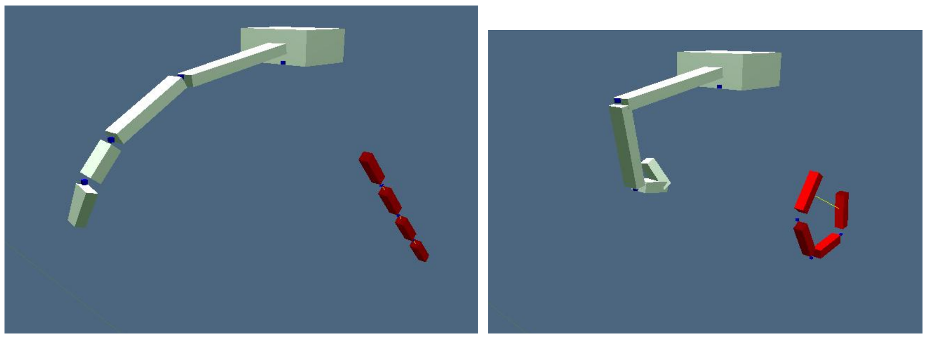


Figure 6.2: The dynamic finger (in white) is equipped with PD controllers on each joint while the linkage (in red) has a line of action force actuator. Both are controlled using the XBox360 gaming controller providing continuous control.

of compliance. The PD controller works fine as long the joint was a *revolute* type with a single degree of freedom. Multiple degrees of freedom require manipulation of the joint axis in addition to the spring constant and damping values. For a highly articulated organ like the hand, with 15 articulate joints (excluding the wrist), the number of tunable parameters would more than double the number of joints.

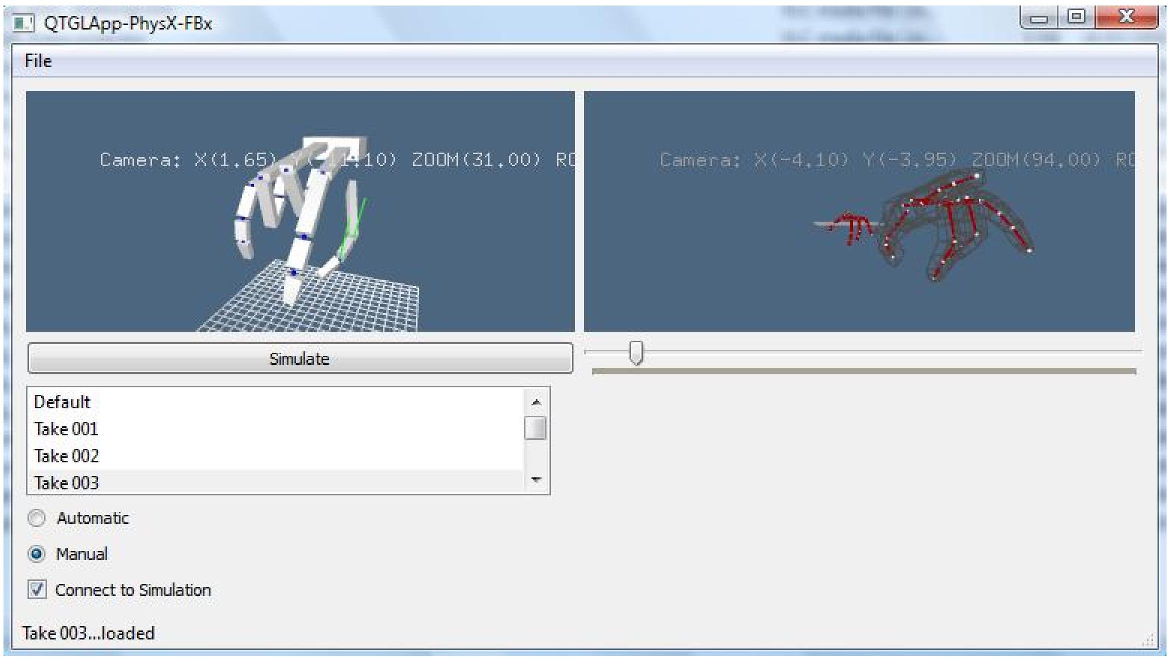


Figure 6.3: A Qt-based application and UI that demonstrates the PD controller. On the right window is the motion capture sample motion (with the animation takes listed in the list box below) and on the left is the simulated rigid body hand model with attached PD controllers at the joints.

A Qt-based stand alone application was also designed to interactively manipulate and observe the PD controller (see Figure 6.3). Once the simulation is connected to the motion capture animation (read from an Autodesk FBX file), the PD controller tries to track the animation. The animation can also be played manually by scrubbing the time-line controller. The PD controller requires a considerable amount of tuning to accurately track the motion.

Anatomical joints, acting merely as constraints unlike the PD controlled joints in PhysX, are not powered by any actuator.

6.3.2 Grasp Contact Determination Tests

Susan J. Lederman in Lederman and Wing (2003) says "Object symmetry is a visual attribute that may contribute to perceptual judgement and to action". For grasping an object, determining the grasping points on the object is essential. And so symmetry of an object might be a deciding factor for a sub-conscious brain function of grasp point determination. Without direct contact interaction, humans achieve this mainly through the primary sense of vision. Visual interpretation of the object is used to estimate the center of mass of the object (Lederman and Wing, 2003). We judge an object to be manipulated by mentally mapping grasping points on the object, depending on object geometry, size and also task. The former two factors are quantifiable as far as computer graphics is concerned, but the last factor is purely based on human judgement and prior knowledge, which would be difficult to describe mathematically in a precise manner. Even though the type of objects in the real world varies, there is a great degree of redundancy in the type of grasps used by humans. Since many successful grasps are whole-hand or enveloping or power grasps, one plausible solution suggested in Pollard (1996) is to develop grasping prototypes, like a generic cylinder grasp, and apply that to common objects. The grasping pre-shapes, which are the result of gross simplification of grasping tasks is classified into grasp taxonomies (Napier, 1956).

One of the easiest approaches is to use the line of action (a direction along which force is applied) of the fingers to determine preliminary contact points on the surface of the object. These lines of action can also be used for computing grasp contacts on the object surface and temporary spring attachments can be created to fix the hand on the object at runtime. The thumb plays a crucial role in any feasible grasp. It is the opposable nature of the thumb that allows for different types of grasps and provides for stability (Napier and Tuttle, 1993). Upon observation of grasps (both power and precision), it can be seen that the thumb is the convergence point of the line of action of the fingers during a grasping action. This *action line convergence point*, which is the meeting point of all the fingers, can also be offset from the thumb but staying within the finger limits (which can together be called a grasp volume) and can be shifted around to direct finger movements (Figure 6.4).

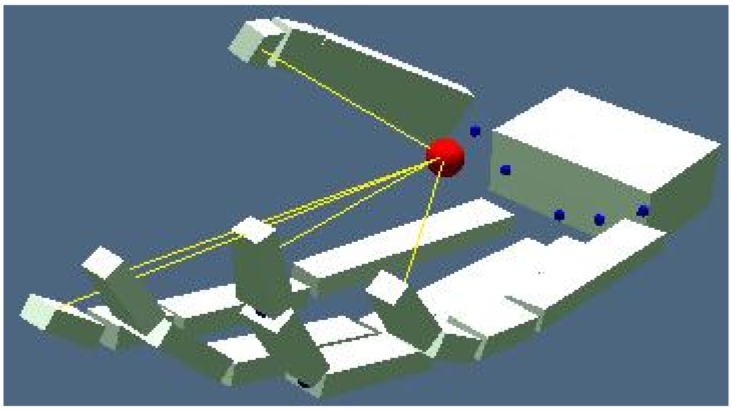


Figure 6.4: Action line convergence point (red sphere), a guide for lines of force.

Initial implementation of this theory was done within the Autodesk Maya graphics environment using the Maya C++ Application Programming Interface (API) (see Figure 6.5). The meshes of the object and hand used were discrete polygonal meshes rather than implicit surfaces. The Maya API facilitates the creation of user-defined nodes, which allow custom plug-ins to be integrated into the Maya dependency graph. With nodes it is possible to get real-time feedback in the view port.

Since discrete meshes (triangulated) are used, contact points on the meshes are represented as triangles. But then this restriction is purely based on how the action line is represented, whether as a line or as an action volume, with the thumb as the convergence point. With the action volume (group of action lines), each contact point would become a contact region. It is also possible to keep the intersection point of the action line as a point rather than as a triangle or group of triangles. In Maya locators are graphical objects that denote position in space acting as a reference point for the artist. In order to test the concept of action lines, locators are placed on the fingertips of the three fingered gripper model.

The gripper is rigged using the standard joint skeleton interface provided in Maya and animated using forward kinematics.

Each triangle can be treated as a plane and intersections of the action line can be calculated using simple vector math. A plane is represented by the C/C++ structure:

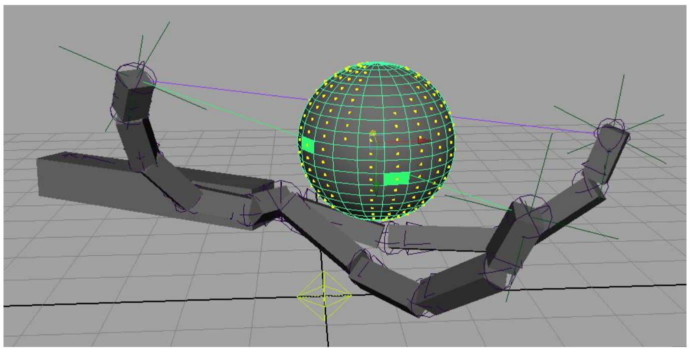


Figure 6.5: Screen capture of the Maya plug-in implementation of contact determination, using a three-fingered gripper. The highlighted green patches on the sphere show possible contact points along the action line.

```
struct Plane
{
  MFloatVector n;
  float d;
};
```

The structure holds the normal vector, n, to the plane and the distance of the plane from the origin, d. Each triangle in the target mesh is iterated over, and the plane equation for the triangle is calculated.

If a,b and c are the vertices of the triangle, then

$$\overline{V}_1 = b - a \tag{6.3}$$

$$\overline{V}_2 = c - a \tag{6.4}$$

where $\overline{V}_1$ and $\overline{V}_2$ are the vectors from vertex a to vertex b and vertex a to vertex c. Then the normal n is

$$n = \overline{V}_1 \times \overline{V}_2 \tag{6.5}$$

$(\times)$ denotes cross product. The distance of the plane from the origin, d, is found by finding the dot product of the normal n and vertex a of the triangle.

$$d = n \cdot a \tag{6.6}$$

Action lines are represented by a C++ class, which holds the two end points of the line. If $(x1, y1, z1)$ and $(x2, y2, z2)$ represent the two endpoints of the line, a vector V can be found using

$$\overline{V} = (x_2, y_2, z_2) - (x_1, y_1, z_1) \tag{6.7}$$

In order to find the intersection between the action line and the plane containing the triangle, let

$$S(t) = A + t(B - A) \tag{6.8}$$

where $0 \leq t \leq 1$ is a parametric equation of the action line, where A and B are the two end points on the line, so $B - A = V$ and

$$(n \cdot X) = d \tag{6.9}$$

is the vector plane equation and $X = (x, y, z)$. Substituting the parametric equation for X in the plane equation and solving for t obtains the t value of intersection of the segment with the plane:

$$(n \cdot (A + t(B - A))) = d \tag{6.10}$$

$$n \cdot A + tn \cdot (B - A) = d \tag{6.11}$$

$$tn \cdot (B - A) = d - n \cdot A \tag{6.12}$$

$$t = \frac{(d - n \cdot A)}{(n \cdot (B - A))} \tag{6.13}$$

If the value of t falls within the range, $0 \leq t \leq 1$, then the line intersects the plane. The expression for t can now be inserted into the parametric equation (see Eqn. 6.8) for the segment to find the actual intersection point Q:

$$S(t) = S(\frac{(d - n \cdot A)}{(n \cdot (B - A))}) = \frac{A + [(d - n \cdot A)}{(n \cdot (B - A))](B - A)} \tag{6.14}$$

A plane is infinite. Therefore, it is not merely enough to find the intersection of the line with the plane, but it is essential to find which triangle on the target mesh contains the intersected plane. Thus a check is required to see if the point of intersection is inside the required triangle or not. Unlike an iterative algorithm like the Jordan Curve Theorem [1] (Hatcher, 2002) a simpler and efficient vector math solution is the *Same Side Technique* and works well with triangles and is also extensible to convex polygons and iteration needs to be done only for the

[1]Essentially, it says that a point is inside a polygon if, for any ray from this point, there is an odd number of crossings of the ray with the polygon's edges.

number of vertices (Scott, 2012). If the mesh used is triangulated, the number of iterations remains 3. The mesh used in the demonstration has convex polygons with different number of sides.

Let $A(x1, y1, z1), B(x2, y2, z2), C(x3, y3, z3)$ be the vertices of the triangle ΔABC and Q denote the intersection point on the plane. Then average of the vertices can be defined by the point, P_{avg} where

$$P_{avg} = \{\frac{(x_1 + x_2 + x_3)}{3}\}, \{\frac{(y_1 + y_2 + y_3)}{3}\}, \{\frac{(z_1 + z_2 + z_3)}{3}\} \tag{6.15}$$

Now let $\overline{V}_1$,$\overline{V}_2$,$\overline{V}_3$ denote the vectors from A to B, A to P_{avg} and A to Q.

$$\overline{V}_1 = B - A \tag{6.16}$$

$$\overline{V}_2 = P_{avg} - A \tag{6.17}$$

$$\overline{V}_3 = Q - A \tag{6.18}$$

Taking the cross product of the vectors,

$$\overline{V}_{n1} = \overline{V}_1 \times \overline{V}_2 \tag{6.19}$$

$$\overline{V}_{n2} = \overline{V}_1 \times \overline{V}_3 \tag{6.20}$$

Now P_{avg} is the centroid of the ΔABC. In order for the intersection point to be inside the triangle, it has to be on the same side of the edges of the triangle as the centroid (Scott, 2012). That condition is satisfied if and only if the product of the z components of the vectors $\overline{V}_{n1}$and $\overline{V}_{n2}$ is a negative value ie.

$$(\overline{V}_{n1} \cdot z)(\overline{V}_{n2} \cdot z) < 0 \tag{6.21}$$

This condition has to be satisfied by all the three edges of the triangle. Then it can be concluded that the intersection point is within the triangle.

6.4 Results of the Current System

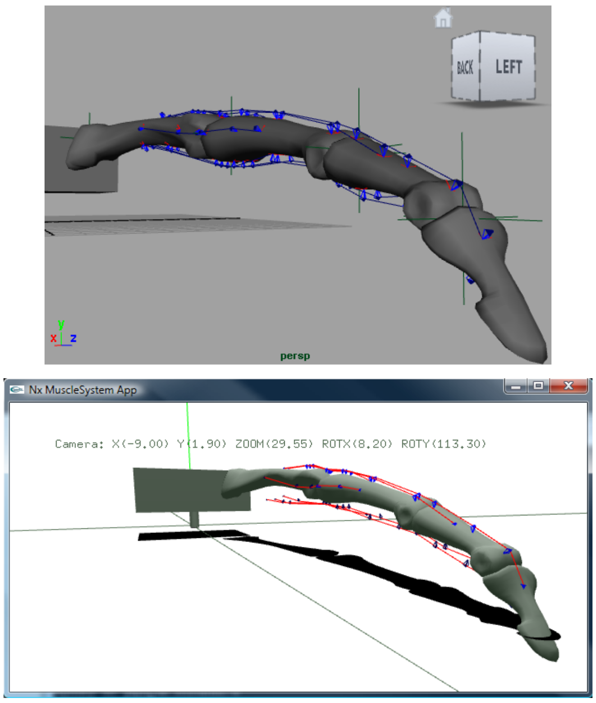
NVidia's real-time physics engine PhysX was used for all the simulations in the implementation. The development platform was a Windows 7 laptop with an Intel i3 2.4 GHz processor and a system memory of 4 Gb equipped with an NVidia GeForce GT 415M graphics card with 1 Gb of video memory. Early development was also done on a Windows Vista laptop with an Intel Core Duo 1.86 GHz processor with 2 Gb of memory and an NVidia GeForce Go 7300 graphics card having 400 Mb of video memory.

In preparation for neural control, the forefinger is rigged using the muscle locators in Autodesk Maya (see Figure 6.6). Using anatomical and other references (from Gray (2006), Valero-Cuevas and Lipson (2004), Valero-Cuevas et al. (2007)), the *tendinuous hood* of the fingers was re-constructed in a simplified manner for a real-time context.

The finger muscles were broken down into 8 muscles based on 4 movement groups (see Table 6.1)

Table 6.1: Movement vs muscles

Finger Movements	**No: of Muscles**
Flexion	3
Extension	3
Abduction	1
Addiction	1



Camera: X(-9.00) Y(1.90) ZOOM(29.55) ROTX(8.20) ROTY(113.30)

Figure 6.6: The images show the muscle rig laid out in Maya (top) and the dynamic muscle system created in PhysX (below) using the rig information. Extensor, flexor, abductor and adductor muscles are added.

During the configuration of the physical hand model in PhysX, the metacarpal phalanx of the forefinger and the cuboid rigid body (representing the wrist) it is attached to are configured as *kinematic* (see Figure 6.7). Kinematic bodies in

PhysX are not dynamic (impervious to forces), retain only position in space and have no velocity component (Ageia, 2006). The position of a kinematic body in space can be moved only by directly changing it and not through the action of forces. The kinematic constraint was imposed to reduce DoFs and number of muscles and also to eliminate the need for simulating antagonistic muscle action (to keep the phalanx steady). If the metacarpal phalanx and the cuboid body were to be made dynamic, then additional muscle structures would be necessary to counteract the forces exerted by the muscles of the proximal, inter and distal phalanges. The training sample motion lacks arm motion thus preventing training of any upper wrist muscles. Moreover, the metacarpal phalanx only articulates at the wrist joint, which in this case is fixed.

Based on the structure of the human hand, there are two ways of creating the neural network controller. They are:

- Single neural network controlling all the muscles as a group
- Individual neural networks for each finger.

The method chosen directly affects training time, performance efficiency and memory usage. The first method has obvious disadvantages over the other. With each finger having 8 muscles that control the movements, there are 40 muscles in the model which suggests that there would be 40 output neurons mapped to the respective muscles. The input to the network is in the form of three dimensional vectors specifying the orientation of the skeleton. There are three vectors per chain having three components each of which are fed into the network via individual input neurons. In a fully connected three layer network, each neuron in a preceding layer is connected to every neuron in the succeeding layer. The number of connections defines the number of weights in the network. Table 6.2 shows the number of connections which in turn gives the size of the chromosome. The table calculates the chromosome sizes for the two possible controller generation methods stated above. The example uses two neural networks with 47 hidden neurons. The total number of muscles for all five fingers is 40 (based on the movement groups for one finger).

The number of connections is given by the following formula:

$$C = \sum_{i=0}^{l-1} (N_i)(N_{i+1}) \tag{6.22}$$

where N is the number of neurons in the layer i.

There are 8 muscles mapped to 8 neurons. Research confirms the existence of independent neural networks controlling individual digits to coordinate finger

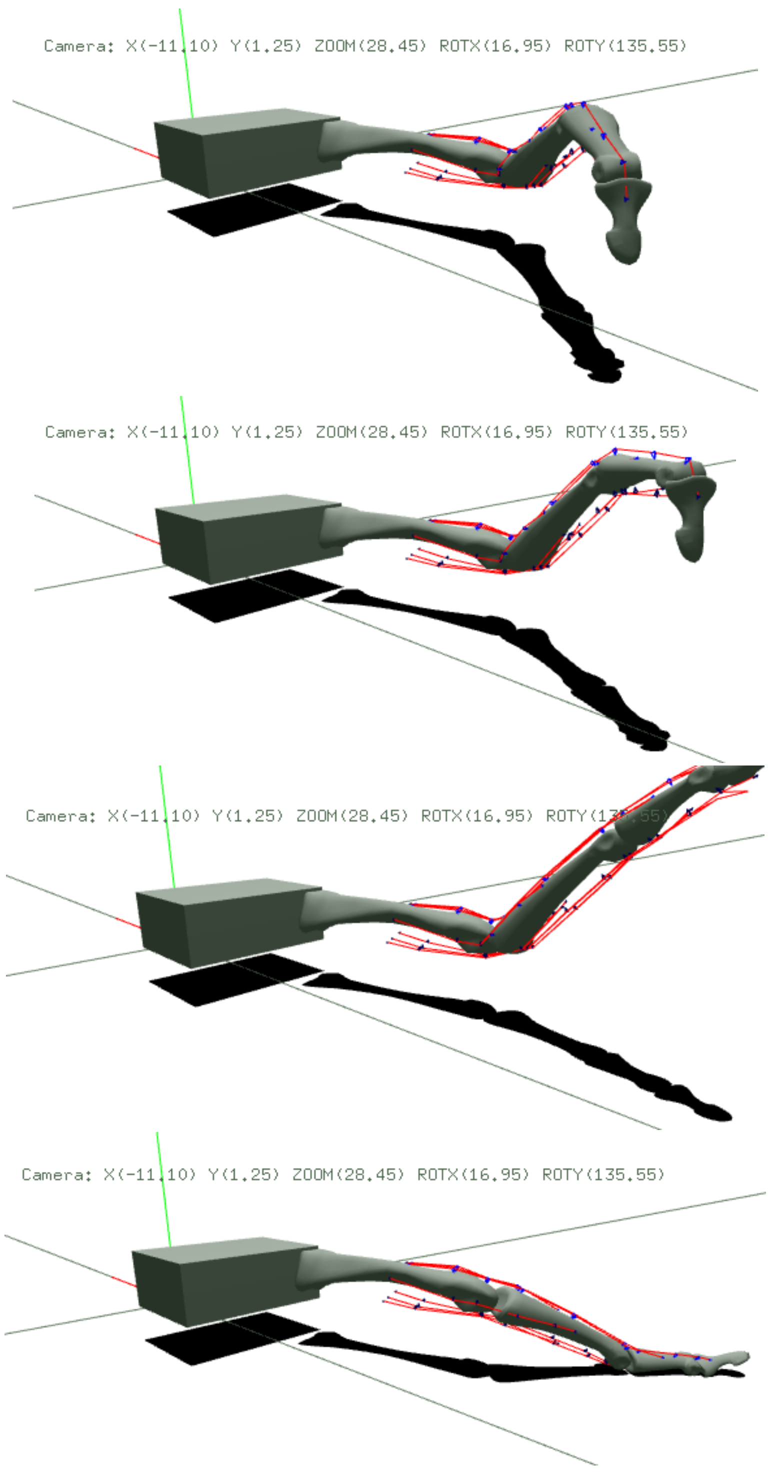
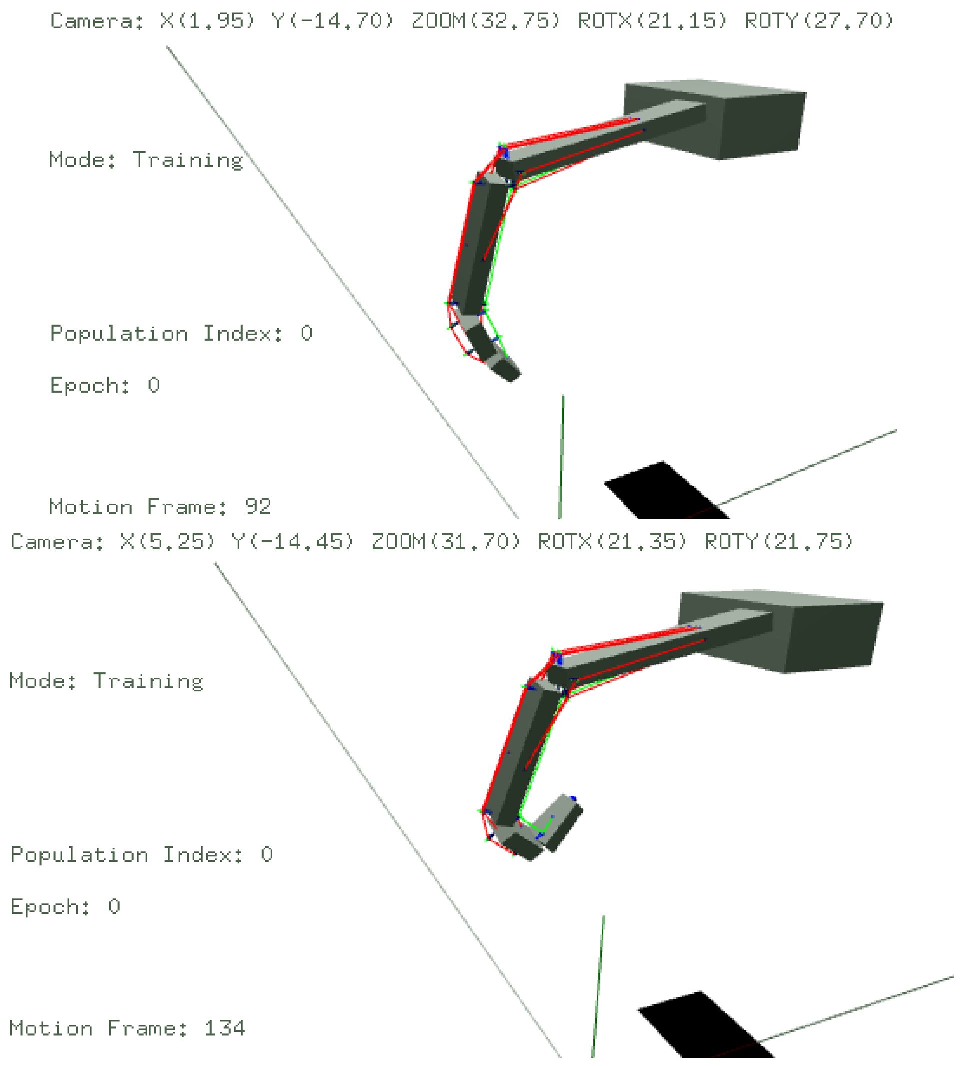

