Parametric Dynamics - A method for addition of dynamic motion to computer animated human characters

Pablo de Heras Ciechomski, D–96

July 6, 2001

Abstract

Virtual human characters in games are constanly interacting with their game environment and need to be animated accordingly. This was previously done by pre-recording all animations made possible by the interactions, giving an exponential growth curve in the number of required pre-recorded animations. Parametric dynamics expands upon pre-recorded animations by superimposing real-time simulated animations. These simulated animations are generated using Featherstone's forward dynamics algorithm and are classified into dynamic situations. A dynamic situation is controlled by a set of parameters tunable by artists, that regulates how the simulated animation will behave. A mathematical structure called the bone pose is developed along with a blending pipeline, to formalize the superposing of animations.

The parametric dynamics method is developed and tested, using a set of dynamic situations on a virtual human character, and the results are analysed. State-of-the-art virtual human animation and dynamics algorithms are scrutinized. Future work and enhancements are discussed.

To Mom and Dad

My supervisors

Ph. D. student Mathias Haage at Lund Institute of Technology
Martin Rystrand at Massive Entertainment AB

Thank you for your inspiration and help

Ph. D. Anders Robertsson at Lund Institute of Technology
Ph. D. Klas Nilsson at Lund Institute of Technology
Ph. D. Brian Vincent Mirtich from USA
Ph. D. Andrés Jaramillo-Botero from Colombia
Ph. D. Tomas Möller at Chalmers Institute of Technology
Ph. D. Magnus Olsson at Lund Institute of Technology
Niklas Hansson at Massive Entertainment AB
Martin Walfisz at Massive Entertainment AB
M. Sc. Daniel Jeppsson at Southend Interactive AB
Massive Entertainment AB
Erik Johnson at Valve Software

My opponents at Lund Institute of Technology

M. Sc. Björn Aspernäs
M. Sc. Emil Naef

Special thanks to my senior high school math and physics teacher

Rolf Olsson at Spyken in Lund

Contents

1 Introduction **1**
- 1.1 Animation Techniques . . . 1
- 1.2 Kinematics . . . 3
- 1.3 Dynamics . . . 4
- 1.4 Problem . . . 5
- 1.5 Approach . . . 6
- 1.6 Report Layout . . . 6

2 Dynamics **8**
- 2.1 Point Mass Dynamics . . . 10
- 2.2 Propagation of Dynamic State . . . 11
- 2.3 Particle System Dynamics . . . 13
- 2.4 Rigid Body Dynamics . . . 13
- 2.5 Multibody Forward Dynamics Algorithms . . . 15
- 2.6 Serial Link Forward Dynamics . . . 17
  - 2.6.1 Propagation of Velocity and Acceleration . . . 17
  - 2.6.2 Expressing Quantities in Different Frames . . . 23
- 2.7 Spatial Algebra . . . 25
  - 2.7.1 Transformation of Velocity and Force . . . 26
  - 2.7.2 Transformation of Acceleration . . . 28
- 2.8 The Featherstone Algorithm . . . 30
  - 2.8.1 Tree Topologies . . . 32
  - 2.8.2 Forward Dynamics Algorithm . . . 33

3 Animation **36**
- 3.1 Rigid Body Animation . . . 36
  - 3.1.1 Parent-Child Hierarchy . . . 38
- 3.2 Skinning Algorithms . . . 38
  - 3.2.1 Skeleton Subspace Deformation or "Skinning" . . . 41
- 3.3 Parametric Animation . . . 44
  - 3.3.1 Interpolation of Rotations - Quaternions . . . 45

3.3.2 Bone Poses 47
3.3.3 Key Reduction 47
3.3.4 Animation Blending 48
3.4 Inverse Kinematics 50
3.4.1 Cyclic Coordinate Descent - CCD 50
3.4.2 Jacobian Transpose 50
3.4.3 Comparison of the Two Methods 51
3.4.4 The Problem of Control 51

4 Parametric Dynamics **53**
4.1 Dynamic Situations 54
4.2 The Blending Pipeline 55
4.3 Feeding the Dynamics Simulator 56
4.3.1 A Dynamic Link 56
4.3.2 Dynamic Links' Relation to Joints 58
4.3.3 Dynamic Links' Relation to Bones 59
4.3.4 Dynamic Links' Relation to Joints II 60
4.4 Feeding the Blending Pipeline 62