Camera: X(-11.10) Y(1.25) ZOOM(28.45) ROTX(16.95) ROTY(135.55)
Camera: X(-11.10) Y(1.25) ZOOM(28.45) ROTX(16.95) ROTY(135.55)
Camera: X(-11.10) Y(1.25) ZOOM(28.45) ROTX(16.95) ROTY(135.55)
Camera: X(-11.10) Y(1.25) ZOOM(28.45) ROTX(16.95) ROTY(135.55)

Figure 6.7: Frames taken from muscle twitching. Finger joint constraints are not imposed and the muscle activation is not coordinated (absence of neural control) thereby creating random movements.

forces during grasping tasks (Burstedt et al., 1997). Based on those findings and also based on the duplicative nature of the musculotendon layout on each finger, individually training each finger is preferred. The laterality in both the dynamic



```
Camera: X(1.95) Y(-14.70) ZOOM(32.75) ROTX(21.15) ROTY(27.70)
Mode: Training
Population Index: 0
Epoch: 0
Motion Frame: 92
```

```
Camera: X(5.25) Y(-14.45) ZOOM(31.70) ROTX(21.35) ROTY(21.75)
Mode: Training
Population Index: 0
Epoch: 0
Motion Frame: 134
```

Figure 6.8: Finger flex using the muscle system described in Chapter 4.

Table 6.2: Neural network and Chromosome statistic

Method	**Input**	**No: of Muscles**	**Hidden**	**Output**	**Chromosome Size**
Single ANN controller for the whole hand	135	40	47	40	8225
Individual ANN controller for each finger	9	8	47	8	799

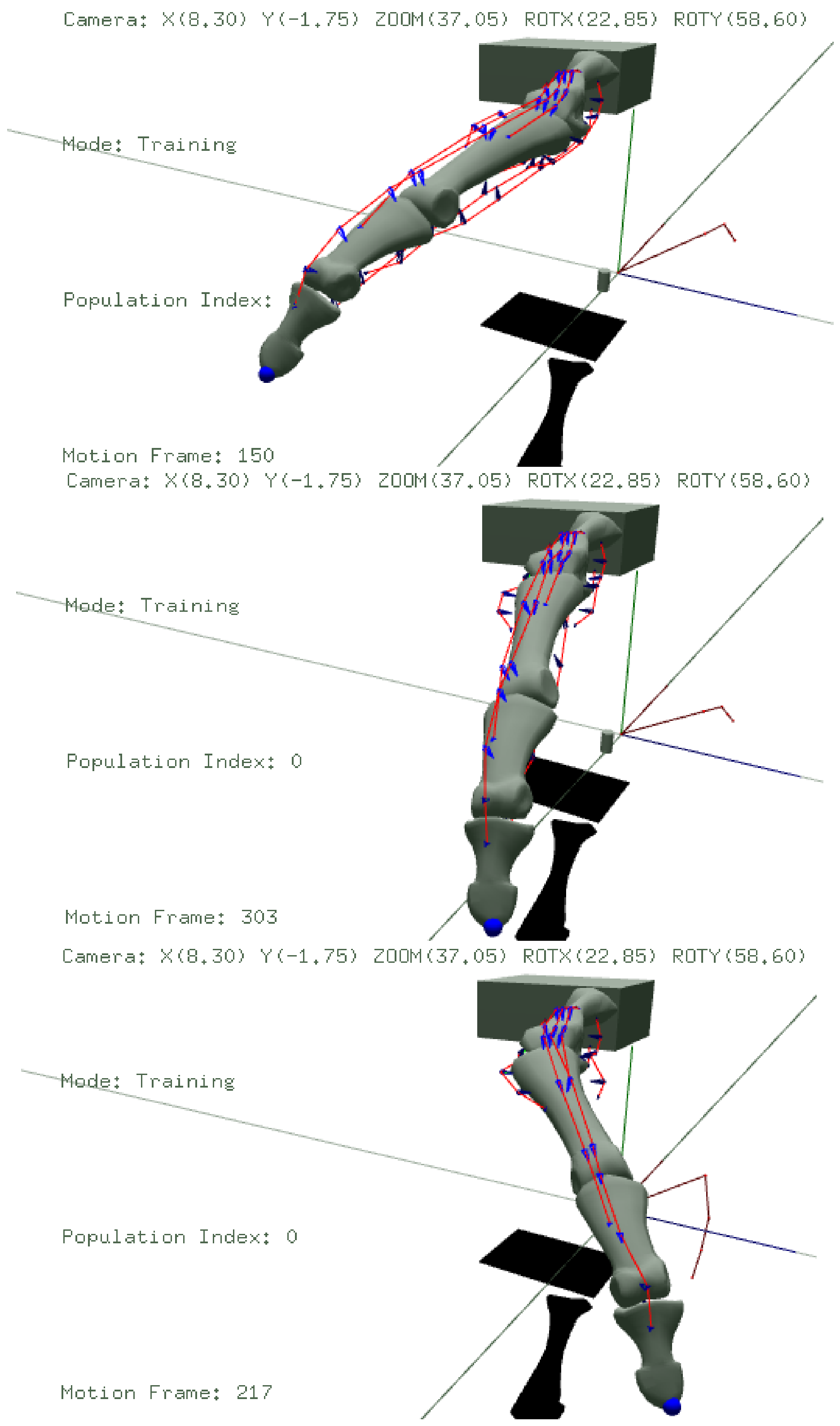

Camera: X(8.30) Y(-1.75) ZOOM(37.05) ROTX(22.85) ROTY(58.60)
Mode: Training
Population Index:
Motion Frame: 150
Camera: X(8.30) Y(-1.75) ZOOM(37.05) ROTX(22.85) ROTY(58.60)
Mode: Training
Population Index: 0
Motion Frame: 303
Camera: X(8.30) Y(-1.75) ZOOM(37.05) ROTX(22.85) ROTY(58.60)
Mode: Training
Population Index: 0
Motion Frame: 217

Figure 6.9: The forefinger bone model with the muscle system performing abduction movement.

hand model and the motion capture model used in training are different - the dynamic hand model is a right hand while the motion capture model is a left

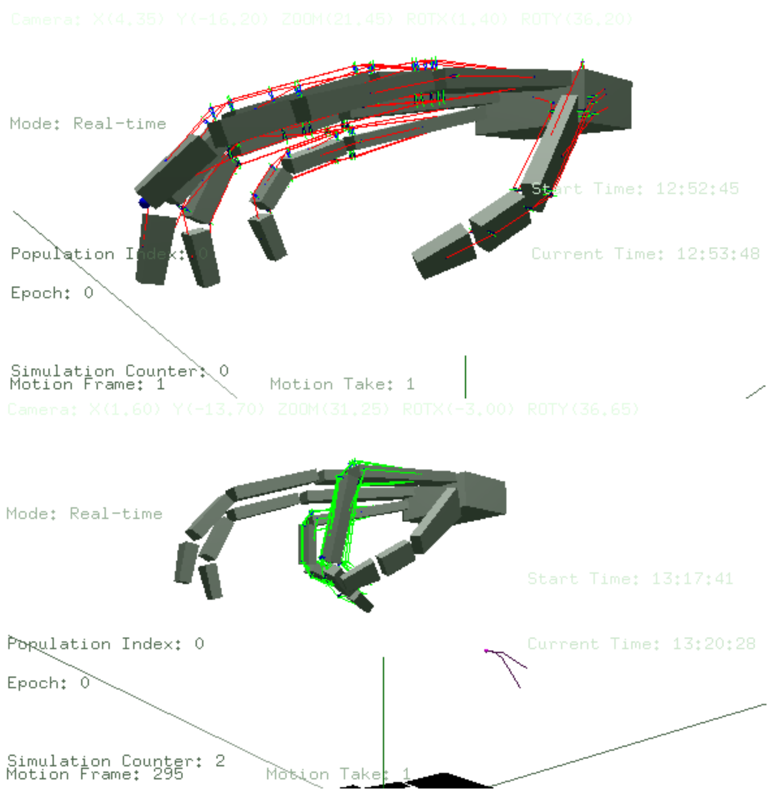


Figure 6.10: The fully dynamic rigid body hand model with the muscles laid out for each finger (top) and the forefinger and the little finger executing a motion (bottom).

Table 6.3: Forefinger muscles to ANN output mapping. The muscle type is also listed.

Output Neuron Index	**Forefinger**	**Muscle type**
0	muscleGrpdistalEXT	distal extensor
1	muscleGrpinterEXT	inter extensor
2	muscleGrpproxiEXT	proximal extensor
3	muscleGrpdistFLEX	distal flexor
4	muscleGrpinterFLEX	inter flexor
5	muscleGrpproxiFLE	proximal flexor
6	muscleGrpABD	abductor
7	muscleGrpADD	adductor

hand.

The generalising property (synaptic plasticity) of the neural network allows it to reproduce motion that is absent from the original training samples. In order to adapt traditional motion capture data for input as training data for the network, relevant data is extracted from the motion samples. The motion capture data used in this project is in the Autodesk FBX animation format. The relevant frame data is read from the FBX data file and converted to a simple text format. The details of the format is available in Appendix C.

The research began with the following questions,

- Is it possible to create a functional mapping between an end motion and the underlying muscle activations without resorting to complex inverse dynamic calculations and activation dynamics?
- What are the various techniques that can enhance the training model of the network?

Muscle activation and force quantification and thereby prediction occupies a vast search space. There is no upper or lower limits that define a range. GAs are well suited to find the correct solution from large spaces. This is largely due to the parallel nature of GAs. In the context of finding muscle activations, the process requires massive iterations using very large chromosomes. Chromosome size is largely dependent on the number of neurons in the artificial neural network which directly affects the size of the encoding chromosome. This methodology of finding the correct muscle activations from a large search space uses two black box systems namely, the GA and the ANN.

The mechanism by which they converge on solutions is little understood. Previous research attempted to theoretically analyse genetic algorithms and explain the reason why they work (Senaratna, 2005). A key example is the *Schema Theorem* which was a stepping stone to succeeding mathematical models that tried to clarify certain fundamental questions regarding the role of the genetic operators in the GA, namely, crossover and mutation (Senaratna, 2005). The argument, famously known as the Crossover-Mutation debate, is as yet unresolved. The success of GA depends on probabilities on how a schema of high fitness is transferred into future generations by surviving the genetic operations performed on it.

The presence of two black box techniques made solution convergence difficult. The reasons are:

- Number of training samples: Effect of the number of training samples varies with the problem. It depends on the type of feature one expects the network to extract from the training samples. In the system under discussion, the network is expected to produce real-valued activations that corresponds to time varying poses and generalise to unknown poses. Hence, the better variation in poses, would allow the network to generalise.

- Population size: Initial population size determines the sampling size from the search space. The bigger the population size, the higher the probability of finding the solution from the search space, but with an increase in time.
- ANN hidden layer size: As mentioned in the previous chapter, the hidden layer size is a heuristic. Initially, the number of neurons in the hidden layer was judged by taking the average of the number of layers in the input and output layer neurons. But during later stages, a brute force manual method of trial and error was performed. There is always the case of over-fitting if there are too many neurons and under-fitting if there are too less. Standard back-propagation neural networks normally start with a high number of hidden layer neurons. But since the learning model is a genetic algorithm, size of the encoding chromosome also needs to be considered.
- Choosing the best convergence strategy: In an evolutionary system, convergence strategies can vary from the type of genetic operators used to the variations in the actual evolution scheme used. For example, pure GAs (using standard genetic operators) or using only the mutation operator or using a completely different strategy like Hill Climbing.

Encoding scheme for network evolution was strictly real valued instead of the standard binary string format popular with GAs. There were a few factors that influenced the choice of the encoding scheme. They are:

- Loss of precision in conversion: Data is lost during conversion to binary format, specifically in deciding the number of bits to represent the real value.
- Length of chromosomes: Binary string chromosomes tend to be large. The bit string length can be compounded due to the fact that in evolutionary neural networks, synaptic weights form the schemata and the number of weights are dependent on the number of connections. A binary string representation of real-valued weight genes can be impossibly large. Such large sized chromosomes can be detrimental to the GA convergence.
- Use of specialized genetic operators: Real-valued chromosomes allow the use of specialized genetic operators like the mutation operator using the variance vector, which is not possible with the binary string chromosomes.

The method that successfully produced a solution was the time series predictor network. From a single input sample, the predictor was able to provide a solution that generalised to three available unknown samples.

The neural control system described explores the learning and generation of muscle activations with multiple constraints imposed. Using motion-training samples, the virtual nervous system is taught to produce synergistic muscle activations to produce motion theoretically matching the training samples. A mapping between a surface motion template and virtual muscle system is attempted through a stochastic exploration of muscle activation space. In order to achieve this goal, the system is presented only with the motion template or sample. There are no electromyographic data recordings to enable parameter extraction for neural learning. This limitation is circumvented through the use of *evolutionary neural networks*. Network weights are evolved through the use of Genetic Algorithms. In the absence of quantifiable data that can help to derive an error between desired and actual output, the system resorts to a random optimisation search method that narrows the search using an objective function. Four different neural input models were used, with each model building on the limitation of the previous model (see Table 6.4).

Table 6.4: Various neural models used.

Neural Network type	**Result**
Standard Feedforward without state feedback	Unsuccessful
Discrete Time recurrent	Unsuccessful
Feedforward with state feedback	Unsuccessful in motion generation, but successful in single pose retention
Time series prediction	Successful in creating similar motion to the input sample

Stochastic optimisation methods like the GA, that behaves as the learning model for an artificial neural network, can successfully extract coordinated muscle activations from purely surface motion is. The speed and efficacy of the convergence is purely dependent on the ability of the objective function to filter out the bad solutions from a massive solution space. The objective functions implemented as a part of the system are detailed in Chapter 5.

The original primary objective function used was a pose matching algorithm that calculated the root mean error from the differences in the vector orientations. The vectors were calculated from the skeletal segments of the motion sample as well as the dynamic rigid body linkage that incorporated the virtual muscles. Pose matching is the simplest and most direct method to compare samples of skeletal motion. But pose matching works best with kinematic motion samples where key frames define the motion and where the interpolation value for the motion samples are equivalent. Physical simulation uses discrete time steps. When comparing key frame data with physically simulated motion data, the interpolation value and the time step are different. This means that there is no one-to-one relation between the frames compared in the motion sample and the output simulation. But this disparity is overcome by the iterative fitness calculation of the GA, because the objective is to compute muscle activations that allow the simulated motion to match the input frames.

A neural model with a state feedback failed to produce the required motion. The ANN was a standard feedforward network that accepted a set of 3D vectors that defined a skeletal pose and also a second set of 3D vectors that defined the output skeletal pose that was the feedback. A single frame (pose) from the motion sample was given as input to the network. Even though unsuccessful in finding the converged network that produced matching activations to the input sample, the network learned to produce the single input poses. In order to achieve it the network generated simultaneous activation of agonistic and antagonistic muscles on the rigid body linkages.

The most crucial feature that the simulation depended on was the muscle system. During simulation, the muscle system proved capable of generating a wide variety of motion that remained within the physical constraints imposed. The neural system during the course of learning generated complex activations that were otherwise impossible to generate manually.

Once the network is trained and is capable of generating coordinated muscle activations, the scaling force can be modified. This enables the appendage to execute an input motion in lesser time. In other words, the speed of an action is controlled by the force input.

The population size used was 50 with the number of hidden neurons at 30. The GA was able to produce a good solution in 32 generations over a time period of approximately 6-7 hours. Due to penalty fitness, the fitness is negative, with higher fitness nearing 0 (see Figure 6.11).

The size of the window or the spacing between the input sequence is a crucial factor in the network's pattern recognition training. In the motion training, with a sample having 308 frames, setting wide spacings of 15 frames failed to produce any convergence even after 80 generations of genetic evolution.

6.4.1 Motion

Control is always an issue in physics-based character animation. The animator requires absolute control over the characters in an industry where the director reigns supreme where character acting is concerned. Kinematic techniques often

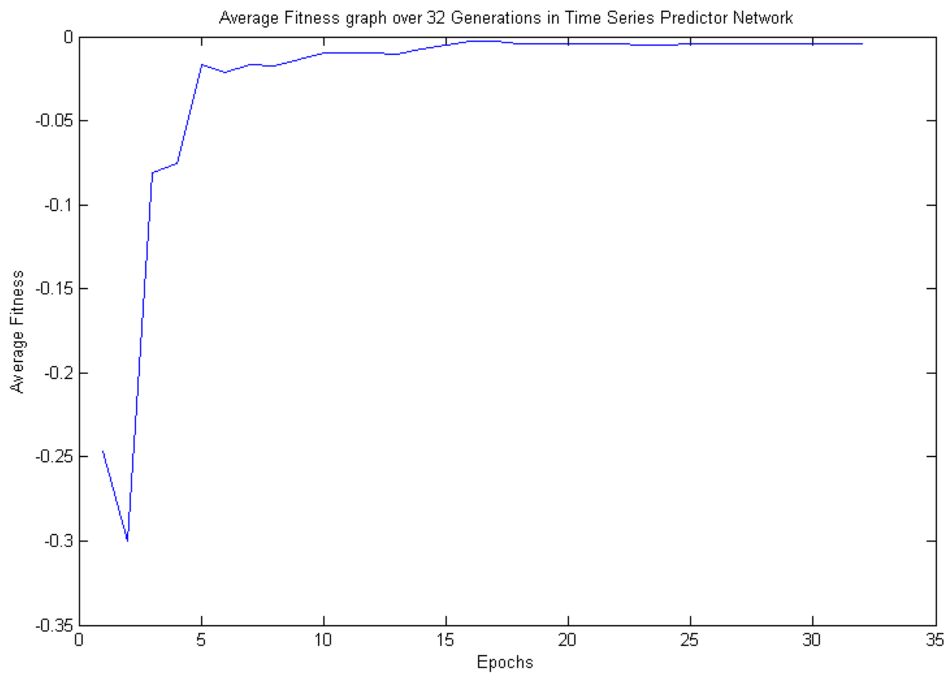


Figure 6.11: The average fitness of each individual in the population over 32 generations in the time series prediction.

provide fine control of computer generated characters for the animators, whereas dynamic techniques often require intuitive representation of the various simulation parameters which the animator can easily understand and conceal as much of the mathematics involved as possible. In spite of that control problems arise when the character moves or behaves in ways unintended by the animator. Making the characters anatomically similar and biomechanically function like the real-world counterparts increases the complexity in control. Dynamics is based on forces acting on bodies and an increase in the number of force actuators (muscles) increases the number of parameters that require tuning to produce the motion the animator desires. Separating forces from activation dynamics allow coordinated muscle action to occur independently of the force generated. This is the idea behind *force scaling*. Section 4.3.4 in Chapter 4 explains this simple concept in detail. Force is not an easily quantifiable parameter which makes it very difficult to manipulate the various muscles on the character to fine tune motion. The complex task of coordinating muscle forces so that the limb reaches its end goal is achieved through the neural controller using force magnitude as a user specified parameter. This allows the user to define how slow or fast an action needs to be. Force scaling is an important aspect of the system because the animator is saved from the difficult task of providing various forces for the muscles and instead controls timing of the motion by providing a singular value for the muscle group.

Research from neuroscience (MussaIvaldi and Bizzi, 2000) shows similarity between force scaling and biological dynamic computation by the central nervous system. The experiments performed in (MussaIvaldi and Bizzi, 2000) suggests a vector summation of the vector field generated by the central nervous system. These forcefields represent motion primitives and the CNS, through a superpositioning of these various vector fields, produces a wide range of motion. The summation of the vector field is used in MussaIvaldi and Bizzi (2000) to replace the generic torque function in the inverse dynamic problem and each field is tuned by a non-negative scalar value representing a *supraspinal command* [2]. Reformulating the inverse dynamic problem suggests that joint torques are produced through the modulation of the force fields. (MussaIvaldi and Bizzi, 2000) proves that modulating the scalar coefficients in the reformulated inverse dynamic equation produces intended minimum jerk movements. But it is still unclear as to how the central nervous system derives the tuning coefficients purely based on the desired movement.

6.4.1.1 Motion Generated versus Muscle Activation

The motion generated by the system is remarkably similar to the input motion. In this section, a comparison is made between the input motion and the corresponding muscle activations generated by the neural system. This gives an idea on the pattern of activation/deactivation of specific muscles during the course of a motion. The number of muscles in the human body are more than the number of DoFs, which is called the *redundancy problem* (Lee et al., 2009). This indicates that there is a probability that a specific motion can be generated at different points in time using a different set of muscles. This is decided by the way the CNS uses muscle synergies in order to solve the redundancy problem (Gazzaniga, 2004). The CNS groups muscles based on tasks in order to reduce control signals. The exact nature of synergy is a continuing subject of research.

In the implemented system there is constant muscle activation at every time step of the simulation. The observed motion has a phase difference from the input motion due to certain issues which are examined in section 6.5. A comparison between activations and motion transitions can be performed to see how activations vary with changes in the direction of motion. Figure 6.12 is the generated muscle activation graphs for the distal phalanx of the forefinger for two different sets of input motion. The highlighted section is enlarged in Figure 6.13.

[2]Supraspinal commands - Motor or postural commands that occur from above the spinal column

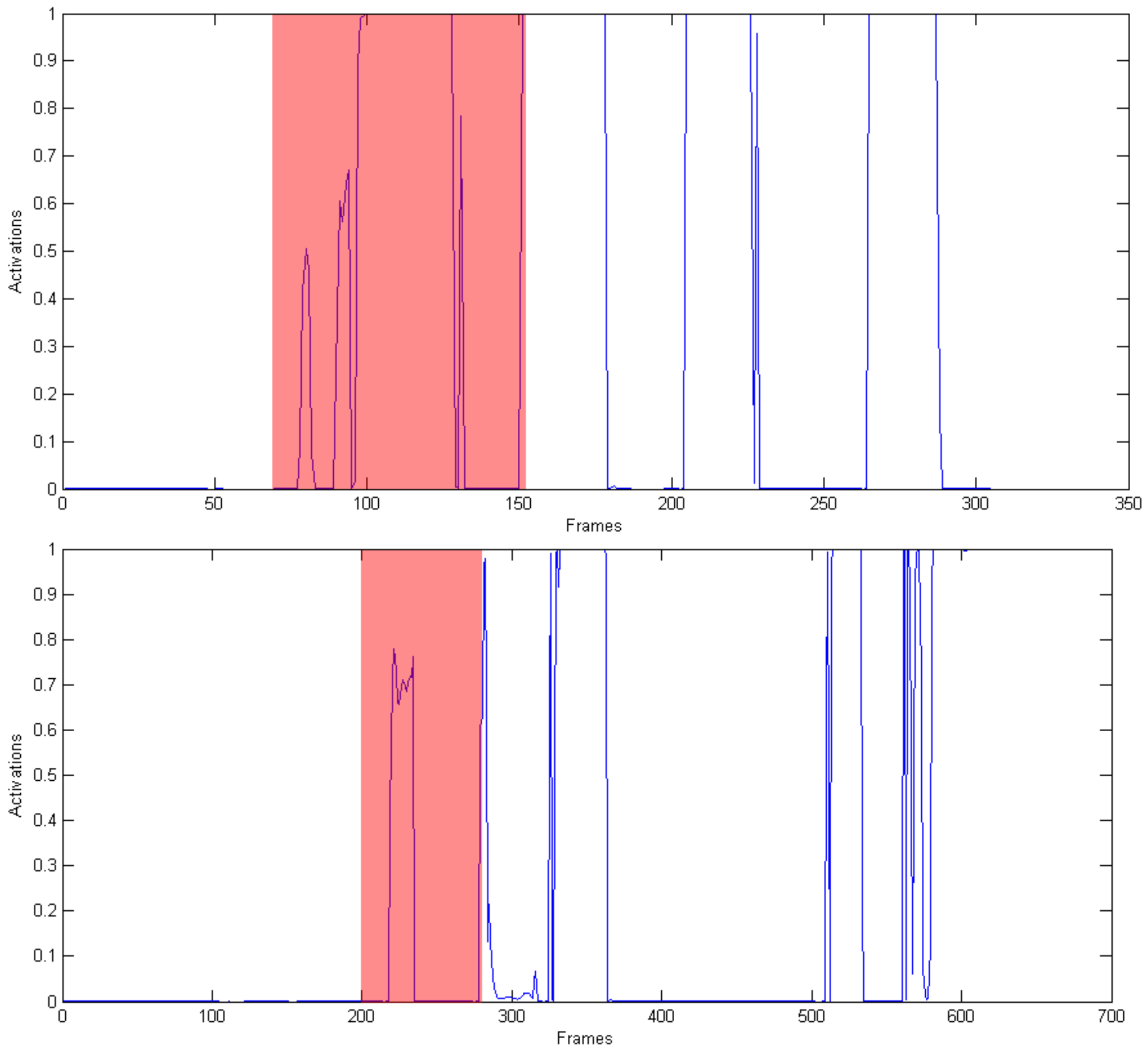
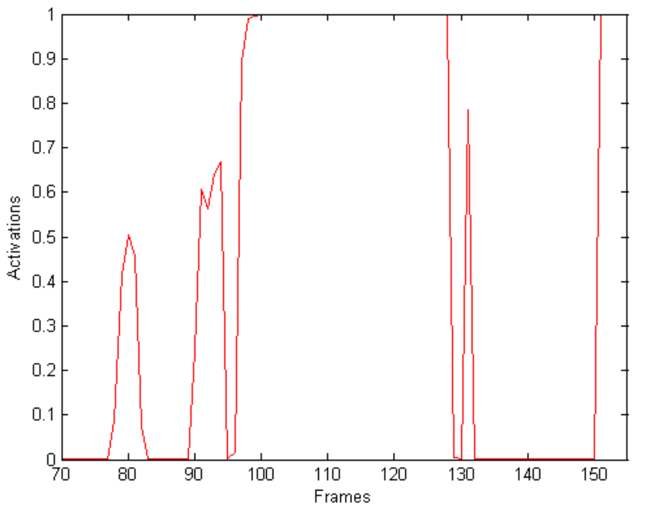


Figure 6.12: The top image is the muscle activation-time graph of the flexor tendon of the distal phalanx of the forefinger, for a motion sample of 305 frames. The image below is the muscle activation-time graph for the same but using a motion sample consisting of 608 frames.

Two different motion samples (Motion Clip 1 and Motion Clip 2) are used to show the activation/deactivation nature of the involved muscles during specific actions like flexion or extension. Figure 6.12 and the scaled segment (see Figure 6.13) of the same, shows the activations of the flexor tendons in the distal phalanx. The top image in Figure 6.13 shows the activations for a frame range of 70-150 for Motion Clip 1. The activations for the muscles are in the interval (0,1). The peaks denote activations for the flexor tendon that causes the flexion of the finger.

In the bottom image in Figure 6.13, the flexion starts from frame 213 for Motion Clip 2. This matches the flexed finger pose for those frames in both the generated and input motion. From frames 221-235, the graph shows fluctuation,



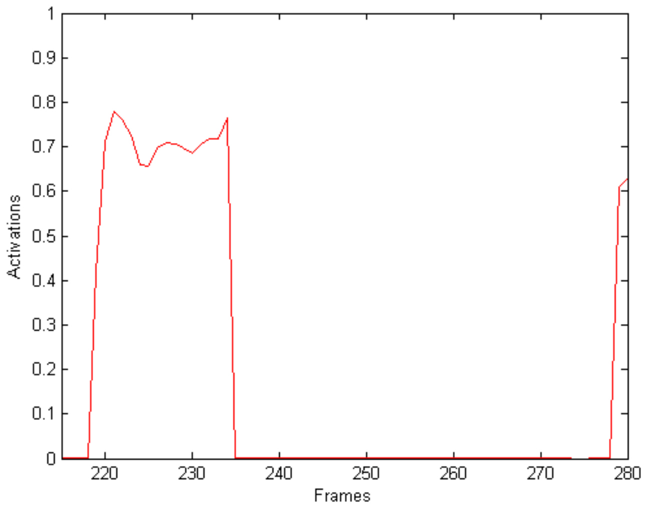
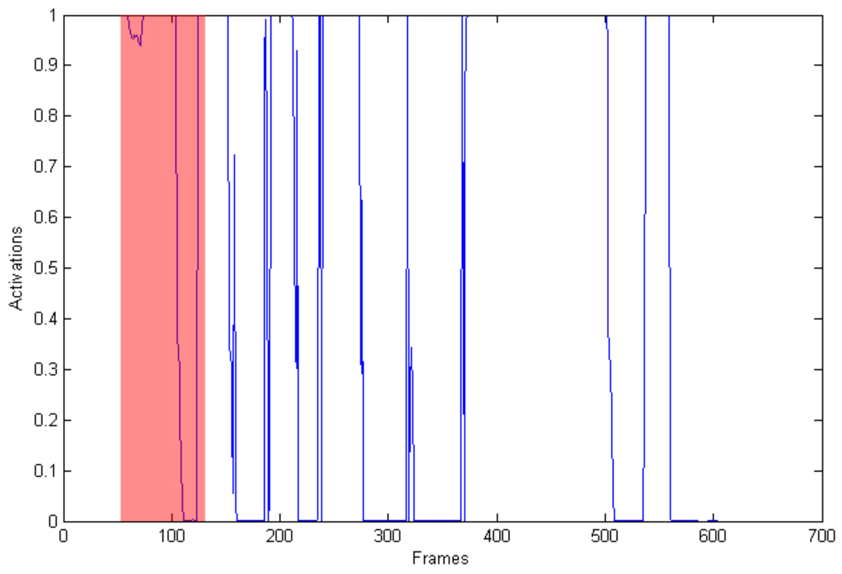


Figure 6.13: Both images show activations/deactivations of the flexor tendons of the distal phalanx for two different motion samples. The top image is from Motion Clip 1 and the bottom image is from Motion Clip 2. These images represent the re-scaled portions highlighted in Figure 6.12.

which reflects a positional turbulence in the motion sample. From frame 236, there is a sudden drop in activation from 0.78 to 0. In the motion sample, this indicates a full extension of the finger that do not require the activation of the flexor muscles. Subsequently from frame 276, there is again a sudden flexion activity. The corresponding motion frames can be seen in Table 6.5.

The graphs in Figure 6.14 show the activations and deactivations of the extensor tendon of the distal phalanx. Between frames 58-76 approximately in the re-scaled graph in Figure 6.14, it can be seen that there is a "W" shaped variation in the activation. In the input motion, this corresponds to a perturbation of the finger during full extension.

This pattern of activation and deactivation of muscles correspond to the



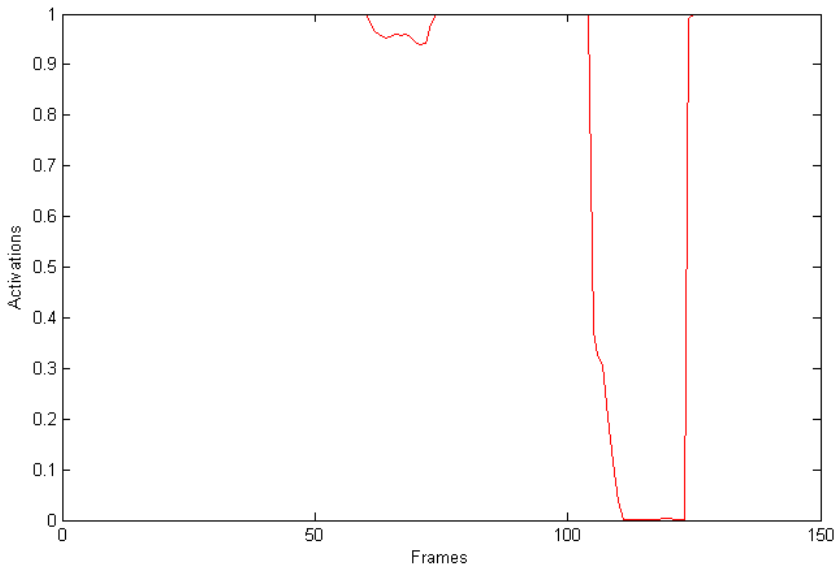
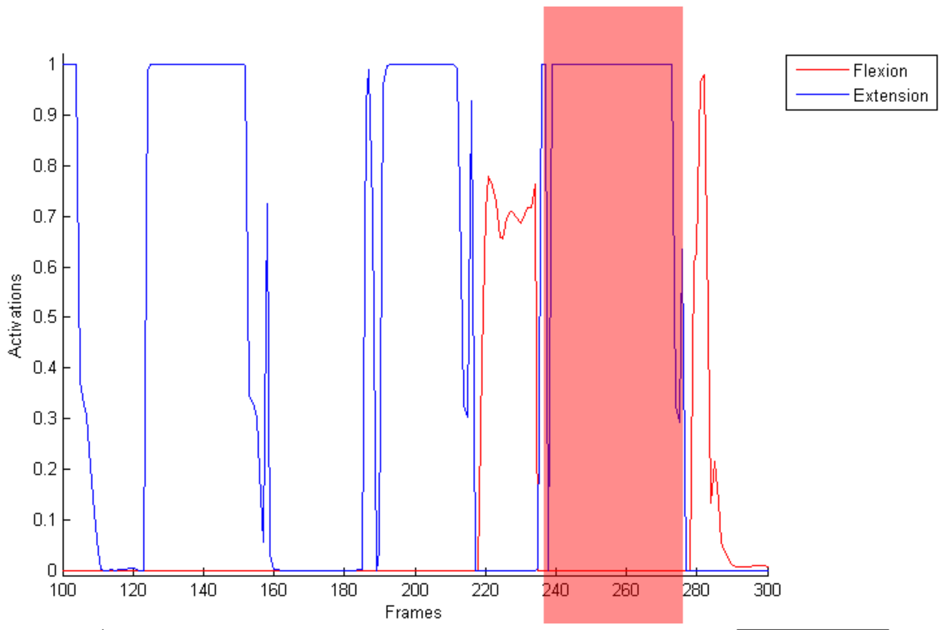


Figure 6.14: The activations/deactivations of the extensor tendon of the distal phalanx of the forefinger and the re-scaled portion of the highlighted section. Both images show activations from Motion Clip 2



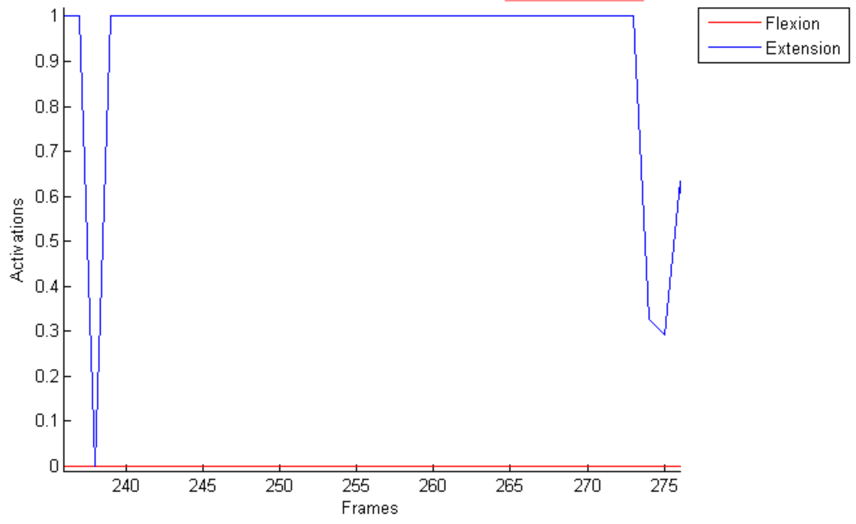


Figure 6.15: On the top is the superimposed image showing the activations of both flexor and extensor tendons of the distal phalanx of the forefinger. The bottom image shows a re-scaled portion of the highlighted section in the top image.

direction of motion and the neural controller schedules activations on a task basis. Muscles in close proximity to the direction of motion are activated while those situated away are deactivated. The neural controller behaves similar to the CNS in the scheduling and creating muscle synergies as per Gazzaniga (2004). Synergestic activity is evident in the behaviour of the controller because activation of flexor muscles during the transition from extension to flexion varies depending on the amount of flexion for each phalanges. The activity of both flexion and extension between frame period 236-276 can be seen in Figure 6.15. Thus it can be clearly seen that there is a pattern of activation and deactivation for the flexor and extensor tendons between the specified frames. The flexor tendon deactivates during the extension of the distal phalanx.

It is also to be noted that there is a sharp decline in the activation of the extensor tendon at frames at 235 and 273 approximately. But at the same time there is an absence of corresponding activation in the flexor tendon at these frames. This is primarily due to the activation of the other flexor tendons for either the proximal or inter phalanges. This is further evidence of the synergestic behaviour of the neural controller mimicking a biological nervous system.

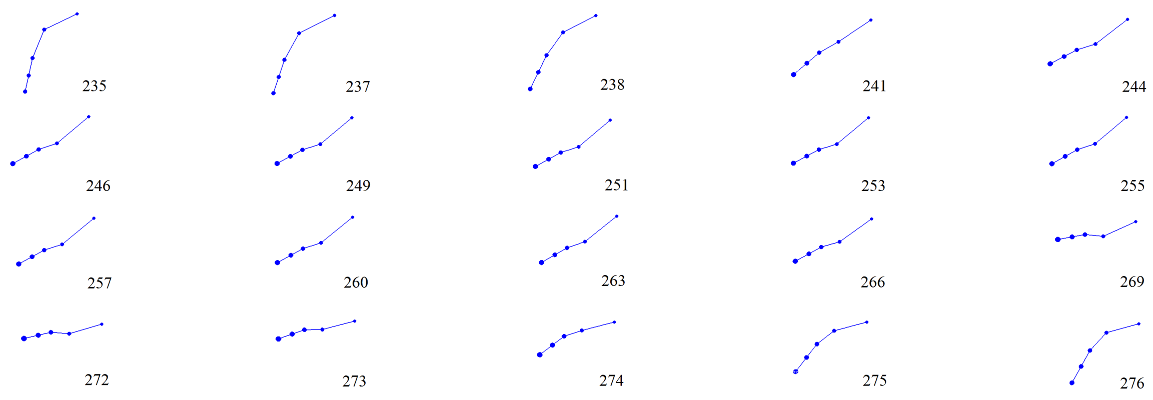
6.4.1.2 Changing the Original Motion Pattern

During training of activation, the forces for each muscle are uniform, having a uniform value for the magnitudes. So the activations are scaled by the same amount. But by providing non-uniform force magnitudes for each muscle, it is possible to alter the original motion pattern. In a hand model though, the possibilities are limited by the limited amount of movement available (extension, flexion, abduction and adduction). But if applied to an upper body or arm or full body motion, the animator can create variations from the original motion allowing transitions.

The force scaling, since it is per muscle, can easily be converted to a user interface in an animation package like Autodesk Maya. The scaling parameters can be animated (having different values at every time step) and re-timed using animation curves.

The motion generated by the trained neural network follows the motion input with the muscles activating in a seemingly expected manner. The motion is generated through muscle activations alone and this makes it possible to introduce noise into the neuron output (see Figure 6.16). In Figure 6.16, a block representation of the hand is used. The muscles are not displayed as rendering the muscles reduces frame rate drastically.

Table 6.5: Select key frames between frame period 236-276. The forefinger extends from a flexed position and maintains the extended pose for a few frames and then flexes again.



Medically, erratic motion in the human body can occur due to many rea-

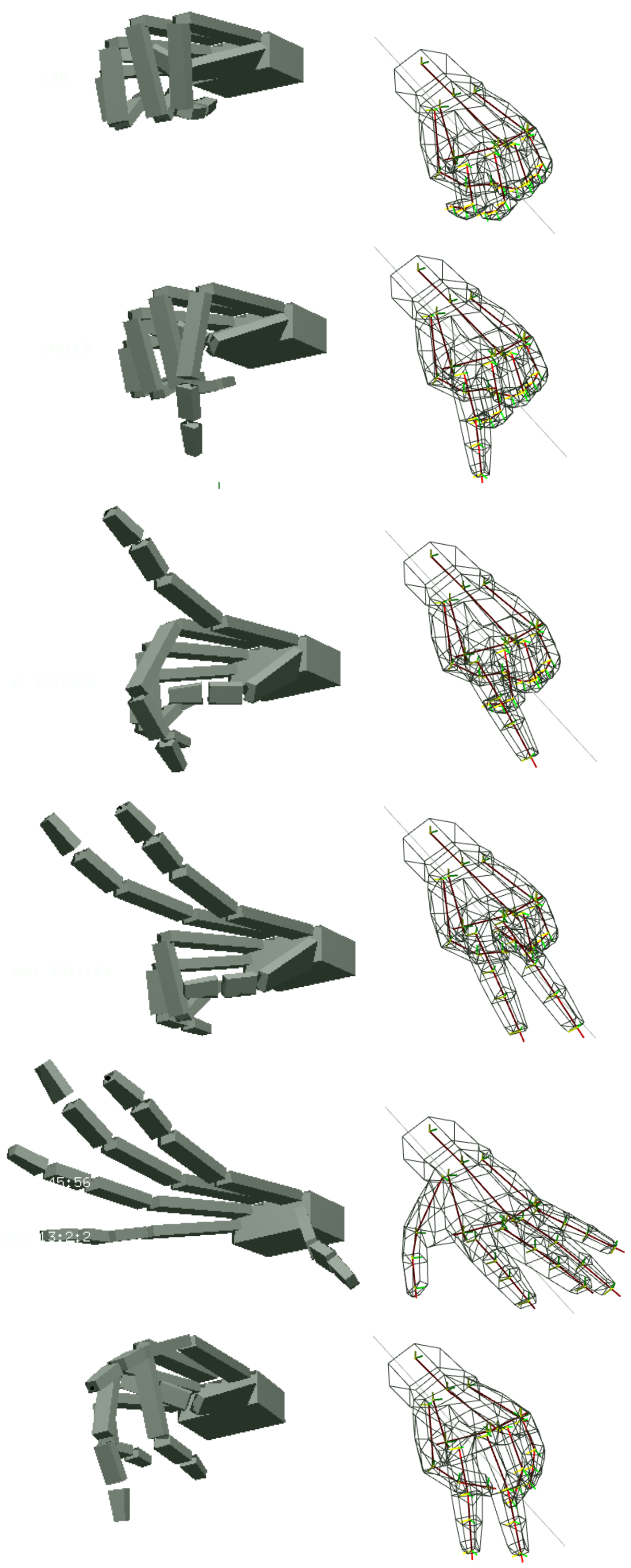


Figure 6.16: A few key frames captured from the generated animation. On the left is the physical hand model and on the right are the corresponding frames from the motion capture data.