5 Tests **63**
5.1 Test Parameters 64
5.2 Tests 65
5.2.1 Test I - Walking and shot in shoulder, GM Good . . . 67
5.2.2 Test II - Walking and shot in leg, GM Bad 69
5.2.3 Test III - Walking and shot in chest, GM Good 71
5.2.4 Tests IV-VI - Waving, GM Good except for Test V . . 73
5.3 Analysis 73

6 Further Work **77**

7 Conclusion **78**

Chapter 1

Introduction

I wanted to do a thesis about some game programming aspect. Character animation was one of three thesis proposals I discussed with Massive Entertainment. I had been involved in a fighting game project in my spare time. In that game called "Kung Fu Fighters", the main problem was to animate the characters as life like as possible. Something I had been missing in fighting games was dynamics or physics, which ignited my interest in the subject. To be able to understand the discussions in the thesis one needs to know some basic techniques in contemporary character animation that are discussed in this introduction. These techniques cover areas such as mathematics and physics. I start by putting computer graphics animation of characters in a historical context.

1.1 Animation Techniques

Disney Inc was one of the pioneers of animation and many technical terms come from the solutions they invented. In a video camera of the NTSC standard, the world is sampled 30 times per second. Objects that move in the physical world are thus sampled at different positions in time. If we play back the recorded film we perceive it as a motion. The cartoon "Snow White and the Seven Dwarfs" by Disney Inc from 1937 was the first feature-length cartoon. In it the actors did not actually exist so they had to be drawn on pieces of paper. To have one second of film, you would need 25 or so pieces of paper, where the character moved a little bit in every piece of paper. Cartoons most often use a frame rate of 24 to 30 frames per second. If the frame rate is lower than this the cartoon or video recording will look chopped up, just like in the movies from the 1920's. It is quite time consuming for one person to draw all those pictures called frames. What Disney Inc did to

speed up production and keep a good quality, was to let the most talented artists draw key frames. Key frames have a lower sample rate of 5 per second or less, so to be able to get the required 25 you have to draw the frames in between. The in between frames were drawn by the less talented artists called in-betweeners. When all the frames were ready the inkers filled them in with colour. This system has actually changed very little for computer-generated animations where the computer is now the in-betweener and the inker. Key frames are still generated by people. This history is treated in [59].

King Kong^{TM} the movie from 1933 by Merian Cooper and Ernest B. Schoedsack, needed non-existing animals that moved, but could not use cartoons because they would not be believable for the theatre audience. King Kong^{TM} was in fact an 18 inch tall miniature model, covered with rabbit hair. The special effects department in the film studio, used puppets of the animals that they animated by moving them a little, then taking a picture and repeating the process. The technique is called stop motion and has been used for many films such as the early Star Wars^{TM} movies, and is still used in animated clay films.

When computer games that immersed the player into a three dimensional world, having virtual actors, started to appear in the 90's they used the tools that were available at the time. The most common technique was to represent the actors using hierarchic rigid body animation. A rigid body animation is a mesh and a way to scale, orient and position it. One way to transform the mesh is to use frames. Frames in this context are not Disney Inc's frames but rather mathematical tools. A frame is a translation and a rotation. A frame transforms the vertices in the mesh. Rigid body animation is good for representing animation of objects that do not deform such as machine parts in engines or robots. Ever wondered why early computer games had so many robots?

To animate a character with rigid body animation one needs a mesh for every limb in the body and a frame to accompany it. The limbs are animated using key frames just as the Disney Inc key frames. In between frames are computed by interpolation of the key frames. The resulting animation is usually very blocky and if examined closely one can see that the limbs are not stitched together. This kind of animation was used much in the early 3D games such as Tomb Raider^{TM} by CORE Design Inc. One easy approach to extend rigid body animation is to use stitched vertices that connect two limbs. This was used in for example Die by the Sword^{TM} by Treyarch LLC, where the characters used rigid body animation except for the faces between the limbs that were stitched together.

A method that was feasible as soon as computers had more memory, was mesh animation. Mesh animation is in fact rigid body animation but the

whole character is one continuous mesh. To animate the arms the whole mesh has to be updated. It is based on the stop motion animation technique of the animated clay movies. Examples of games that use this technique are Ground Control^{TM} by Massive Entertainment AB, and Quake^{TM} and Quake 2^{TM} by Id Software Inc.