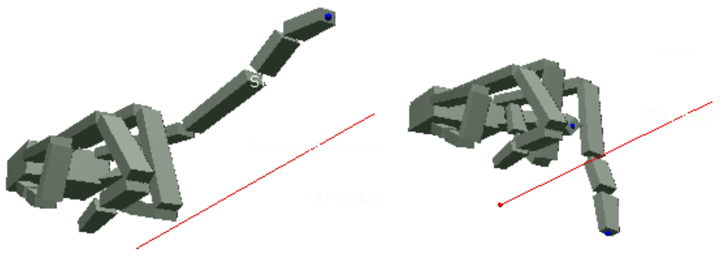


Figure 6.17: Motion frame 385. On the left, is the frame without noise and on the right is the frame with noise added on all the muscles of the forefinger.

sons. It occurs when the neural transmission of signals from the brain are not conveyed properly to the muscles. This causes specific muscles to activate with a delay or not activate at all. The limb or appendage is unable to achieve the end goal due to improper activation. There are two ways to achieve this in the current simulation - disconnecting specific muscles, thereby having zero effect of activations and by introducing noise into the neural outputs. Figure 6.18 plots path traced by the positions (x,y,z) of the distal phalanx of the forefinger before and after noise addition. The 3D vector plot shows an entire motion cycle. It can be seen in the top image the number of lines retracing the path in the course of the motion. When noise is added, the original motion structure is retained but path retrace is considerably less. Visually, the finger displayed sluggish motion with perturbations in the motion. The noise addition causes erratic activations causing the finger to be out of phase with the normal course of motion. Thus, the finger occupies a different position at the same frame as the noiseless frame (see Figure 6.17).

Figure 6.19 depicts the activation graph for the flexor tendon of the distal phalanx prior to and after training for a particular training sample. There are marked discrepancies in the activation at every frame in comparison. The range of activation is also limited in the untrained version, whereas activation occurs over the entire spectrum after training.

Capturing hand motion is difficult when compared to whole body motion capture due to occlusion and capture volume (Jin and Hahn, 2005). Conventional optical marker method is difficult to use due to the reduction in size of the subject. A sensor glove is a widely used equipment in capturing hand motion data. But there are normally medial axis offset errors, due to glove fitting problems. Jitter in the data can also arise due to sensor noise. Such sensor noise causes errors in the motion recorded. Physics engines produce motion using discrete time steps. But since the forward motion is dependent on the previous temporal states of the

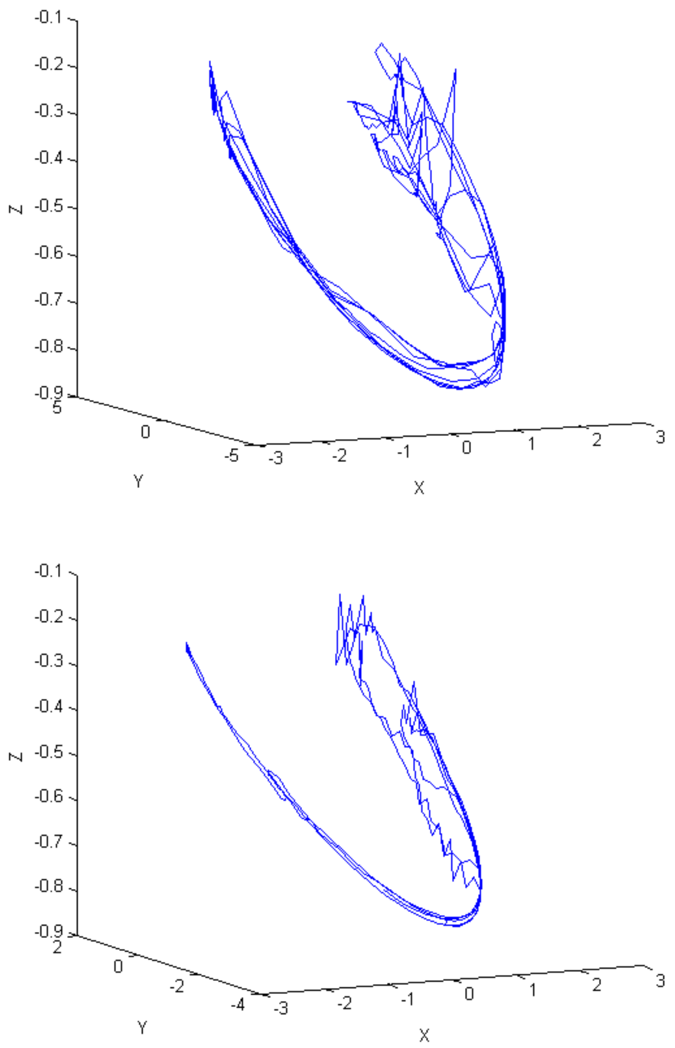


Figure 6.18: A comparison of the motion graph of the distal phalanx of the forefinger, original (top) and when noise is added(bottom).

system, the resulting motion is smooth and continuous. There are no interpolation errors which are seen in key frame animation or motion capture (recorded set of key frames). The continuous nature of the motion generated by the physics engine smoothed any motion disparities that existed in the original motion sample.

6.4.2 Procedural Control using an End-Effector

It is possible to change the inputs into a suitable animator-friendly format through modification of inputs to the network and also through re-training, which would push the technique towards its application in a production pipeline. An example of such an animator-friendly format is an *inverse kinematic* (IK) style interface. Inverse Kinematics computes the positions and orientations of each link in a chain

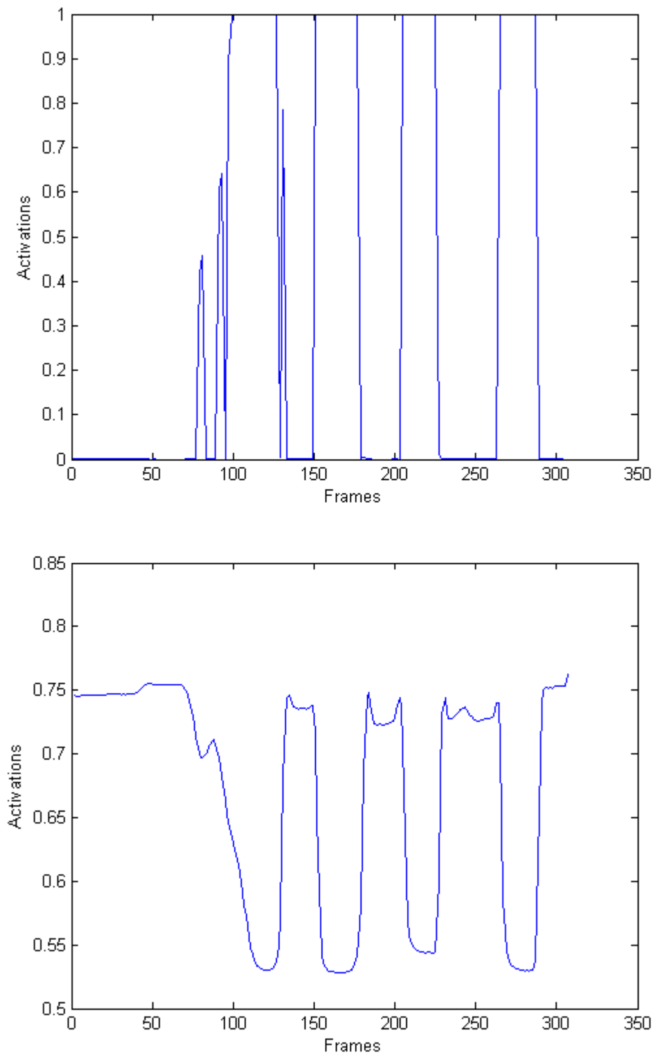


Figure 6.19: The top image is the activation pattern of the distal flexor of the forefinger after training and the image below is the pattern for the same prior to training.

in order to position the end link at a desired position. IK-based control is essential for procedural animation of characters as it is easier to specify an end position than specify the position of each link. IK algorithms are commonly used in games where motion blending is used in combination with IK to take into account the weight and dynamics of the character (Edsall, 2003). A similar end-effector style control guidance system is very useful in directing the motion of physically-based characters using muscle actuators. The ANN inputs were modified to accept three temporal states of an end effector position rather than the position of all the segments of the finger. The hidden layer neurons were set to 30 as in the previous motion training example. The forefinger motion was used to train the forefinger model. The input end-effector was calculated from the motion capture sample.

$$\overline{V}_{end} = \overline{V}_{root} + \overline{V}_1 + \overline{V}_2 + \overline{V}_3 \tag{6.23}$$

where $\overline{V}_{end}$ is the end-effector vector, $\overline{V}_{root}$ is the root joint vector of the forefinger and $\overline{V}_1$, $\overline{V}_2$ and $\overline{V}_3$ are the phalanx vectors of the finger.

$$\overline{V} = \overline{V}_{end} - \overline{V}_{root} \tag{6.24}$$

Vector $\overline{V}$ is is calculated both for the motion capture model and the dynamic model. Vector comparison is performed using Equation 4.10 in Chapter 4.

It was found that the muscle activation coordination was synchronized with the direction of movement of the end target (see Fig. 6.20). The end target position was procedurally controlled.

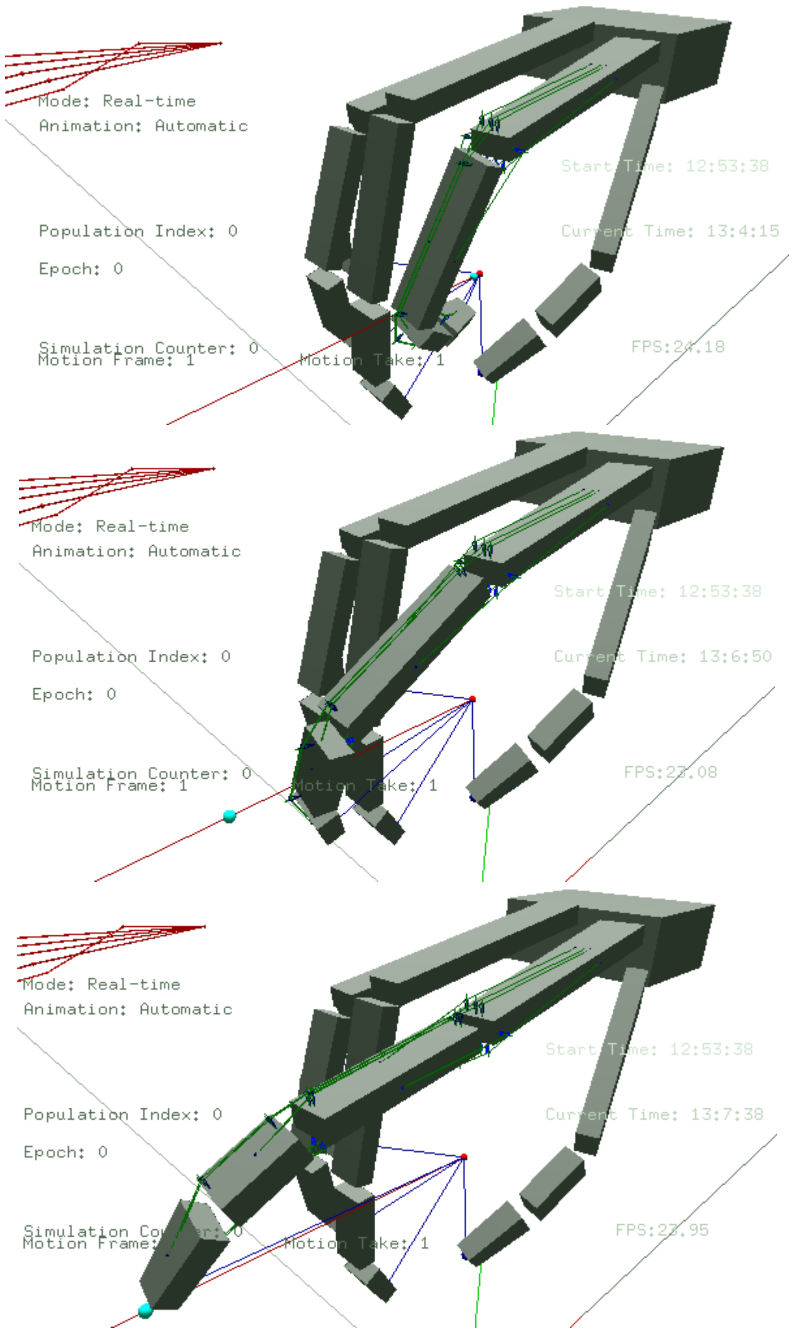
6.5 Current Problems and Shortcomings

Though successful in approximating an input motion, there are domains in the system that would benefit from fine tuning and improved models. These are:

- Current real-time physics engines are able to perform complex simulations. But even so, these software simulators are forced to compromise on simulation techniques, collision resolution, constraint dynamics and other physics implementation methodologies in order to achieve high frame rates and not impede overall game performance. This causes accuracy issues, especially in handling complex rigs of dynamic characters (the human hand in this research). Constraints break down and collision handling can fail (See Figures 6.21 and 6.22). There are instances when the collision system fails to handle the collision between the distal phalanx and the metacarpal phalanx during forward motion (see Figure 6.21). But collision handles on the reverse motion, thereby locking the finger indefinitely.

- With the help of licensed versions of commercial quality physics engines, it is possible to extend existing feature sets and mould the engine to function better with biomechanical character animations of high accuracy and real-time performance.

- Accuracy in motion approximation can be improved by resorting to anatomical muscle layouts. The movement of linkages is dependent on the point and direction of application of force. This describes the basic functionality of muscles and the layout of muscles on the underlying skeletal structure. The


```
Mode: Real-time
Animation: Automatic
Start Time: 12:53:38
Population Index: 0
Current Time: 13:4:15
Epoch: 0
Simulation Counter: 0
Motion Frame: 1
Motion Take: 1
FPS:24.18
```

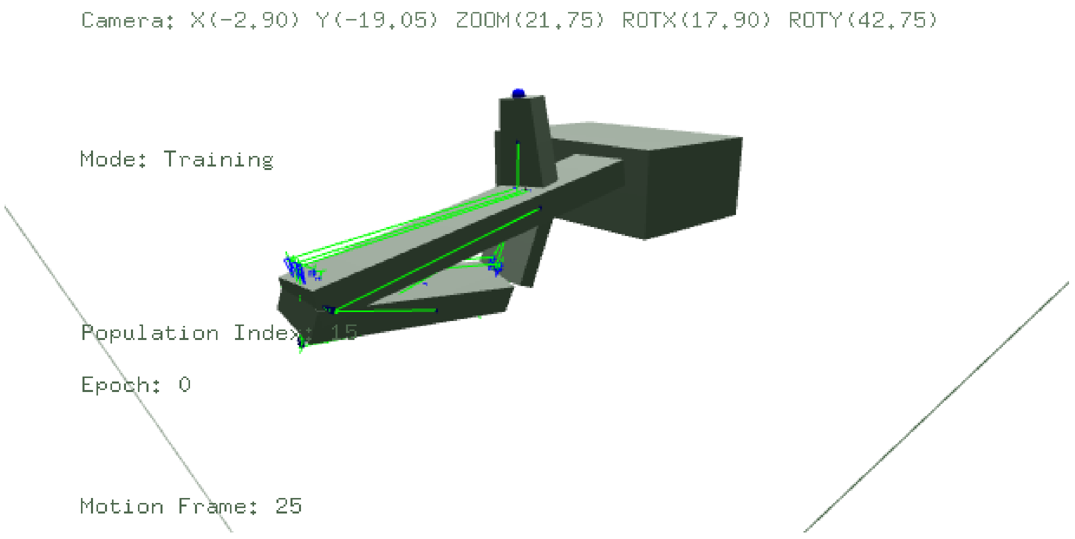
```
Mode: Real-time
Animation: Automatic
Start Time: 12:53:38
Population Index: 0
Current Time: 13:6:50
Epoch: 0
Simulation Counter: 0
Motion Frame: 1
Motion Take: 1
FPS:23.08
```

```
Mode: Real-time
Animation: Automatic
Start Time: 12:53:38
Population Index: 0
Current Time: 13:7:38
Epoch: 0
Simulation Counter: 0
Motion Frame: 1
Motion Take: 1
FPS:23.95
```

Figure 6.20: The movement of the forefinger is controlled by a procedurally animated locator (blue sphere).

current system employs a muscle layout which is referenced from robotics literature (Pollard and Gilbert, 2002). Anatomical landmarks on the skeletal structure can be used as 3D fitting points in superpositioning using least squares to easily transfer muscle layouts between characters having similar morphology.

- Better and accurate objective functions for genetic evolution can help in isolating networks that are more accurate in generating the input motion. The difficulty in achieving fast convergence necessitates the use of objective functions that generates a fitness metric by taking into account the entire range of the input and output motion rather than per-frame pose states. One method that accomplishes this is, generating a velocity map of the generated motion and then comparing it with the map of the input motion could give a better fitness factor. Comparison of velocity vectors can be performed using the vector directional cosine algorithm described in Chapter 5.



```
Camera: X(-2.90) Y(-19.05) ZOOM(21.75) ROTX(17.90) ROTY(42.75)

Mode: Training

Population Index: 15

Epoch: 0

Motion Frame: 25
```

Figure 6.21: A rare instance caught when the simulation broke down.

6.6 Summary

This chapter presented the simulation results and the possibility of using existing game software and hardware technology to create biomechanically accurate physical animation is explored. Prior to presenting the results of the current system, some early test results of a PD controller-based dynamic hand is described. The PD controlled system was a stepping stone to designing the more anatomically accurate current solution. A grasp contact determination concept is also explored

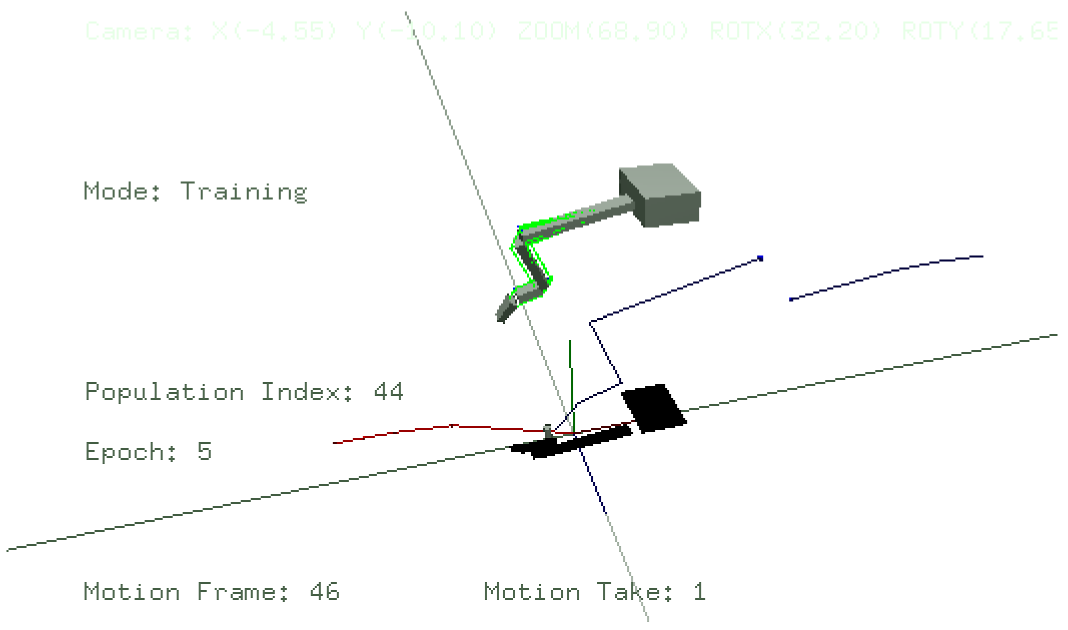


Figure 6.22: An example of constraint violation when the rigid body segment bends in an unnatural manner.

which is sufficiently generic to plug-in to the current system. The grasping contact point generation determines the finger contact points on an object based on line-of-action tests. The concept is implemented as a Maya plug-in to demonstrate viability and compatibility with a conventional 3D software package.

Following that the results of the current unified system, consisting of the neural controller and muscle model, are presented. The system is used to create gestural animations of a dynamic model of the human hand. The movement and constraints of the finger model is examined via manual input of muscle forces. Noise is added to the output activations and the resulting motion is monitored. It was also seen that the timing of the generated animation can be modified by increasing or decreasing the magnitude of force for the muscles. A procedural end-effector based interface for creating procedural animation using the current system is also presented. This interface enables programmatic control over the rigid body linkages by using end-effector positions rather than the 3D pose vectors of the entire linkage. The shortcomings and problems of the existing system regarding physics engine accuracy issues were also addressed.

7

CONCLUSIONS AND FUTURE WORK

7.1 Discussion

Physically-based character animation in games use joint torques generated by PD controllers to achieve target poses. PD controllers are simple to implement and are part of standard real-time physics engines. Active rag-dolls in games implement PD controllers to add realism to character motions during a fall or death. These controllers are also used to create intelligent transitions between and from motion capture and physically-based motion (see (Motion, 2011a)). But PD controllers have limitations due to notable differences in function from anatomical motion generation. The limitations prevent certain types of adaptive and procedural motion that are possible with a more accurate muscle actuated system. Deterministic methods for simulating muscle dynamics are computationally expensive to perform in-game. The alternative is to create a re-usable system that generalises to unknown target inputs and reduces computational overhead by an order of magnitude. The concept of neuromuscular control is demonstrated using a virtual physically-based model of the human hand with virtual muscle models tracking an input kinematic gestural motion. In view of the neuromuscular control system developed in this thesis, there are a few points that needs to be addressed.

Feedback systems in the human body play a vital role in control learning of voluntary movements. Feedback is essential for skill acquisition, especially response-initiated feedback that is common in physical movements (Shadmehr and Wise, 2005). In order to generate feedback information, sensors are necessary.

The currently implemented system is an open-loop controller that foregoes direct environmental feedback and hence there are no joint sensors in the physical model of the hand. The only information that acts as an indirect feedback resides in the objective functions for the GA, where a comparison is made between the input pose and the output pose.

The accuracy of the motion tracking (muscle-based motion with captured motion) is dependent on the muscle models. Mathematical models are approximations of the real anatomical muscles. The muscle locator-based muscle model is a very simplified model compared to proper biomechanical models like the Hill muscle model. For performance reasons, speed is preferred over accuracy. Thus, the generated motion has certain phase discrepancies with the input motion. The timing of the muscle activations is delayed and this reflects on the pose transitions during the motion. One possible reason is that there are two sub-systems that reside on top of the actual physical hand model, namely, the neural activation layer and the virtual muscle layer. But in spite of this discrepancy, the order of the activations is in accordance with the direction of movement. So, the correct muscles activate in the correct order so that the rigid body linkages of the fingers follow the motion captured fingers. Also, muscle laying based on anatomy would improve the motion. The system correctly mimics agonist/antagonist action found in the human body. Even though a detailed anatomical study of the hand is beyond the scope of this thesis, a sufficiently comprehensive description is given in Chapter 3 for reference to creating a better muscle layout system. A proper anatomical muscle model would automatically warrant an anatomically compliant muscle layout system.

Automatic trajectory correction is absent in the current system. This means that the system is prone to perturbations due to external forces like gravity or any other applied force. For corrective activations, the training process would require an additional alteration. Since dynamic turbulences during motion capture is not specifically captured unless required, the perturbations would have to be introduced to the physical model during the activation training. One possible method is to add a *perturbation vector* as an input to the neural network. The perturbation vector would be applied at every frame of the motion training allowing the evolutionary learning model to incorporate perturbation adjustments during weight modifications. For this process, the objective functions would remain unchanged.

The current system was modelled with game technology in mind. The system caters more to procedural in-game animations post-training. But the possibility exists to convert it to an off-line animation system commonly found in 3D animated films and visual effects. Before proceeding with the conversion, it is essential to dissect the tools and methodology involved in a conventional animation production. 3D animation is often created using an animation software which provides various tools for animating a character. A typical example is Autodesk's Maya or 3D Studio MAX. The mode of animations is kinematic, with dynamics acting as an enhancement that animates the participating objects connected to the character (like cloth or particles). There are also various muscle tools available within these animations software, although these muscle tools are used for simulating the effects of muscle movements as skin deformations instead of animating the skeletal structure of the character. Animators are familiar with standard IK and FK tools customised using their own control rigs. Therefore, there are two options available for converting the neuromuscular animation system into a workable tool in an off-line system. For both options, the muscle model is required for creating rigs. The options differs in the methodology of animation generation. The options are:

- Interactive simulation of the muscle-rigged character is created within the software (Maya or 3D Studio MAX) viewport allowing the artist to animate the character using muscles which is then "baked" for render.
- The animator creates the animation traditionally (using IK/FK) while a biomechanical proxy of the character with the muscles laid out, creates the final animation during render time.

The second option is efficient because it brings the best of both kinematic and dynamic techniques. Kinematic techniques are very efficient in giving real-time feedback to the animator allowing him/her to modify key-frames and adjust animation curves, whereas dynamics is very useful in bringing a level of secondary realism that is very tedious to create using kinematic methods. There are numerous other ways to integrate the current system into a production pipeline. Refer to Appendix 7.3 for a detailed description.

7.2 Further Research

Further research with the existing system is possible in the following areas:

- Creating an improved skeletal muscle model: The muscle model implemented currently was constructed with games and real-time physics engines in mind. So complex activation dynamics calculations were abandoned and a linear system favoured. Therefore, force-length and force-velocity dependency calculations are absent. Implementing force-length cause and effect can improve realism and also pave way for an alternate method of training the neural network, based on passive activations. Another area that would benefit is creating a physics engine compliant true tendinuous structure using b-spline models. This would better capture tendon-bone interaction mechanics without compromising the quality of animation in a real-time physics engine.

- Using different neural network architectures and type: There are more complex neuron models available that are time dependent and which model a biological neuron more accurately. Examples of these types of networks are *Spiking Neural Networks* and *Continuous Time Recurrent Neural Networks* using time dependent neuron models. These neural networks model the complex time dynamics involved in the CNS control system such as time delays in the production of muscle forces. Using accurate neuron models bridges the gap in understanding how motor control is achieved in biological organisms. Simulating these types of networks is extremely processor intensive and getting real-time performance and investigating suitably similar alternatives is a good research area.

- Full body animation: The feasibility of the existing system is demonstrated using the human hand which is an extremity of the human body. But the same technique is well suited for full body animation. The primary research focus would be a multi-network strategy for learning muscle activations and motion styles from motion capture and combining it with balance simulation.

- GPU Acceleration: General purpose computing using the GPU is gaining popularity where the problem to be solved is parallel in nature. The artificial neural network and genetic algorithms are highly suitable for parallel processing on the GPU. The performance improvement and architecture design of the machine learning components using parallel languages like CUDA and OpenCL is well worth researching. This would result in changes in the design of the GA and Artificial Neural Network code. Likely advantages would be speed in evolutionary processes and training.

7.3 Summary

In this thesis, a machine learning approach is undertaken, that successfully attempted and simulated muscle activations from an input kinematic motion to create physically-based muscle actuated motion of a set of articulated rigid bodies, namely the hand. The main contributions and innovation of this research are:

- An artificial neural network-based predictor model that generates muscle activations on a virtual muscle model which corresponds to the direction of movement in motion data.
- A unique force scaling method that is used by the muscle model emulate anatomical functionality with a game physics engine environment.
- A simplified linear piece-wise line segment-based muscle model that is optimized for a generic commercial quality real-time game physics engine. From a real-time perspective, this simplified model is very important. This muscle model was specifically created for use in real-time media and foregoes elastic behaviour. The elastic behaviour is simulated explicitly using joint springs.
- A training model for the artificial neural network that is based on an evolutionary model using suitable objective functions that selects the fittest controller network from a population of networks. The objective function is also capable of extremely fast retrieval of a pose from a motion sample.

By encapsulating complex activation dynamics in a neural network model, this system emphasises the ease of re-usability for converting kinematic animation to dynamic animation.

Previous research using machine learning techniques attempting to teach connectionist architectures, used recorded muscle activations as a base for gradient-based learning algorithms. The research in this thesis was successful in circumventing the requirement of such recorded data, using purely captured motion as a basis to reconstruct the muscle activations that originally produced the motion. The problem was divided into two parts: creating a lightweight physics engine-based muscle system and an artificial neural network- based controller that coordinated the muscle activations.

The ANN controller is a standard feed-forward neural network that accepts pose vectors as inputs and outputs muscle activations for the locator-based muscle system. Captured motion is used as a basis to teach the computer to generate coordinated muscle activations using artificial neural networks. 3D vectors are generated from the motion sample and fed into the ANN. The ANN is also able to provide solutions by generalising to unknown input vectors. The search space of activations is huge primarily due to the number of participating muscles in an action with varying forces and the size of the neural network that affects training time. The space of activations a muscle can produce is constrained to an interval (0,1). In order to traverse the vast search space, a genetic algorithm is implemented to search the population of randomly generated neural networks and select the best networks that produce the closest activations which would produce the input motion. A time series prediction method is used in the input architecture of the ANN in order to aid in the time series pattern identification of the input motion sample.

The muscle system implemented uses specific location specifiers (locators) on the rigid body to define the muscle path. These locators act as contact force generators when the muscle contracts. Muscle contraction is simulated using a force scaling method that scales the magnitude of forces using the activations generated by the controller.

The system was used to create gestural animations of the human hand. A fully physics-based rigid body dynamic model of the human hand was constructed and the muscle system laid out. The trained ANN successfully generated muscle activations and produced hand motion that matches the input sample. The ANN also generalised successfully, by producing matching motion to unknown input motion. As an extended development, a user controlled end-effector target-based ANN was also developed that successfully generated motion of the rigid body linkages by following the end-effector. This is useful in creating procedural grasping animations. The system was also used to create motion changes through noise additions in the network output.

Even though neural control is a non-trivial problem and produces the overhead of neural training, the existing system performs robustly in a real-time environment. Moreover, training is performed only once as an off-line process (though still producing animations in real-time). A character endowed with biomechanical virtual muscles can modify the target motion trajectory depending on changing game scenarios. Such a biomechanical system allows for individual muscles in a game character to be disconnected and the simulation would reflect the effect of the de-activation in the motion generated (subject to other object interactions), whereas in deterministic methods, muscle dynamics is not easily separable from the set of mathematical equations that describe the contraction behaviour and hence detrimental effects of muscular atrophy and neural damage in motion generation is not easily simulated. This type of effect is useful in games to simulate injured characters which is not possible with conventional PD controller driven physics characters. From a production point of view, since the trained neural network generalises to unknown motion samples, the system would be useful in converting a kinematic animation (key-framed or motion captured) into a physics-based dynamic animation.

Physics-based character animation is still in its infancy. Due to its mathematical nature, animator or artist friendly control parameters are always an issue. Issues in managing control parameters and maintaining fine control are the main reasons why dynamic character animation is only used in small portions in game animations. It is very processor intensive to drive all the animations physically. Animators are comfortable with user interfaces that have been a part of the production pipeline for a long time and drastic changes both in the interfaces and animation methods require some time for adaptation . The key is to design intuitive controllable parameters that allows the animator to treat physics-based character animation and control in a manner very similar to its kinematic counterparts.

Appendix A

-

Numerical Methods

Physical simulation requires numerical solvers to solve the differential equations of the laws of motion. Real-time physics engines (commercial and open source) always use one or a combination of these methods for stepping the simulation forward in time. Normally, physical simulations fall under a category called *initial value problems*. All three methods use *Taylors Theorem* to approximate the value of the function at time $t + h$.

$$x(t_0 + h) = x(t_0) + h\,\dot{x}(t_0) + \frac{h^2}{2!}\,\ddot{x}(t_0) + \frac{h^3}{3!}\,\dddot{x}(t_0) + ... + \frac{h^n}{n!}\frac{\partial^n x}{\partial t^n} + ...$$

Eberly and Shoemake (2004) defines the theorem as: If $x(t)$ and its derivatives $x^{(k)}(t)$ for $1 \leq k \leq n$ are continuous in the closed interval $[t_0, t_1]$ and $x^{(n)}(t)$ is differentiable on the open interval (t_0, t_1) then there exists $t^{-}E[t_0, t_1]$ such that

$$x(t) = \sum_{k=0}^{n} \frac{x^{(k)}(t_0)}{k!}(t_1 - t_0)^k + \frac{x^{(n+1)}(t^{-})}{(n+1)!}(t_1 - t_0)^{n+1}$$

The polynomial,

$$P_n(t) = \sum_{k=0}^{n} \frac{x^{(k)}(t_0)}{k!}(t - t_0)^k$$

is called the *Taylor polynomial of degree* n and can be used to compute an approximate value for $x(t)$ and the remainder is

$$R_n(t) = \frac{x^{(n+1)}(t^{-})}{(n+1)!}(t - t_0)^{n+1}$$

The summation form, the polynomial and the remainder are given as in Eberly and Shoemake (2004), while the fully expanded Taylor series is as given in Witkin (1997).

Euler's Method

The Eulers method of numerical solving of differential equations is the simplest and the fastest. The drawback is its lack of stability and accuracy during simulation. This is largely because of large truncation errors. Using $n = 2$ in the equation for the Taylors theorem, it is possible to find the value of the simulation function at a future time and gives,

$$x(t_{i+1}) = x(t_i) + \dot{x}(t_i)h + \ddot{x}(t^-)\frac{h^2}{2}$$

Removing the remainder part of the equation (the second derivative of x), we get the equation for Eulers method

$$y_{i+1} = y_i + \dot{x}(t_i)h, \{i \geq 0, y_0 = x_0\}$$

The truncation of terms is evident from the fully expanded Taylor series given in the beginning. This truncation of terms is crucial in understanding the error, which in turn is needed to improvise the Euler method (for the RK4 method). The error term starts with

$$\frac{h^2}{2!}\ddot{x}(t_0)$$

From this the error is $O(h^2)$. Bigger values of h would increase the error. If the step is halved, thereby reducing the error to one fourth, the number of steps required to reach h would double. So the error is directly dependent on the time step h.

Runga-Kutta Method

The Runge-Kutta methods of integration is a well used category of numerical solving methods used in physical simulations. Eulers method also falls into this category. This section looks into a particular form of the Runge-Kutta method known as the Fourth Order Runge-Kutta method or RK4 for short. The RK4 is $O(h^4)$ order of complexity. The RK4 requires computing four intermediate steps between t and $t + h$.

$$\begin{aligned}
&u_1 = h_i f(t_i, y_i) \\
&u_2 = h_i f(t_i + \tfrac{h_i}{2}, y_i + \tfrac{1}{2}u_1 \\
&u_3 = h_i f(t_i + \tfrac{h_i}{2}, y_i + \tfrac{1}{2}u_2) \\
&u_4 = h_i f(t_i + h_i, y_i + u_3) \\
&y_{i+1} = y_i + \tfrac{1}{6}[u_1 + u_2 + u_3 + u_4]
\end{aligned}$$

The cost of the improved accuracy of the method is the requirement of four intermediate calculations per time step. The RK4 is a combination of two methods (improved accuracy) not shown here, the modified Euler method and the mid-point method.

Verlet Integration

The Verlet class of integration methods is a crossover from the field of molecular dynamics into the world of games. Thomas Jakobsen developed and implemented the algorithms using this method for the physical simulation of ragdolls in IO Interactives hit game *Hitman: Codename 47* (Jakobsen, 2001). Verlet integration is otherwise known as a *velocity-less* because it integrates position from acceleration without using velocity. Hence, it is extremely useful in simulating particles (collection of very small, volumeless point masses) and also when simulating constrains between the particles (Van Verth and Bishop, 2008).

Adding the Taylor expansion for the current time step with the previous time step gives,

$$\begin{aligned} &y(t+h) + y(t-h) = y(t) + h\,\dot{y}(t) + \tfrac{h^2}{2}\,\ddot{y}(t) + ... + y(t) - h\,\dot{y}(t) + \tfrac{h^2}{2}\,\ddot{y}(t) - ... \\ &y(t+h) = 2y(t) - y(t-h) + h^2\,\ddot{y}(t) + O(h^4) \end{aligned}$$

Eliminating the higher order terms,

$$y(t+h) = 2y(t) - y(t-h) + h^2\,\ddot{y}(t)$$

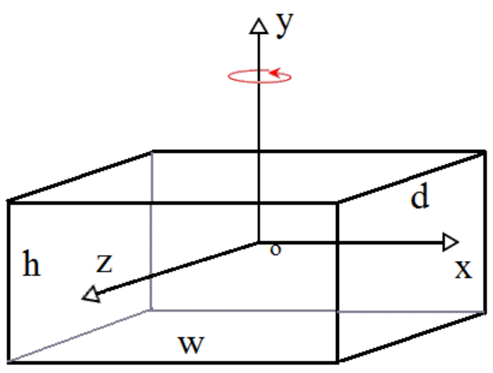
This is an $O(h^2)$ solution that integrates position from acceleration without the need for velocity and hence the velocity-less nature of the Verlet method. Being time invariant, particle simulation using the Verlet method can be run forward and backwards in time (Van Verth and Bishop, 2008). There are Verlet methods that calculate velocities. *Leapfrog Verlet* and *velocity Verlet* are two Verlet methods that track velocity (Van Verth and Bishop, 2008). Tracking velocity is essential for the simulation of friction.

Appendix B

Common Geometry Primitive Inertia Tensors

Physical simulation of rigid bodies requires the moment of inertia or inertia tensor or masses in order to compute angular velocity and angular acceleration. The inertia tensors of common object primitives are given below. Most of the commercial (and non-commercial) physics engines support these primitives and therefore use the following calculations.

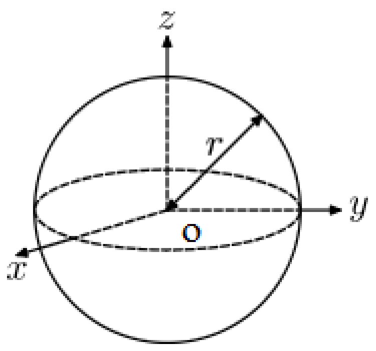
Cuboid



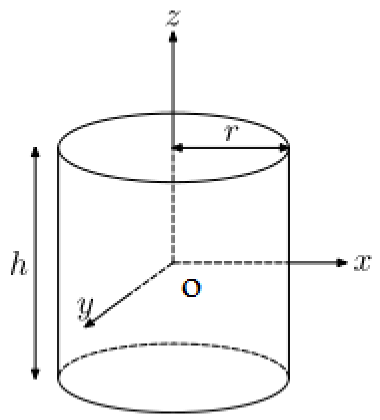
$$I = \begin{bmatrix} \frac{1}{12}m(h^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+d^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+h^2) \end{bmatrix}$$

Sphere

$$I = \begin{bmatrix} \frac{2}{5}mr^2 & 0 & 0 \\ 0 & \frac{2}{5}mr^2 & 0 \\ 0 & 0 & \frac{2}{5}mr^2 \end{bmatrix}$$

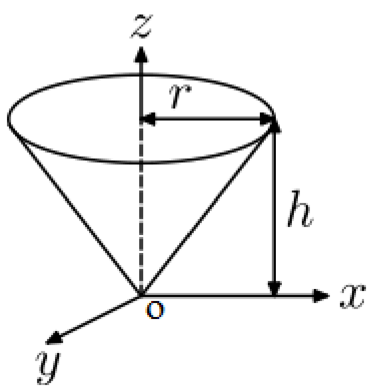


Cylinder



$$I = \begin{bmatrix} \frac{1}{12}m(3r^2+h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2+h^2) & 0 \\ 0 & 0 & \frac{1}{12}mr^2 \end{bmatrix}$$

Cone



$$I = \begin{bmatrix} \frac{3}{5}mh^2 + \frac{3}{20}mr^2 & 0 & 0 \\ 0 & \frac{3}{5}mh^2 + \frac{3}{20}mr^2 & 0 \\ 0 & 0 & \frac{3}{10}mr^2 \end{bmatrix}$$

Appendix C

Custom Text file Formats

Below are given the various text-based file formats created for exporting data between the modelling framework (Autodesk Maya) and the simulation framework (the PhysX application, training data extraction from the FBX motion data and the neural net simulation module). All the formats are simple tag-based formats similar to XML. In the future, possibility exists to convert the formats into pure XML thereby standardizing the parsing process.

Joint file format

The joint export file format, as the name implies, is used to export the joint information (position, orientation, type and connecting bodies). The tags are self-explanatory. The information of a joint is placed between the $< jnt >< /jnt >$ tags, within which the above-mentioned information is placed, either as name-value pairs or within nested tags, like the transformation matrix. The joint file format has a file extension (.jnt).

```
<jnt>
name=DIPJnt
type=revjnt
actor1=DistalPhalange
actor2=InterPhalange
<matrix>
0.7921975181 0.5870075717 0.1668688198 0
−0.5866499215 0.8078492867 −0.05675737545 0
−0.1681218662 −0.05293052805 0.9843441458 0
−11.40712835 8.165527518 1.5984333 1
</matrix>
```

```
</jnt>
<jnt>
name=MCPJnt
type=sphjnt
actor1=MetacarpalPhalange
actor2=ProximalPhalange
<matrix>
0.9714396469 0.2372867726 0 0
-0.2372867726 0.9714396469 0 0
0 0 1 0
-3.89624726 10.60723316 1.760809397 1
</matrix>
</jnt>
<jnt>
name=PIPJnt
type=revjnt
actor1=InterPhalange
actor2=ProximalPhalange
<matrix>
0.9366352191 0.3478620296 0.04130949894 0
-0.347524196 0.9375457367 -0.0153272588 0
-0.04406131597 1.734723476e-017 0.9990288286 0
-9.111888256 9.074044594 1.720138193 1
</matrix>
</jnt>
<jnt>
name=WristJnt
type=fixjnt
actor1=MetacarpalPhalange
actor2=WristBox
<matrix>
0.9969053365 0 -0.07861138582 0
0 1 0 0
0.07861138582 0 0.9969053365 0
5.326832888 12.08383844 0.8308477492 1
</matrix>
</jnt>
```

Object Export file Format

The various surface meshes (bones) are exported with vertex data to the PhysX application. This object export file format serves this purpose. The vertex data is given within the $< object >< /object >$ tag. The vertex data is used by the PhysX application to generate a convex hull rigid body. The object file format has a file extension (.bdy).

```
<OBJECT>
NAME=WristBox
NUMVERTS=8
5.751612273 10.05044714 1.855332545
10.170724 10.05044714 1.855332545
5.751612273 12.44223467 1.855332545
10.170724 12.44223467 1.855332545
5.751612273 12.44223467 −4.551213571
10.170724 12.44223467 −4.551213571
5.751612273 10.05044714 −4.551213571
10.170724 10.05044714 −4.551213571
</OBJECT>
<OBJECT>
NAME=MetacarpalPhalange
NUMVERTS=8
−3.528156996 10.31964684 2.297899961
5.045786858 11.61166954 1.700646043
−3.62928009 10.99070358 2.297899961
4.944664001 12.28272724 1.700646043
−3.736160994 10.97459793 1.097776055
4.805333138 12.26173115 0.1361449957
−3.635037899 10.30354118 1.097776055
4.906455994 11.5906744 0.1361449957
</OBJECT>
<OBJECT>
NAME=ProximalPhalange
NUMVERTS=8
−8.786824226 8.788265231 2.298346444
−4.039862156 10.23062802 2.298346444
−8.984120369 9.437586787 2.298346444
−4.237157822 10.87994862 2.298346444
−8.984120369 9.437586787 1.093364521
```

```
−4.237157822 10.87994862 1.093364521
−8.786824226 8.788265231 1.093364521
−4.039862156 10.23062802 1.093364521
</OBJECT>
<OBJECT>
NAME=InterPhalange
NUMVERTS=8
−11.13336919 7.955673218 2.161617041
−9.275342683 8.652562141 2.309717894
−11.3716924 8.591082573 2.161617041
−9.513665895 9.28797245 2.309717894
−11.28772328 8.622576714 0.9599769711
−9.429696779 9.319465637 1.108078003
−11.04940007 7.987166882 0.9599769711
−9.191373567 8.684056282 1.108078003
</OBJECT>
<OBJECT>
NAME=DistalPhalange
NUMVERTS=8
−12.75316715 6.348068237 1.509700141
−11.38812256 7.696584225 2.071010194
−13.18757725 6.771625042 1.509700141
−11.87401962 8.170343399 2.071010194
−13.07863998 6.883353233 0.6978121023
−11.7152462 8.333185196 0.8876871686
−12.64423084 6.459795952 0.6978121023
−11.22934914 7.859426975 0.8876871686
</OBJECT>
```