Lately a mix of mesh animation and rigid body animation is being used in computer games. The character is built of a single continuous mesh but is not stop motion animated. Bones transform the vertices in the mesh. The mesh acts as a skin on a skeleton of bones, thus the method is called skinning. An example of a game that uses this technique is Half-Life^{TM} by Valve Software Inc. Most new games use this technique. It is memory cheap and gives great visuals, but costs more than mesh animation and rigid body animation in terms of processing.

1.2 Kinematics

Kinematics describes how objects move in space. There are two kinds of kinematics, inverse and forward. Forward kinematics describes how to move an object in some direction. Inverse kinematics on the other hand, tells where the object is supposed to be, and the motion is derived from where the object is at the moment. Most 3D editors today work using a mix of forward and inverse kinematics, and can be told to produce forward kinematics key frames.

Animations in 3D video games are mostly strictly forward kinematics. Games that use inverse kinematics are very few, because of the complexity involved in the computations and the loss of control. Inverse kinematics is often used in modelling tools, relieving the animator from having to use forward kinematics for long linkages. When the linkage is too long the control is lost though since the inversely generated motion is jerky or inexact. The unwanted motion that the inverse kinematics generates, has to be constrained, by parameters set by the animator, on the individual linkages. A constraining parameter that can be set, is for example the number of degrees of freedom, and the angular limitations on the degrees of freedom. Inverse kinematics is used for all kinds of linked structures that need to be animated, not only humans, since it is a handy and fast technique, once one has learnt how to control it. I had to ponder for quite a while to remember any game that had used inverse kinematics. Actually Red Orb Entertainment Inc used inverse kinematics in real-time in Prince of Persia 3^{TM}, using a system called Motivate^{TM}. Motivate^{TM} is a system developed by The Motion Factory Inc and the system supposedly has intelligent actors. An intelligent actor is run

by a high level system that feeds the actor with moves it needs to perform. One part of the system uses inverse kinematics to control the motion in real-time.

Industrial Light and Magic Inc, the world famous special effects studio, ILM for short, made the special effects for the motion picture Jurassic Park^{TM}. Steven Spielberg, the famous motion picture director, who directed the hit movie Jurassic Park^{TM}, immediately wanted all of the dinosaurs to be animated by computers, when he was shown how good they looked. ILM was faced with having to re-school all of their stop motion animators, who were used to manipulate animatronics beasts covered with foam latex skin. Animatronics beast are elaborate remotely controlled models. What ILM did was brilliantly simple. Instead of re-schooling all of the stop motion animators to use the computer animation tools, they told the stop motion animators to manipulate the animatronics skeletons, while a computer sampled the skeletons. This way the animatronics skeleton generated key frames that were then interpolated by the computer. The effect of this was very realistic animation from the stop motion animatronics skeleton, and smoothly interpolated values by the computer. More of these historical details can be found in [59]. Sampling of motion this way is called motion capture and is used in many games that need motions from the real world. From reading Jeff Lander's article on motion capture [27] I know of House of Moves Inc, a studio in the US that is often contracted to generate the moves needed by game developers. Using professional acrobats or athletes the animation is of very high quality. As told by Klas Nilsson at the computer science department at Lund Institute of Technology, "Snow White and The Seven Dwarfs" used real dancers as motion capture data for the large dance numbers, featured in the cartoon.

1.3 Dynamics

Dynamics is how objects interact with each other using forces. Forward dynamics applies a force on an object and the output is a motion, while inverse dynamics is a desired motion and the output is the needed force. An example of a computer game that uses forward dynamics is Trespasser^{TM}. The player can push, stack and manipulate objects in the world based on Newtonian dynamics, in this game. Almost every game uses some kind of dynamics algorithms. Frequently used dynamics are collision detection, repelling and impulse force interactions. Mostly simulated objects are spheres or boxes because of their simplicity that translates into fast and robust algorithms. A game's primary objective is to have a believable and enjoyable environment.

If that means changing some parameters driving it far from earthly physics or Newtonian laws, then that is what it takes. In fighting games the physics is violent and instantaneous. Players want the opponent to drop down dramatically and explosively when they throw them the final blow. Real dynamics are not very interesting in this setting, just as in action movies where the hero does a fly-kick and stays in the air for several seconds. What is often boring about these games is that the animation looks the same every time a certain event occurs, which becomes repetitive, just as this sentence. Dynamics could be added to the animations by mixing another animation to it that is automatically and dynamically animated. This is the method I have chosen to implement.