Muscle Locator Export file Format

The muscle locator placed on the bone surface in Maya is exported using this format. At present, not all the attributes of the muscle locator are exported. The information exported include, the name of the bone surface on which the locator is placed, the muscle group it belongs to, the transformation matrix, the type of muscle locator (origin, insertion, intermediate) and the locators order in the muscle group. The muscle locator file has a file extension (.mus).

```
<muscleset>
<musclegroup id=muscleGrp_distalEXT>
<muscle>
<origin>
order=0
segment=MetacarpalPhalange
<matrix>
0.9888357722 0.1490094484 0 0
-0.1490094484 0.9888357722 0 0
0 0 1 0
1.996658875 11.83848695 1.317274304 1
</matrix>
</origin>
<intermediate>
order=1
segment=MetacarpalPhalange
<matrix>
0.9888357634 0.1490095067 -2.067951531e-025 0
-0.1490095067 0.9888357634 2.27704485e-008 0
3.393013297e-009 -2.251623382e-008 1 0
-3.576016678 10.99873016 1.688745584 1
</matrix>
</intermediate>
<intermediate>
order=2
segment=ProximalPhalange
<matrix>
0.9568066023 0.2907251722 0 0
-0.2907251722 0.9568066023 0 0
0 0 1 0
-4.44748138 10.81604193 1.740745721 1
</matrix>
</intermediate>
<intermediate>
order=3
segment=ProximalPhalange
<matrix>
0.9568066106 0.290725145 0 0
```

```
−0.290725145 0.9568066106 0 0
0 0 1 0
−8.8494893 9.47849436 1.665783631 1
</matrix>
</intermediate>
<intermediate>
order=4
segment=InterPhalange
<matrix>
0.9363078774 0.3511802368 0 0
−0.3511802368 0.9363078774 0 0
0 0 1 0
−9.595594244 9.257243031 1.618755017 1
</matrix>
</intermediate>
<intermediate>
order=5
segment=InterPhalange
<matrix>
0.9363076618 0.3511808116 0 0
−0.3511808116 0.9363076618 9.943059203e−008 0
3.491811601e−008 −9.309762513e−008 1 0
−11.22654475 8.64552301 1.513454727 1
</matrix>
</intermediate>
<insertion>
order=6
segment=DistalPhalange
<matrix>
0.6852964908 0.7282626923 −0.001473352693 0
−0.7282626923 0.6852895929 −0.00340951965 0
−0.001473352693 0.00340951965 0.9999931022 0
−12.4335533 7.571922934 1.307185648 1
</matrix>
</insertion>
</muscle>
</musclegroup>
<musclegroup id=muscleGrp_interEXT>
<muscle>
```

```
<origin>
order=0
segment=MetacarpalPhalange
<matrix>
0.9888357722 0.1490094484 0 0
-0.1490094484 0.9888357722 0 0
0 0 1 0
2.009875248 11.84047856 1.471239349 1
</matrix>
</origin>
<intermediate>
order=1
segment=MetacarpalPhalange
<matrix>
0.9888357634 0.1490095067 -2.067951531e-025 0
-0.1490095067 0.9888357634 2.27704485e-008 0
3.393013297e-009 -2.251623382e-008 1 0
-3.581821129 10.99785541 1.892260284 1
</matrix>
</intermediate>
<intermediate>
order=2
segment=ProximalPhalange
<matrix>
0.9568066023 0.2907251722 0 0
-0.2907251722 0.9568066023 0 0
0 0 1 0
-4.401774251 10.82993001 1.866454082 1
</matrix>
</intermediate>
<intermediate>
order=3
segment=ProximalPhalange
<matrix>
0.9568066106 0.290725145 0 0
-0.290725145 0.9568066106 0 0
0 0 1 0
-8.816305211 9.488577326 1.774725637 1
</matrix>
```

```
</intermediate>
<insertion>
order=4
segment=InterPhalange
<matrix>
0.9363078774 0.3511802368 0 0
-0.3511802368 0.9363078774 0 0
0 0 1 0
-10.2266597 9.020549828 1.740064683 1
</matrix>
</insertion>
</muscle>
</musclegroup>
<musclegroup id=muscleGrp_proxiEXT>
<muscle>
<origin>
order=0
segment=MetacarpalPhalange
<matrix>
0.9888357722 0.1490094484 0 0
-0.1490094484 0.9888357722 0 0
0 0 1 0
1.941687484 11.8302032 1.010575503 1
</matrix>
</origin>
<intermediate>
order=1
segment=MetacarpalPhalange
<matrix>
0.9888357634 0.1490095067 -2.067951531e-025 0
-0.1490095067 0.9888357634 2.27704485e-008 0
3.393013297e-009 -2.251623382e-008 1 0
-3.569413846 10.99972527 1.371007116 1
</matrix>
</intermediate>
<insertion>
order=2
segment=ProximalPhalange
<matrix>
```

```
0.9568066023 0.2907251722 0 0
−0.2907251722 0.9568066023 0 0
0 0 1 0
−6.72974206 10.12257834 1.507633453 1
</matrix>
</insertion>
</muscle>
</musclegroup>
</muscleset>
```

Training Motion data Export file Format

The training data is extracted from the Autodesk FBX motion file. The vectors specifying a linkage are calculated from the skeleton data. The data extracted consists of global space coordinates of the computed vector as well as the origin point of the vector and the direction cosines of the vector for orientation. The above data is extracted on a per frame basis and stored between the $< frame >< /frame >$ tags. The frames are children of the $< motion >< /motion >$ tag. The training data export file format has a file extension (.trn).

```
<motion>
<frame=1>
<vectorchain>
<id=0>
<vector>
<coords>
10.2684 1.21229 1.21229
</coords>
<dcosines>
0.29335 1.45754 1.30139
</dcosines>
<pos>
0 0 0
</pos>
</vector>
<vector>
<coords>
4.80482 0.168459 0.168459
```

```
</coords>
<dcosines>
0.079388 1.53584 1.49955
</dcosines>
<pos>
10.2684 1.21229 2.85501
</pos>
</vector>
<vector>
<coords>
2.60172 −0.0937687 −0.0937687
</coords>
<dcosines>
0.0796643 1.60673 1.49973
</dcosines>
<pos>
15.0732 1.38074 3.19814
</pos>
</vector>
<vector>
<coords>
2.40235 −0.0868062 −0.0868062
</coords>
<dcosines>
0.0797058 1.60682 1.49973
</dcosines>
<pos>
17.675 1.28698 3.38347
</pos>
</vector>
<endnode>
20.0773 1.20017 3.5546
</endnode>
</vectorchain>
</frame>
</motion>
```

Neural Network saving file Format

This file was created for writing out the neural network (neurons, layers, neuron weights, threshold) settings. In the present state, it does not write out the architecture because the application uses a fully connected (between neurons of two consecutive layers) network. The ANN file format has a file extension (.net).

```
<network>
<layers=3>
<layer id=0>
<layer=input>
<neuron id=0>
<inputs=1>
<threshold=0.0635681>
<weight=-0.498749>
</neuron>
<neuron id=1>
<inputs=1>
<threshold=0.308716>
<weight=-0.306702>
</neuron>
<neuron id=2>
<inputs=1>
<threshold=-0.0201416>
<weight=0.0849915>
</neuron>
</layer>
<layer id=1>
<layer=hidden>
<neuron id=0>
<inputs=3>
<threshold=0.246582>
<weight=-0.149719>
<weight=0.395935>
<weight=0.322815>
</neuron>
<neuron id=1>
<inputs=3>
<threshold=0.0135193>
<weight=-0.325897>
```

```
<weight=0.358917>
<weight=0.21048>
</neuron>
<neuron id=2>
<inputs=3>
<threshold=−0.135559>
<weight=−0.196014>
<weight=−0.485016>
<weight=−0.4086>
</neuron>
<neuron id=3>
<inputs=3>
<threshold=−0.0543213>
<weight=−0.352692>
<weight=−0.334106>
<weight=0.488495>
</neuron>
<neuron id=4>
<inputs=3>
<threshold=−0.122131>
<weight=−0.38092>
<weight=−0.495331>
<weight=−0.491089>
</neuron>
</layer>
<layer id=2>
<layer=output>
<neuron id=0>
<inputs=5>
<threshold=0.163025>
<weight=0.0316467>
<weight=0.071167>
<weight=0.101746>
<weight=0.107147>
<weight=−0.333771>
</neuron>
<neuron id=1>
<inputs=5>
<threshold=0.302582>
```

```
<weight=-0.0492249>
<weight=-0.147888>
<weight=-0.442963>
<weight=0.107666>
<weight=0.283295>
</neuron>
<neuron id=2>
<inputs=5>
<threshold=0.42569>
<weight=0.0198669>
<weight=-0.198059>
<weight=0.375946>
<weight=0.226654>
<weight=0.455872>
</neuron>
</layer>
</network>
```

End-effector Export file Format

The end effector locator is a matrix that denotes the position of the end of the joint chain. This is important to rebuild the link in the simulation environment. This file format has a file extension (.end).

```
<endeffectorpoint>
linkID=WristJnt
name=endEffectorPointLocator
endlink=DistalPhalange
<matrix>
1 0 0 0
0 1 0 0
0 0 1 0
-12.88794707 6.529908363 1.065800936 1
</matrix>
</endeffectorpoint>
```

Genetic Population Export file Format

It is possible to output the entire genetic population into a file. The following file format was created with that purpose in mind. This is useful to stop and start the evolution at any point. The file extension is .gao.

```
<HEADER>
EPOCH=10
POPULATION SIZE=50
MUTATION PROB=0.0175
CROSSOVER PROB=0
GENE SIZE=1050
VVECTOR SIZE=52500
</HEADER>
<CHROMOSOME ID=0>
FITNESS=−0.0112199
<GENES>
1.51318
1.36645
−1.00025
0.869028
0.02544
0.40674
−0.751089
0.871438
−0.716687
0.403261
.
.
.
.
</GENES>
<VARIANCE>
0.31787
0.533222
−0.317877
0.376147
−0.121852
0.450917
−0.444236
```

```
0.546081
−0.338346
0.118954
−0.283533
0.689133
−0.103283
0.143662
−0.116119
0.684874
−0.131253
−0.701738
0.186323
0.011913
.
.
.
.
</VARIANCE>
</CHROMOSOME>
<CHROMOSOME ID=0>
FITNESS=−0.02119
<GENES>
.
.
.
</GENES>
<VARIANCE>
.
.
.</VARIANCE>
</CHROMOSOME>
```

Chromosome Export file Format

A single network weight encoded chromosome can also be exported. This is important refine the solution using other evolutionary optimization methods like Hill Climbing. The file extension is .gas.

```
<HEADER>
POPULATION SIZE=50
GENE SIZE=1050
VVECTOR SIZE=52500
</HEADER>
<CHROMOSOME ID=9>
FITNESS=−0.00380829
<GENES>
2.26061
4.51739
−1.5248
3.77671
−1.03259
3.26371
−2.60857
.
.
.
.
</GENES>
<VARIANCE>
0.497428
1.00317
−0.490078
0.893702
−0.24529
0.776144
−0.709042
0.82311
−0.665519
0.231937
−0.494683
1.62176
−0.215485
0.260736
−0.164094
1.38509
−0.339546
```

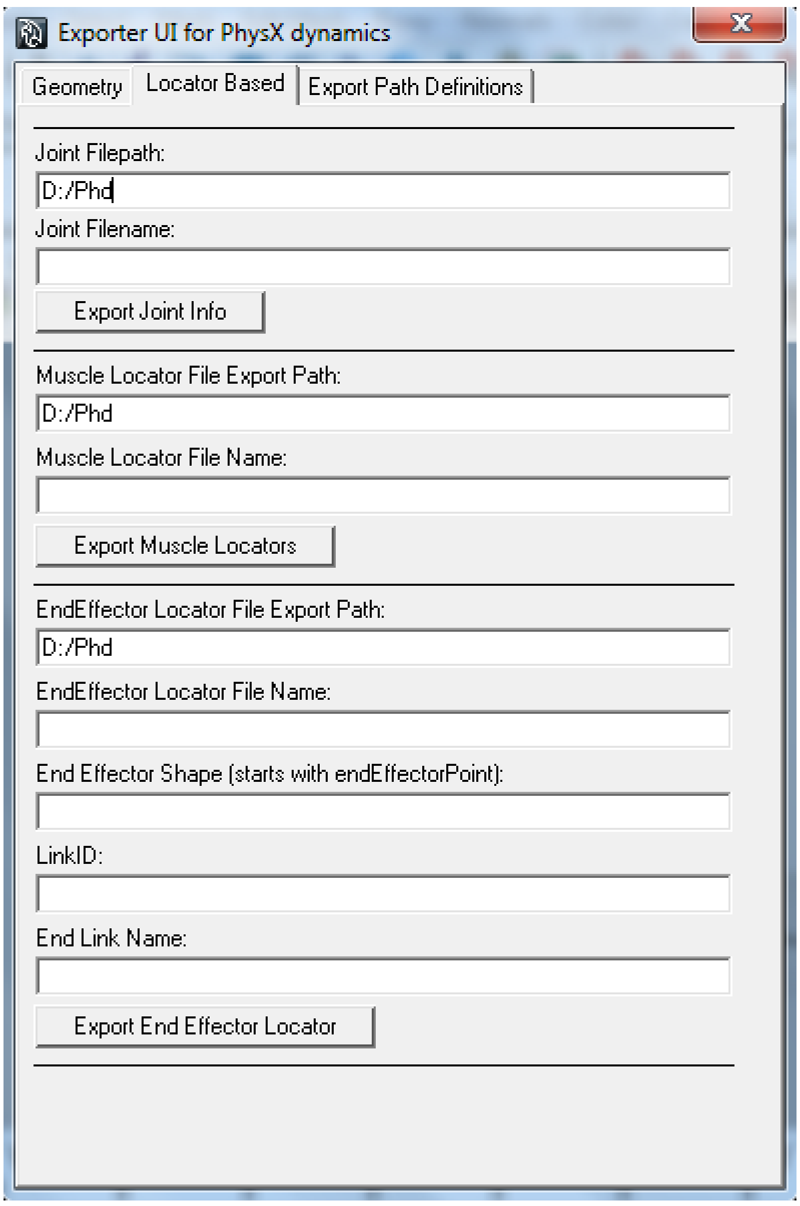
```
−1.11774
.
.
.
.
</VARIANCE>
</CHROMOSOME>
```

Appendix D

Maya Embedded Language (MEL) User Interfaces

User interfaces are essential for the artists to interact with the underlying algorithms in any software package. The user interfaces designed in the course of this research are mainly for Autodesk Maya for exporting muscle rigs that are set up by the artist. The main window consists mainly of text fields where the user enters the file name and the path to export to, with the exception of the joint export window. It mainly acts as a data entry interface for the user to specify exporting parameters. The muscle rigs and various other data is exported from Maya into the simulation using the various file formats detailed above. The export files are configured depending on the user interface settings specified by the artist.

The joint export window provides a convenient interface that makes it easier for the user to select and associate the bone meshes with the correct locator that indicates the joint between them. The UI also provides a selection mechanism for the types of joint the locator represents. The joint type is based on the standard joints available in the Nvidia PhysX simulation engine. The joint types are selected based on the similarity in function between it and an anatomical joint.



Joint Filepath: D:/Phd

Joint Filename:

Muscle Locator File Export Path: D:/Phd

Muscle Locator File Name:

EndEffector Locator File Export Path: D:/Phd

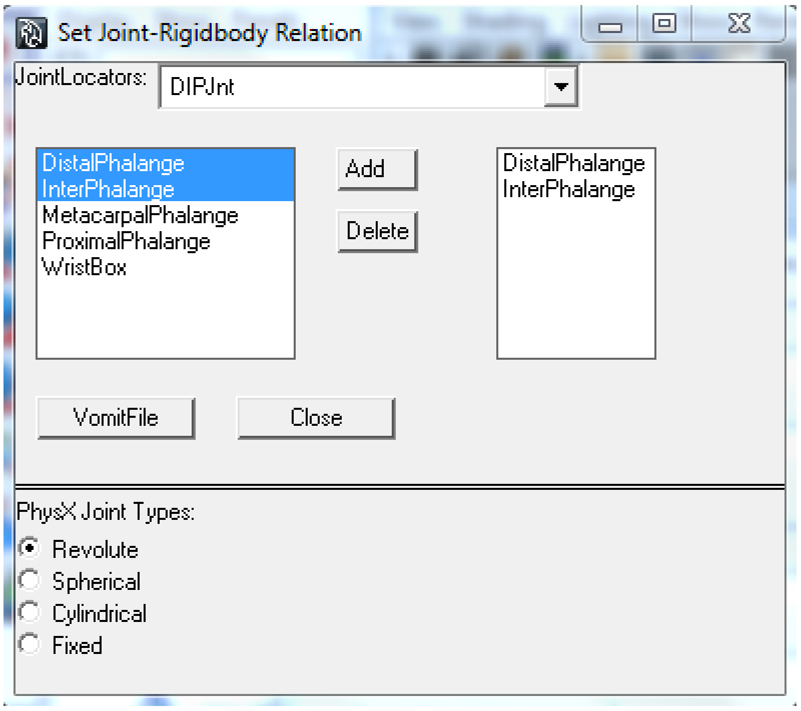
EndEffector Locator File Name:

End Effector Shape (starts with endEffectorPoint):

LinkID:

End Link Name:

Figure 1: MEL interface for exporting data.


Set Joint-Rigidbody Relation

JointLocators: DIPJnt

DistalPhalange
InterPhalange
MetacarpalPhalange
ProximalPhalange
WristBox

DistalPhalange
InterPhalange

PhysX Joint Types:

- [✓ Revolute]
- Spherical
- Cylindrical
- Fixed

Figure 2: MEL interface for selecting and exporting joint data based on the 3D model. The artist can also choose the type of joint used in the PhysX simulation environment.

Appendix E

Introduction

The system detailed as part of this research is prototypical. It is a proof-of-concept resulting from an interdisciplinary study. Physics-based character animation and control is still an active area of research and there are not many applications that have managed to integrate it into a usable animation pipeline. The application that deserves mention due to proper integration of machine learning and physics-based character animation to synthesize motion behaviours is Natural Motions *Endorphin* (Motion, 2011a). *Endorphin* uses pre-created motion behaviours to drive a physics-based full-body character skeleton. Adjusting parameters and changing the simulation properties can develop various animation styles and thus there is no "canned" animation. Other character animation tools include *Havok Behaviour* from Havok, *Kynapse* from Autodesk and Natural Motions *Morpheme* (Havok, 2011)(Autodesk, 2011a)(Motion, 2011b). These toolsets are state-machine based and uses parametric motion graphs or blend trees to blend between motion clips from a database depending on the blend graphs created by the animator. *Kynapse* is an AI middle-ware for games that use path finding algorithms and mesh navigation for navigating the non-player characters (NPCs) in large game environments. It does not generate character animation. Havok also has a similar tool called *Havok AI*. With the exception of *Endorphin*, all of the software mentioned above uses traditional kinematic methods for animation generation.

In the system that forms the body of this research, a lot of feature sets can be added to make it full-fledged and also modifications made in order to make the system fit into a production animation pipeline. This appendix explores the various parts where modifications are possible and also investigates adding certain features, in view of research done elsewhere, so that the system becomes production friendly.

Possible Feature Sets

From a conventional production pipeline point of view, there are various software and tools, which are used to create the effects and animations seen in visual effects, games and animated movies. There are various departments that come into combining all the output produced to generate a cohesive whole. But when an isolated part of the pipeline is examined in-depth to integrate a completely new system, it makes sense to plug in the different parts of the system under the controlling application in the system. See Figure 3 and Figure 4 that shows a fairly accurate production pipeline schematic followed in the games industry and the film visual effects industry.

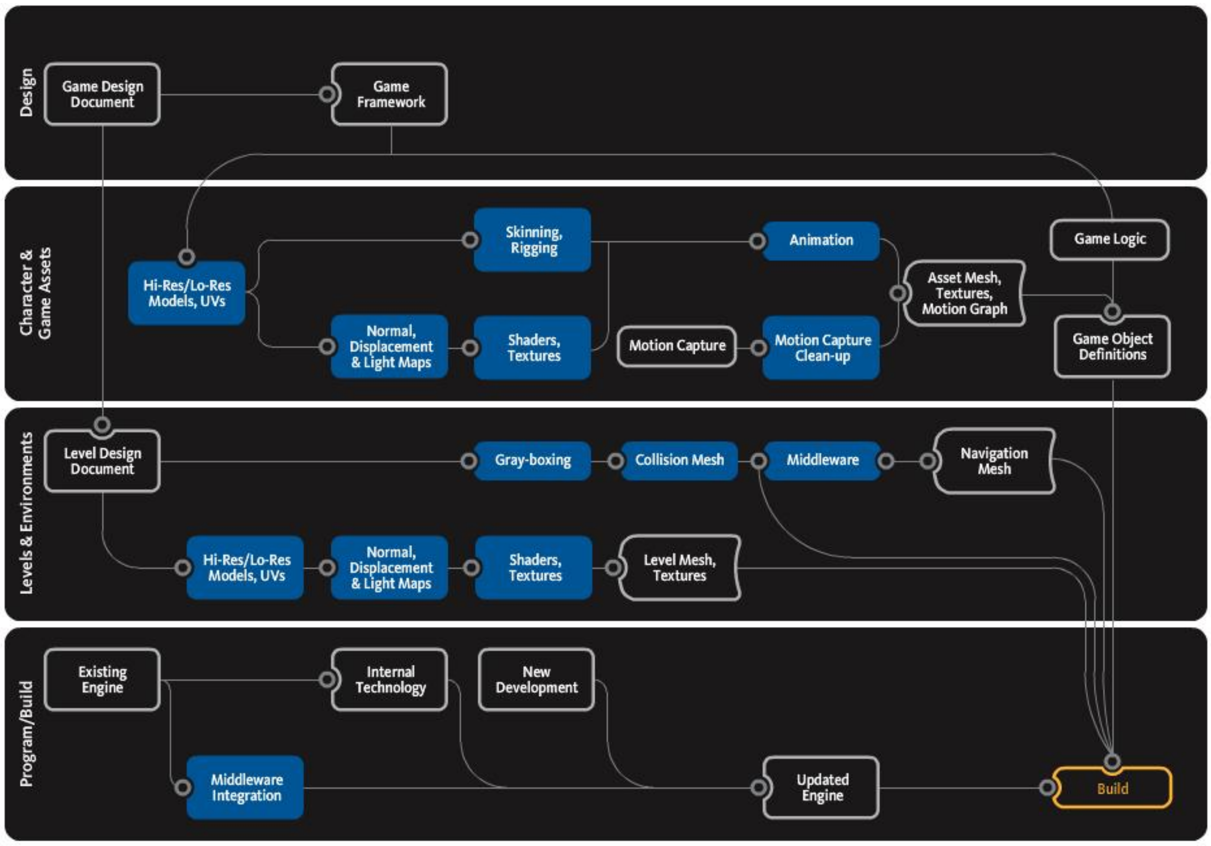


Figure 3: Games Pipeline. Image courtesy (Autodesk, 2011b)

Integration into the core Animation Application

Autodesk Maya is a core component of the production pipeline in the film visual effects industry. The easy-to-use intuitive interfaces and expandable plug-in architecture allows developers to add functionality and new features when the situation asks for. Similarly, Autodesk 3D Studio Max is also very popular in the gaming industry fast gaining popularity in the visual effects industry also. In a

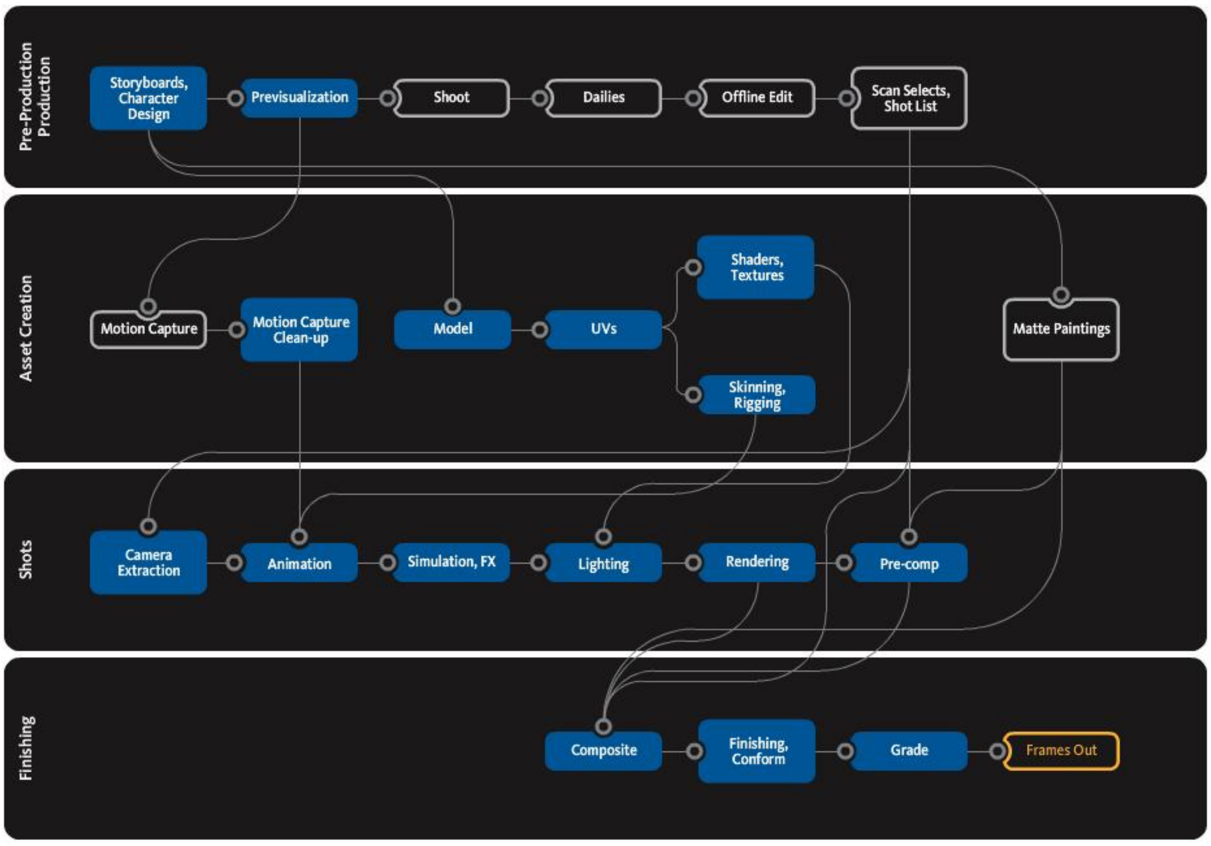


Figure 4: Film visual effects Pipeline. Image courtesy (Autodesk, 2011b)

conventional visual effects pipeline (see Figure 4), a 3D animation and modelling package like Maya or 3D Studio Max plays an active role in the first three of the four layers, namely, Pre-production and production, Asset creation and finally Shots. Therefore it is beneficial to the production schedule, logistically (time) and technically (troubleshooting and standardization of interchangeable formats) to integrate the simulation system into Autodesk Maya (similarly for any other core package). The two core aspects of the system that can be integrated in Maya are, the simulation module and the neural network machine-learning module.

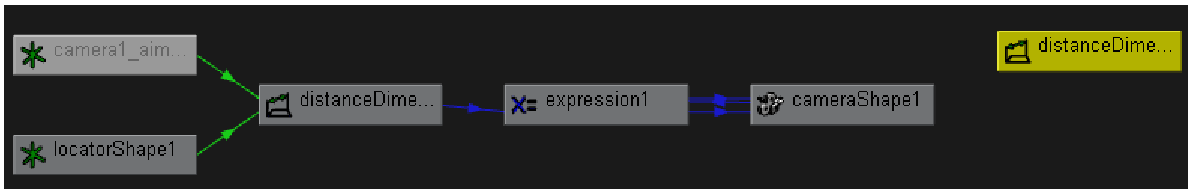


Figure 5: Maya hypergraph.

During the initial stages of exploratory research, the lack of a proper built-in real-time physics simulation toolset in Maya (version 6.5) prompted the decision to use a third party simulation technology (PhysX). The rigid body solver existing in Maya had stability issues and lacked a good joint constraint solver, which was essential to creating rigid body linkages. At present, many of the commercial physics engine companies provide simulation plug-ins for standard animation packages, allowing developers to use them as is or use the classes provided to develop custom simulations that are many orders more powerful than Mayas built-in simulation system. But Maya, since version 8.5, has Autodesks Nucleus simulation framework integrated into it, which brings unified simulation architecture to simulate a wide variety of natural phenomena. The Nucleus system is a common dynamics solver that simulates various phenomena using methods that are independent of what the phenomena is (Autodesk, 2009). So there is no specialised code for solving cloth, fluids or rigid body simulations. The common dynamics solver solves everything. This unified framework is achieved through the use of distributed building block design of the core solver (Autodesk, 2009). The creation of the Nucleus system makes it ideal for performing the simulations also within Maya, rather than outside. Proper joint constraints (like that available in physics engines like PhysX and Havok) are still not available, but can be developed using Mayas plug-in architecture using the Nucleus framework, making constraint dynamics efficient and stable. The Nucleus solver is one of the options available. The other is to use the PhysX plug-in for Autodesk Maya by NVidia. The PhysX plug-in brings in the simulation power of the PhysX physics engine to the Maya platform. This is a highly desirable option as the higher level operating principles and underlying simulation frameworks is the same as the standalone engine. More importantly, all of the joint constraints available in the standalone are available in the plug-in. The D6 joint is supported and with this highly configurable joint, other joints can be emulated. The PhysX plug-in is supported on Autodesk 3D Studio Max, Autodesk Maya and Autodesk SoftImage. While it exists as a plug-in for the former two, it is built into SoftImage. Due to a wider application support for PhysX, it is reasonable to use PhysX to integrate the system into the animation package than Nucleus, which is restricted to Autodesk Maya.

Internally Maya represents a scene as a collection of interconnected nodes. This interconnected structure is called a dependency graph. Nodes are processing elements that process the data coming in through its connections. The interconnections are visualized in the Maya hypergraph window (see Figure 5). Though the implicit graph structure of the graph might suggest compatibility for neural network creation, graph traversals and computation becomes inefficient as the size of the network increases. Moreover, the node itself is a non-lightweight class structure.

An alternative and faster approach is to directly encode the neural network as part of a single plug-in object that can either create a new neural network based on parameters provided by the user or load an existing network from a specification file. This way much of the existing code for the network can be re-used without partitioning them into dependency nodes. This would be far more efficient because dependency graph traversal is reduced to a bare minimum because the neural network exists within its own control class. The training process itself is performed off-line, so the network classes exist to load trained networks and act as the control program for the animation.

Adaptive Grasping Guidelines Based on Object Properties

Grasping has inherent task complexities, which the human brain performs with ease and at a sub-conscious level that can be broken into rules that can act as guidelines. Some examples would be direction of approach to the object, topology of the object, size of the object etc. From an implementation point of view, these rules are algorithmic and can be programmed. The difficulty is trying to quantify a psychological guideline like object purpose in a task. The way an object is grasped is also governed by the purpose of the object in question. Object purpose is a human assigned attribute and mathematically a mapping between object and its purpose is severely under constrained. This can be treated as an artificial intelligence problem. But for simplicity, geometric properties can also be used to create an adaptive grasping strategy that would be markedly different from any algorithmic approach. The center of mass (CoM) is one such property, which humans estimate to decide grasping points on the object (Lukos et al., 2007). As shown in Zhang and Tsuhan (2001), it is possible to extract object properties from its 3D mesh representation. The psuedocode in (Eberly, 2003) computes the CoM and the inertia tensor for a convex polyhedron. The vertices of the polyhedron and the indices to the vertices are passed in as an arrays. and the function computes the mass value, CoM vector and the inertia tensor respectively. Chapter 6 explains a line of action-based grasp contact point estimation and also an action line convergence point that can provide an early distribution of contact points on the surface of the object. The early distribution can be used as a starting point for adaptive modification of the points as a secondary process. The contact points on the surface are shifted to coincide better with the center of mass of the object. Distance functions can be used for good objective functions. The problem space can be represented mathematically for the adaptive system, which is minimizing the distance between the early contact points and the center of mass of the object. Stability of the grasp also can be estimated based on the number of contact points on the surface. But in a dynamic environment, a completely different approach has to be taken. Force vectors for the fingers have to be considered and opposability has to cancel the torques generated for a stable grasp. An adaptive neural network can be generated for this purpose. Samples would consist of various convex hulled rigid bodies, with the previous mentioned method used to determine contact points. The current system would be used to generate the muscle activations, except that the objective function would be to minimize the forces required for a grasp that maintains the object at a given distance from the ground plane.

Inverse Kinematics Style User Interface to Animate Physically Actuated Linkages

Without an intuitive interface to manipulate characters, animators cannot "direct" the behaviour of characters. Traditional key-frame approaches make it easy for animators to pose characters and adjust timing between key frames to perform the desired action. This is easy because key-frame approaches directly manipulate positional and orientation information. Simulation interfaces with respect to character animation is still in a nascent state, with many of packages like Maya, 3D Studio Max, SoftImage, Houdini etc relying on kinematic approaches. The most common type of simulation method employed in character animation is the Proportional Derivative controller. Being linear and feedback based, they are easily simulated and use positional information (orientation) as input. Also, the number of parameters to tune is manageable, increasing only with the number of joints or constraints on the linkage. This is a convenient method to integrate into a conventional kinematic pipeline because positional and orientation information from one can be used to drive the other. Simulation interfaces using PD for articulated characters are supported by third party physics plug-ins for popular packages like 3D Studio Max and Maya (NVidia PhysX plug-in). Predictive and suggestive interfaces are commonly used techniques for interactive control of physics-based character animation (Laszlo et al., 2005). Predictive interfaces provide the user with possible states forward in time depending on dynamically changing properties while suggestive interfaces present possible actions stemming from the current state or based on the editing action (Laszlo et al., 2005). Natural Motions Endorphin makes use of a predictive element as one of its features, showing the user the state of the simulation forward in time (Motion, 2011a). There are interfaces that map the mouse accelerometer to joint angles and the changing angles are used to drive the PD controller (Laszlo et al., 2000). The key aspect of PD control interface systems is devising a convenient angle input method, which is highly user friendly and tractable even with increased DOFs. High DOF input devices like data gloves can be used for virtual puppetry using physics-based animation but they used reduced character models to map the joint angles effectively to the data glove output (Laszlo et al., 2000). Keystroke mapping is also another interface method to drive physically actuated character models. Inverse Kinematics (IK) is used to calculate the desired joint angles, which in turn is input into the PD controllers (Laszlo et al., 2000). The invoking of the IK trajectory is based on keystroke mapping (Laszlo et al., 2000).

Having an inverse kinematic style user interface for active muscle control of physically actuated linked skeletons, would allow the animator to work with a familiar interface without the necessity for them to learn a new and complicated interface. The convention of key-framing and 3D animation tool-sets for character animation are based on a joint skeleton system as a primary interface for interaction with the characters to be animated. End-effector based interfaces for linked bodies have proved to be an efficient approach to animate articulated characters. The character joint skeleton also allows for mapping motion capture data. Therefore the adjustment time required to familiarise an artist with any new system can be reduced if the original IK style interface is either adopted or used as a template for a new interface.

The current system uses vector directional cosines (a set of three angles) as an input rather than single joint angles. The vectors are computed from the underlying skeleton and the corresponding muscle activations are computed by the system. In order to incorporate end-effector based activations, an additional training process is required that maps the computed muscle activations to end-effector positions of the linkage. Section 6.4.2 in Chapter 6 shows the results of a similar system for procedural control of the physical linkages.

Physics-based Deformations and Rendering

Visual realism in appearance is as important as realistic motion behaviours and that makes the difference between a good shot and bad shot. The rendering process happens towards the end of the pipeline (for visual effects as can be seen in Figure 4). Also for games it is the absolute end point of the pipeline, as the game engine takes the various forms of data and creates the final composite output in real-time (see Figure 3). Deformations happen at an intermediate stage at the point where animation is performed. For physics-based deformations, the ideal location along the pipeline is where middleware technology (physics engines, AI middleware) is integrated into the pipeline. The visual effects pipeline would have deformations grouped along with simulation and effects. The common type of deformations for simulated organic characters is surface skin deformations due to effects of muscle simulation. Usually surface deformations are generated through finite element methods, sub-surface deformations and other techniques. On some occasions geometric muscles (as opposed to physical muscles) are used to perform deformations (Albrecht et al., 2003). They are constructed from anatomy and make use of mass-spring systems that are attached to the skin (Albrecht et al., 2003). These methods are efficient when large surface area is to be deformed. The tendinuous structure simulated in this project are volume-less and single dimensional. And so deformations, which are localised sub-spaces of the deformation domain, are the kind that can be simulated best. These can be categorised as deformations caused due to the movement of the tendons underlying the skin surface. Standard deformation artifacts like rubber-tube effect can be avoided through the use of curve skeleton skinning that uses a b-spline to deform the mesh (Yang et al., 2006). The curve skeleton can also be customised for GPU processing. For rendering (in real-time), the deformations on dense meshes can be accelerated using latest Graphics Processing Unit (GPU) techniques. The muscle system (in the current iteration) is real-time existing within a real-time framework. So GPU-based rendering techniques can be used in specific areas, like shadow rendering, layered sub-surface scattering for improved skin shaders and soft body systems to simulate muscle jiggle, to improve visual realism in the system. Skinning methods, which are the fundamental deformation methods, are already implemented on the GPU (Fernando, 2004). With the level of GPU capability today, Bidirectional Reflectance Distribution Functions (BRDF) can be calculated real-time for realistic rendering of skin also (Nguyen, 2007).

Integration into an existing pipeline system - Games and Visual Effects

For any new animation or simulation system, it is always important to analyse how it fits into an existing pipeline. The lesser it changes the existing pipeline the better compatibility it has with the pipeline. Games are autonomous in that the entities are imbued with AI logic and that logic dictates animation playback of the entities. So adaptive animation systems are ideal in the largely automated world of games. In the pipeline, the training process for adaptive muscle activation is always an offline process, preferably external to the pipeline, though acquiring resources from the pipeline (see Figure 6).

The system will exist as a middle-ware and therefore after the training process can be added into the middle-ware integration part of the pipeline, which is in the Program/Build layer of the pipeline (see Figure 3).

The middle-ware commonly used are AI path-finding modules and physics engines. Therefore, at that juncture the muscle activation system exists as an additional component of the game engine runtime. If the characters in the game

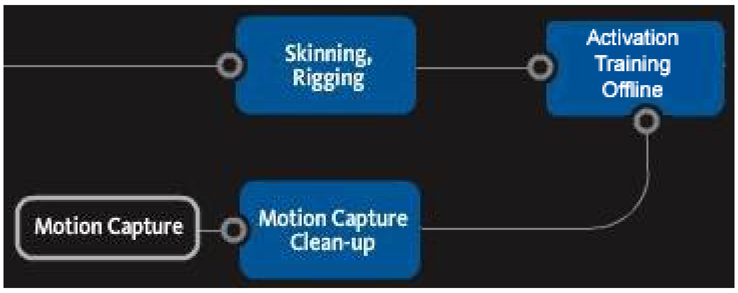


Figure 6: Modified segment of the games pipeline from Figure 3. The activation training is an offline process not directly a part of the pipeline. But existing pipeline processes are used for the training..

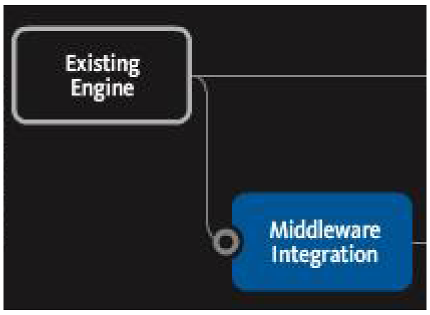


Figure 7: Middleware integration with the game engine. At this point the activation system would become a part of the main pipeline.

follow more or less very similar muscular anatomy, then there is no necessity for duplication of the activation networks. Musculature layout can be morphed in three dimensions between characters using the anatomical landmarks on the underlying skeleton structure and the muscle locators. The morphing process would come into play after the rigging process. Rigging would be performed on a base character of artist choosing and then morphed to fit the similar characters grouped beforehand. In a game there is little intervention by the animators once the game is built. The engine autonomously takes care of all the assets distributing the tasks to its sub-systems. Intelligent animation is performed based on the differing states of the game world, either through the action of the user or by the action of bots.

In visual effects, artist intervention takes place almost throughout the entire pipeline. Automation is possible only during asset creation. The final output is static where there is no scope for any interactive behaviour.

The animator would require complete control in the animation package he or she uses. Thus a complete integration of the current system (modified) into the core animation package would achieve that. This would be a culmination point of physics-based animation and control interface that would help the animator to create physics-based animation in a traditional manner. A plausible interface for

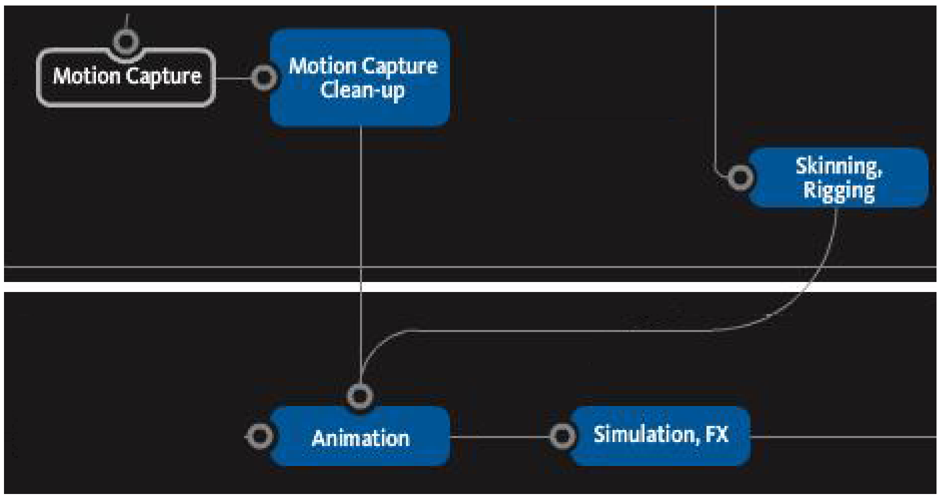


Figure 8: The various processes that is required in the training process, from the visual effects pipeline. Extracted from the original pipeline in Figure 4.

this purpose is given in sub-section 7.3.

Summary

In addition to the given improvements, there are low-level enhancements that can be programmed into the system like taking advantage of multi-cores of the system and use multi-threading to perform operations in parallel. The system that forms the subject of this research has all the core elements of an animation pipeline, namely, modelling, rigging, simulation and rendering. Thus it is entirely possible to combine the various parts of the system into a comprehensive tool set that exists as part of an animation package like Maya. A singular tool-set is easier to integrate into the production pipeline than a distributed set of applications. It also helps in keeping the existing pipeline relatively unchanged with a fewer number of processes.

Bibliography

Adamczyk, M. M. and Crago, P. E., 2000. Simulated feedforward neural network coordination of hand grasp and wrist angle in a neuroprosthesis. In *Rehabilitation Engineering, IEEE Transactions on*, volume 8, p. 297–304.

Afshar, P. and Matsuoka, Y., 2004. Neural-based control of a robotic hand: Evidence for distinct muscle strategies. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, p. 4633–4638 Vol. 5. IEEE. ISBN 0780382323.

Ageia. Physx documentation, 2006. Date of access: 02/02/2010.

Ahsan, M. and Ibrahimy, M. I., 2009. EMG signal classification for human computer interaction: a review. In *European Journal of Scientific Research*, volume 33, p. 480–501.

Albrecht, I., Haber, J., and Seidel, H. P., 2003. Construction and animation of anatomically based human hand models. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, p. 98–109. Eurographics Association. ISBN 1581136595.

Allen, B., Curless, B., and Popovi, Z., 2003. The space of human body shapes: reconstruction and parameterization from range scans. In *ACM Transactions on Graphics (TOG)*, volume 22, p. 587–594. ACM. ISBN 1581137095.

Annunziato, M., Bertini, I., Lucchetti, M., and Pizzuti, S., 2003. Evolving weights and transfer functions in feed forward neural networks. In *Proc. EUNITE2003, Oulu, Finland.*

Aubel, A. and Thalmann, D., 2001. Interactive modeling of the human musculature. In *Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings*, p. 167–255. IEEE. ISBN 0780372379.

Autodesk. Nucleus in autodesk maya - a white paper, 2009. URL http://images.autodesk.com/adsk/files/autodeskmaya_nucleus_whitepaper.pdf. Date of access: 05/10/2011.

Autodesk. Autodesk kynapse, 2011a. URL http://usa.autodesk.com/adsk/servlet/pc/index?id=11390544&siteID=123112. Date of access: 05/10/2011.

Autodesk. Modern pipeline, 2011b. URL http://usa.autodesk.com/adsk/servlet/index?id=16116742&siteID=123112. Date of access: 05/10/2011.

Barber, C. B., Dobkin, D. P., and Huhdanpaa, H., 1996. The quickhull algorithm for convex hulls. In *ACM Transactions on Mathematical Software (TOMS)*, volume 22, p. 469–483.

Brian, A. G. and Marcus, G. P., 2000. The obstacle-set method for representing muscle paths in musculoskeletal models. In *Computer methods in biomechanics and biomedical engineering*, volume 3, p. 1–30.

Burstedt, M. K. O., Edin, B. B., and Johansson, R. S., 1997. Coordination of fingertip forces during human manipulation can emerge from independent neural networks controlling each engaged digit. In *Experimental brain research*, volume 117, p. 67–79.

Camazine, S., 2003. *Self-organization in biological systems.* Princeton Univ Pr. ISBN 0691116245.

Chadwick, J. E., Haumann, D. R., and Parent, R. E., 1989. Layered construction for deformable animated characters. In *ACM Siggraph Computer Graphics*, volume 23, p. 243–252. ACM. ISBN 0897913124.

Chaitow, L. and DeLany, J. W., 2000. *Clinical application of neuromuscular techniques. Vol. 2, the lower body.* Churchill Livingstone Elsevier.

Ciocarlie, M., Miller, A., and Allen, P., 2005. Grasp analysis using deformable fingers. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, p. 4122–4128. IEEE. ISBN 0780389123.

Day, S., 2002. Important factors in surface EMG measurement. In *Bortec Biomedical Ltd publishers*, p. 1–17.

Deliagina, T. G., Pavlova, E. L., Rossignol, S., Dubuc, R., Gossard, J., Saltiel, P., Wyler-duda, K., Ajemian, R. J., Bizzi, E., and Zelenin, P. V., 2008. Neural bases of postural control.

Eberly, D., 2003. Polyhedral mass properties (revisited). In *www. magic-sofiware. com/Documentation/PolyhedratMassProperties. pdf.*

Eberly, D. H. and Shoemake, K., 2004. *Game physics.* Morgan Kaufmann. ISBN 1558607404.

Eccles, J. C., Eccles, R. M., and Lundberg, A., 1957. The convergence of monosynaptic excitatory afferents on to many different species of alpha motoneurones. In *The Journal of Physiology*, volume 137, p. 22–50.

Edsall, J. Animation blending: Achieving inverse kinematics and more, 2003. URL http://www.gamasutra.com/view/feature/3456/animation_blending_achieving_.php. Date of access: 05/02/2012.

Enoka, R. M., 2008. *Neuromechanics of human movement.* Human Kinetics Publishers. ISBN 0736066799.

Ericson, C., 2005. *Real-time collision detection*, volume 1. Morgan Kaufmann. ISBN 1558607323.

Faloutsos, P., van de Panne, M., and Terzopoulos, D., 2001. Composable controllers for physics-based character animation. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, p. 251–260. ACM. ISBN 158113374X.

Fattal, R. and Lischinski, D., 2006. Pose controlled physically based motion. In *Computer Graphics Forum*, volume 25, p. 777–787. Wiley Online Library. ISBN 1467-8659.

Fernando, R., 2004. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education. ISBN 0321228324.

Ferrari, C. and Canny, J., 1992. Planning optimal grasps. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, p. 2290–2295 vol. 3. IEEE. ISBN 0818627204.

Fogel, D. B., 2006. *Evolutionary computation: toward a new philosophy of machine intelligence*, volume 1. Wiley-IEEE Press. ISBN 0471669512.

Francik, J. and Szarowicz, A., 2005. Character animation with decoupled behaviour and smart objects. In *6th International Conference on Computer Games CGAIMS, Louisville, Kentucky, USA*.

Frank, R. J., Davey, N., and Hunt, S. P., 2001. Time series prediction and neural networks. In *Journal of Intelligent & Robotic Systems*, volume 31, p. 91–103.

Garcia, J. S. D., Avila, S. L., and Carpes, W. P., 2006. Introduction to optimization methods: A brief survey of methods. In *IEEE Multidisciplinary Engineering Education Magazine*, volume 1.

Gazzaniga, M. S., 2004. *The cognitive neurosciences*. The MIT Press. ISBN 0262072548.

Goldberg, D. E., 1989. *Genetic algorithms in search, optimization, and machine learning*. Addison-wesley. ISBN 0201157675.

Goldfinger, E., 1991. *Human anatomy for artists: The elements of form*. Oxford University Press, USA. ISBN 0195052064.

Gray, H. Gray's anatomy, 2006. URL http://www.bartleby.com/107/67.html. Date of access: 05/04/2010.

Gregory, J., Lander, J., and Whiting, M., 2009. *Game engine architecture*. AK Peters. ISBN 1568814135.

Gritz, L. and Hahn, J. K., 1995. Genetic programming for articulated figure motion. In *The Journal of Visualization and Computer Animation*, volume 6, p. 129–142.

Grzeszczuk, R. *Neuroanimator: Fast neural network emulation and control of physics-based models*. PhD thesis, Citeseer, 1998.

Hatcher, A., 2002. *Algebraic Topology*. The Cambridge University Press. ISBN 730210588X.

Havok. Havok physics, 2011. URL http://www.havok.com/products/physics. Date of access: 05/10/2011.

Havok. Customer projects: Games, 2012. URL http://www.havok.com/customer-projects/games. Date of access: 05/02/2012.

Heaton, J., 2005. *Introduction to neural networks with Java*. Heaton Research Inc, 2nd edition. ISBN 097732060X.

Hogan, N., 1984. Adaptive control of mechanical impedance by coactivation of antagonist muscles. In *Automatic Control, IEEE Transactions on*, volume 29, p. 681–690.

Hogan, N. and Flash, T., 1987. Moving gracefully: quantitative theories of motor coordination. In *Trends in Neurosciences*, volume 10, p. 170–174.

Holmes, J. W., 2006. Teaching from classic papers: Hill's model of muscle contraction. In *Advances in physiology education*, volume 30, p. 67–72.

Holmes, P., Koditschek, D. E., and Guckenheimer, J., 2006. The dynamics of legged locomotion: Models, analyses, and challenges.

Huxley, A. F., 1957. Muscle structure and theories of contraction. In *Progress in biophysics and biophysical chemistry*, volume 7, p. 255.

IvyRose. The structure of muscle filaments, August 2010. URL http://www.ivy-rose.co.uk/HumanBody/Muscles/Muscle_Filaments.php. Date of access: 23/08/2010.

Jakobsen, T., 2001. Advanced character physics. In *Game Developers Conference*, p. 383–401.

Jeannerod, M., 1981. Intersegmental coordination during reaching at natural visual objects. In *Attention and performance IX*, volume 9, p. 153–168.

Jeannerod, M., Paulignan, Y., and Weiss, P., 1998. Grasping an object: one movement, several components. In *Novartis Foundation Symposium 218Sensory Guidance of Movement*, p. 5–20. Wiley Online Library. ISBN 0470515562.

Jin, G. and Hahn, J., 2005. Adding hand motion to the motion capture based character animation. In *Advances in Visual Computing*, p. 17–24.

Jordan, M. I., 1988. Supervised learning and systems with excess degrees of freedom. In *COINS Technical Report*, number 88-27, p. 1–41.

Kallmann, M. *Object interaction in real-time virtual environments*. PhD thesis, Citeseer, 2001.

Kelso, J. A., 1995. *Dynamic patterns: The self-organization of brain and behavior*. The MIT Press. ISBN 0262611317.

Kim, J., Cordier, F., and Magnenat-Thalmann, N., 2000. Neural network-based violinist's hand animation. In *Computer Graphics International, 2000. Proceedings*, p. 37–41. IEEE. ISBN 0769506437.

Kirkpatrick, D., Mishra, B., and Yap, C. K., 1992. Quantitative steinitz's theorems with applications to multifingered grasping. In *Discrete & Computational Geometry*, volume 7, p. 295–318.

Komatsu, K., 1988. Human skin model capable of natural shape variation. In *The visual computer*, volume 3, p. 265–271.

Konczak, J., 2004. Neural development and sensorimotor control.

Kry, P. G. and Pai, D. K., 2006. Interaction capture and synthesis. In *ACM Transactions on Graphics (TOG)*, volume 25, p. 872–880.

Kuiken, T. A., Li, G., Lock, B. A., Lipschutz, R. D., Miller, L. A., Stubblefield, K. A., and Englehart, K. B., 2009. Targeted muscle reinnervation for real-time myoelectric control of multifunction artificial arms. In *JAMA: the journal of the American Medical Association*, volume 301, p. 619.

Kuperstein, M., 1988. Neural model of adaptive hand-eye coordination for single postures. In *Science*, volume 239, p. 1308–1311.

Lassabe, N., Luga, H., and Duthen, Y., 2007. A new step for artificial creatures. In *Artificial Life, 2007. ALIFE'07. IEEE Symposium on*, p. 243–250. IEEE. ISBN 142440701X.

Laszlo, J., Neff, M., and Singh, K., 2005. Predictive feedback for interactive control of physicsbased characters. In *Computer Graphics Forum*, volume 24, p. 257–265. Wiley Online Library. ISBN 1467-8659.

Laszlo, J., van de Panne, M., and Fiume, E., 2000. Interactive control for physically-based animation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, p. 201–208. ACM Press/Addison-Wesley Publishing Co. ISBN 1581132085.

Lederman, S. J. and Wing, A. M., 2003. Perceptual judgement, grasp point selection and object symmetry. In *Experimental brain research*, volume 152, p. 156–165.

Lee, D., Glueck, M., Khan, A., Fiume, E., and Jackson, K. A survey of modeling and simulation of skeletal muscle. Technical report, Citeseer, 2009.

Lee, S. H. and Terzopoulos, D., 2006. Heads up!: biomechanical modeling and neuromuscular control of the neck. In *ACM Transactions on Graphics (TOG)*, volume 25, p. 1188–1198. ACM. ISBN 1595933646.

Levin, D. I., Gilles, B., Madler, B., and Pai, D. K., 2008. A fiber tracking method for building patient specific dynamic musculoskeletal models from diffusion tensor data. In *MICCAI Workshop on Computational Diffusion MRI, page (to appear)*.

Lewis, J. P., Cordner, M., and Fong, N., 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, p. 165–172. ACM Press/Addison-Wesley Publishing Co. ISBN 1581132085.

Li, Y. and Pollard, N. S., 2005. A shape matching algorithm for synthesizing humanlike enveloping grasps. In *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*, p. 442–449. IEEE. ISBN 0780393201.

Lin, J., Wu, Y., and Huang, T. S., 2000. Modeling the constraints of human hand motion. In *Human Motion, 2000. Proceedings. Workshop on*, p. 121–126. IEEE. ISBN 0769509398.

Lukos, J., Ansuini, C., and Santello, M., April 2007. Choice of contact points during multidigit grasping: effect of predictability of object center of mass location. In *The Journal of neuroscience*, volume 27, p. 3894–3903.

Macpherson, J. M., 1991. How flexible are muscle synergetics? In *Motor Control: Concepts and Issues*.

Maurel, W., Wu, Y., Thalmann, D., and Thalmann, N. M., 1998. *Biomechanical models for soft tissue simulation*. Springer-Verlag Berlin (Germany)/Heidelberg (NY). ISBN 3540637427.

Miall, R. C., Weir, D. J., Wolpert, D. M., and Stein, J. F., 1993. Is the cerebellum a smith predictor? In *Journal of motor behavior*, volume 25, p. 203–216.

Miconi, T. and Channon, A., 2006. An improved system for artificial creatures evolution. In *Proceedings of Artificial Life X*, p. 255–261.

Miller, A., Allen, P., Santos, V., and Valero-Cuevas, F., 2005. From robotic hands to human hands: a visualization and simulation engine for grasping research. In *Industrial Robot: An International Journal*, volume 32, p. 55–63.

Miller, A. T., Knoop, S., Christensen, H. I., and Allen, P. K., 2003. Automatic grasp planning using shape primitives. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, p. 1824–1829 vol. 2. IEEE. ISBN 0780377362.

Millington, I., 2007. *Game physics engine development*. Morgan Kaufmann. ISBN 012369471X.

Molina-Vilaplana, J., Feliu-Batlle, J., and Lpez-Coronado, J., 2007. A modular neural network architecture for step-wise learning of grasping tasks. In *Neural networks*, volume 20, p. 631–645.

Motion, N. Natural motion endorphin, 2011a. URL http://www.naturalmotion.com/endorphin. Date of access: 05/10/2011.

Motion, N. Natural motion morpheme, 2011b. URL http://www.naturalmotion.com/morpheme. Date of access: 05/10/2011.

Murai, A., Yamane, K., and Nakamura, Y., 2008. Modeling and identification of human neuromusculoskeletal network based on biomechanical property of muscle. In *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, p. 3706–3709. IEEE. ISBN 1424418143.

MussaIvaldi, F. A. and Bizzi, E., 2000. Motor learning through the combination of primitives. In *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, volume 355, p. 1755–1769.

Nakamura, M., Miyawaki, C., Matsushita, N., Yagi, R., and Handa, Y., 1998. Analysis of voluntary finger movements during hand tasks by a motion analyzer. In *Journal of Electromyography and Kinesiology*, volume 8, p. 295–303.

Napier, J. R., 1956. The prehensile movements of the human hand. In *Surger*, volume 38, p. 902–913.

Napier, J. R. and Tuttle, R., 1993. *Hands*. Princeton Univ Pr. ISBN 0691025479.

Neff, M. and Fiume, E., 2002. Modeling tension and relaxation for computer animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, p. 81–88. ACM. ISBN 1581135734.

Negnevitsky, M., 2005. *Artificial intelligence: a guide to intelligent systems*. Addison-Wesley Longman. ISBN 0321204662.

Nelson, A. L., Barlow, G. J., and Doitsidis, L., 2009. Fitness functions in evolutionary robotics: A survey and analysis. In *Robotics and Autonomous Systems*, volume 57, p. 345–370.

Nguyen, H., 2007. *GPU Gems 3*. Addison-Wesley Professional. ISBN 0321545427.

Nishikawa, K., Biewener, A. A., Aerts, P., Ahn, A. N., Chiel, H. J., Daley, M. A., Daniel, T. L., Hale, M. E., Hedrick, T. L., and Lappin, A. K., 2007. Neuromechanics: an integrative approach for understanding motor control. In *Integrative and Comparative Biology*, volume 47, p. 16–54.

Nolfi, S. and Floreano, D., 2000. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. The MIT Press. ISBN 0262140705.

Nolfi, S. and Parisi, D., 2002. Evolution of artificial neural networks. In *In Handbook of brain theory and neural networks*. Citeseer.

NVidia. Physx:games, 2012. URL http://www.geforce.com/Hardware/Technologies/PhysX/games. Date of access: 05/02/2012.

Ohnishi, K., Weir, R. F., and Kuiken, T. A., 2007. Neural machine interfaces for controlling multifunctional powered upper-limb prostheses. In *Expert review of medical devices*, volume 4, p. 43–53.

OpenSim. Simm, 1990. URL http://www.musculographics.com/products/simm.html. Date of access: 05/10/2012.

Oyama, E., Maeda, T., Gan, J. Q., Rosales, E. M., MacDorman, K. F., Tachi, S., and Agah, A., 2005. Inverse kinematics learning for robotic arms with fewer degrees of freedom by modular neural network systems. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, p. 1791–1798. IEEE. ISBN 0780389123.

Oztop, E., Bradley, N. S., and Arbib, M. A., 2004. Infant grasp learning: a computational model. In *Experimental Brain Research*, volume 158, p. 480–503.

Parent, R., 2007. *Computer animation: algorithms and techniques.* Morgan Kaufmann Pub. ISBN 0125320000.

Park, S. I. and Hodgins, J. K., 2006. Capturing and animating skin deformation in human motion. In *ACM Transactions on Graphics (TOG)*, volume 25, p. 881–889. ACM. ISBN 1595933646.

Parry-Barwick, S. and Bowyer, A., 1993. Is the features interface ready? In *Directions in Geometric Computing, Ed. Martin R., Information Geometers Ltd, UK, Cap*, volume 4, p. 129–160.

Pixologic. Zbrush, August 2010. URL http://www.pixologic.com/zbrush/. Date of access: 05/10/2011.

Pollard, N. S., 1996. Synthesizing grasps from generalized prototypes. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 3, p. 2124–2130 vol. 3. IEEE. ISBN 0780329880.

Pollard, N. S., 2004. Closure and quality equivalence for efficient synthesis of grasps from examples. In *The International Journal of Robotics Research*, volume 23, p. 595.

Pollard, N. S. and Gilbert, R. C., 2002. Tendon arrangement and muscle force requirements for human-like force capabilities in a robotic finger. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, p. 3755–3762 vol. 4. IEEE. ISBN 0780372727.

Pollard, N. S. and Zordan, V. B., 2005. Physically based grasping control from example. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, p. 311–318. ACM.

Popovi, J., Seitz, S. M., Erdmann, M., Popovi, Z., and Witkin, A., 2000. Interactive manipulation of rigid body simulations. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, p. 209–217. ACM Press/Addison-Wesley Publishing Co. ISBN 1581132085.

Prentice, S. D., Patla, A. E., and Stacey, D. A., 1998. Simple artificial neural network models can generate basic muscle activity patterns for human locomotion at different speeds. In *Experimental Brain Research*, volume 123, p. 474–480.

Prives, M., Lysenkov, N., and Bushkovich, V., 1989. *Human Anatomy Volume I.* Mir Publishers. ISBN 5030007732.

Reil, T. and Husbands, P., 2002. Evolution of central pattern generators for bipedal walking in a real-time physics environment. In *Evolutionary Computation, IEEE Transactions on*, volume 6, p. 159–168.

Rezzoug, N. and Gorce, P., 2003. A biocybernetic method to learn hand grasping posture. In *Kybernetes*, volume 32, p. 478–490.

Rijpkema, H. and Girard, M., 1991. Computer animation of knowledge-based human grasping. In *ACM SIGGRAPH Computer Graphics*, volume 25, p. 339–348. ACM. ISBN 0897914368.

Ruebsamen, G. Evolving efficient locomotive strategies in embodied agents. masters thesis, California State University, Long Beach, Dept. of Computer Engineering and Computer Science, 2002.

Schneiderman, B. and Plaisant, C., 1998. *Designing the user interface.* Addison-Wesley Longman. ISBN 032122325X.

Scott, J. Computational geometry, 2012. URL `http://www.blackpawn.com/texts/pointinpoly/default.html`. Date of access: 05/10/2012.

Senaratna, N. I., 2005. Genetic algorithms: The Crossover-Mutation debate.

Shadmehr, R. and Wise, S. P., 2005. *The computational neurobiology of reaching and pointing: a foundation for motor learning.* The MIT press. ISBN 0262195089.

Shim, J. K., Latash, M. L., and Zatsiorsky, V. M., 2005. Prehension synergies in three dimensions. In *Journal of neurophysiology*, volume 93, p. 766.

Sifakis, E., Neverov, I., and Fedkiw, R., 2005. Automatic determination of facial muscle activations from sparse motion capture marker data. In *ACM Transactions on Graphics (TOG)*, volume 24, p. 417–425. ACM. ISBN 0730-0301.

Sims, K., 1994. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, p. 15–22. ACM. ISBN 0897916670.

Smeets, J. B. and Brenner, E., 1999. A new view on grasping. In *Motor Control*, volume 3, p. 237–271.

Smith, R. L. *Intelligent motion control with an artificial cerebellum.* PhD thesis, DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING, UNIVERSITY OF AUCKLAND, 1998a.

Smith, R. Open dynamics engine, 1998b. URL `http://ode.org/ode.html`. Date of access: 05/10/2011.

Sueda, S., Kaufman, A., and Pai, D. K., 2008. Musculotendon simulation for hand animation. In *ACM Transactions on Graphics (TOG)*, volume 27, p. 83. ACM. ISBN 1450301126.

Teran, J., Sifakis, E., Blemker, S. S., Ng-Thow-Hing, V., Lau, C., and Fedkiw, R., 2005. Creating and simulating skeletal muscle from the visible human data set. In *Visualization and Computer Graphics, IEEE Transactions on*, volume 11, p. 317–328.

Thalmann, D., Shen, J., and Chauvineau, E., 1996. Fast realistic human body deformations for animation and VR applications. In *Computer Graphics International, 1996. Proceedings*, p. 166–174. IEEE. ISBN 0818675187.

Thelen, D. G., Anderson, F. C., and Delp, S. L., 2003. Generating dynamic simulations of movement using computed muscle control. In *Journal of Biomechanics*, volume 36, p. 321–328.

Ting, L. H. and McKay, J. L., 2007. Neuromechanics of muscle synergies for posture and movement. In *Current opinion in neurobiology*, volume 17, p. 622–628.

Todorov, E., 2000. Direct cortical control of muscle activation in voluntary arm movements: a model. In *nature neuroscience*, volume 3, p. 391–398.

Tsang, W., Singh, K., and Fiume, E., 2005. Helping hand: an anatomically accurate inverse dynamics solution for unconstrained hand motion. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, p. 319–328. ACM. ISBN 1595931988.

Valero-Cuevas, F. J. and Lipson, H., 2004. A computational environment to simulate complex tendinous topologies. In *Engineering in Medicine and Biology Society, 2004. IEMBS'04. 26th Annual International Conference of the IEEE*, volume 2, p. 4653–4656. IEEE. ISBN 0780384393.

Valero-Cuevas, F. J., Yi, J. W., Brown, D., McNamara, R. V., Paul, C., and Lipson, H., 2007. The tendon network of the fingers performs anatomical computation at a macroscopic scale. In *Biomedical Engineering, IEEE Transactions on*, volume 54, p. 1161–1166.

Van de Panne, M. and Fiume, E., 1993. Sensor-actuator networks. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, p. 335–342. ACM. ISBN 0897916018.

Van de Panne, M., Kim, R., and Fiume, E., 1994. Virtual wind-up toys for animation. In *Graphics Interface*, p. 208–208. Citeseer. ISBN 0713-5424.

Van Verth, J. M. and Bishop, L. M., 2008. *Essential mathematics for games and interactive applications: a programmer's guide*. Morgan Kaufmann Pub. ISBN 0123742978.

Watson, R. A., Hornby, J. B., and Pollack, J., 1998. Modeling building block interdependency. In *Proceedings of Parallel Problem Solving from Nature V (PPSN V)*, p. 97–106.

Weinstein, R., Guendelman, E., and Fedkiw, R., 2008. Impulse-based control of joints and muscles. In *Visualization and Computer Graphics, IEEE Transactions on*, volume 14, p. 37–46.

Weiss, P. and Jeannerod, M., 1998. Getting a grasp on coordination. In *News in physiological sciences: an international booktitle of physiology produced jointly by the International Union of Physiological Sciences and the American Physiological Society*, volume 13, p. 70.

Witkin, A., 1997. Physically based modeling: Principles and practice constrained dynamics. In *SIGGRAPH Course notes*, p. 11–21.

Witkin, A. and Kass, M., 1988. Spacetime constraints. In *ACM Siggraph Computer Graphics*, volume 22, p. 159–168. ACM. ISBN 0897912756.

Xia, Y., Wang, J., and Fok, L. M., 2004. Grasping-force optimization for multifingered robotic hands using a recurrent neural network. In *Robotics and Automation, IEEE Transactions on*, volume 20, p. 549–554.

Xu, J., Gannon, P. J., Emmorey, K., Smith, J. F., and Braun, A. R., 2009. Symbolic gestures and spoken language are processed by a common neural system. In *Proceedings of the National Academy of Sciences*, volume 106, p. 20664–20669.

Yang, X., Somasekharan, A., and Zhang, J. J., 2006. Curve skeleton skinning for human and creature characters. In *Computer Animation and Virtual Worlds*, volume 17, p. 281–292.

Yao, X., 1999. Evolving artificial neural networks. In *Proceedings of the IEEE*, volume 87, p. 1423–1447.

Zajac, F. E., 1989. Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. In *Critical reviews in biomedical engineering*, volume 17, p. 359.

Zatsiorsky, V. M., Gregory, R. W., and Latash, M. L., 2002a. Force and torque production in static multifinger prehension: biomechanics and control. i. biomechanics. In *Biological cybernetics*, volume 87, p. 50–57.

Zatsiorsky, V. M., Gregory, R. W., and Latash, M. L., 2002b. Force and torque production in static multifinger prehension: biomechanics and control. II. control. In *Biological cybernetics*, volume 87, p. 40–49.

Zhang, C. and Tsuhan, C., 2001. Efficient texture extraction for 2d/3d objects in mesh representation. In *Proceedings of Image Processing International Conference 2001*, volume 3.