Before closing this section I would like to describe the use of inverse dynamics. There is a reason to why inverse dynamics is not needed in games just now. Inverse dynamics decides which angular velocities and accelerations an object must have to be able to create a force with its end effector. An end effector is a term from robotics, and it is the last part of the robot that has to interact with the world. For example in a human arm the hand would be the end effector, if we were not to elbow someone that is. Just like inverse dynamics there is a problem with multiple links, that need constraining parameters, so that they do not break. This is mainly an issue in industrial design and simulation of robots. Robots that have to wash glass windows need some kind of regulation, of how much pressure they can exercise on the window so that they do not break it. Actually that is not even enough since more often the glass window will have irregularities in the surface, or the robot is not calibrated perfectly. What is then added is a haptic interface or a sense interface just as we have in our finger tops. This interface measures the pressure against the glass and adjusts the robot arm accordingly. Since games do not need to do high precision simulations of force goals, there is no interest in the area yet. The last part on haptic interfaces was inspired by ongoing research done by Anders Robertsson at the department of automatic control at Lund Institute of Technology. He helped me out when I had questions about robot dynamics algorithms.

1.4 Problem

The problem is to add physically accurate dynamical motions to a character. What kind of motion can we generate dynamically? How do we add it to the recorded motion? What kind of parameters can be adjusted in run-time and in build-time by an animator to gain control of the resulting animation?

1.5 Approach

Formulating the problem in general is quite easy but to find a more specific problem to solve is harder. When writing this document what hit me was that there is no good distinction between the problem and the solution. Since I didn't know the problem more specifically I had to find a solution, which is the research problem in general, pointed out to me by my supervisor Mathias Haage at the computer science department at Lund Institute of Technology. I did not have a magic wand unfortunately. Looking for a solution platform I knew that if the solution was to be accepted it had to be easy to use for animators. Mostly inventions that are quickly accepted by myself are those that build upon some widely used standard, so I started looking at different character modelling standards. A character built in a modelling program like 3D Studio MAX^{TM} constitutes of a skeleton of bones that many times look just like the real bones in a human being. The problem would then be to consider the dynamics in a hierarchy of interconnected bones. A hierarchy of bones resembles an interconnected chain of rigid bodies. Due to extensive robotics research it is now a solvable problem, to simulate forces acting onto such a hierarchy in real-time. I found a thorough explanation in a PhD dissertation by Brian Mirtich [42] of Featherstone's algorithm, originally in Roy Featherstone's article from 1983 [12], for the dynamics of articulated bodies and of how to solve for tree structures. Driving a linked structure with forces is comfortable. By approximating the limbs as pendulums the addition of external forces is done by setting specific external torques. The algorithm spreads the forces evenly to the connected bones. The problem comes down to the control of the dynamically generated motion that is to be mixed with the pre-generated one. One method is not to simulate those bones that are not interesting for a specific motion. Choosing a set of bones that are to be simulated I call generating a situation. On the other hand one wants control of how much the dynamic motion is to be mixed with the pre-generated one. This control is more low level and demands that the user sets some kind of priorities for the bones. More control is necessary in form of limbs that have variable mass and a specific vector for the angular freedom. These parameters and more are needed to be able to control the dynamically generated motion.

1.6 Report Layout

The rest of the report will deal with all of the specific details of an implementation of the system. The necessary theoretical background and algorithms

are explained and analysed. I begin with dynamics, I then go on to computer animation theory and conclude the reasoning with the actual system. A thorough analysis of the system describing the pros and cons of it by testing it on actual virtual actors. Further development and enhancement ideas are discussed.

Chapter 2

Dynamics

Dynamics is the knowledge of how bodies interact with each other. The field of dynamics is a tool for humans to try to understand and predict these behaviours. The behaviour I am interested in, and hopefully you too, is the one of interconnected bodies. Why interconnected bodies? Well, since we have a skeleton in our body, where the bones are connected to each other by tendons and joints, I thought a suitable dynamic model of the skeleton would be the one of a tree of interconnected bodies. This was my presumption. There was a small problem though, as there is always in research, and that was that I had not the slightest idea of where to start. I was quite prepared though, since I had a thorough physics course in my first year at Lund Institute of Technology. I still had the course book [66] in my shelf and started to brush up on my physics, using it, and another book on dynamics [40]. I understood that what I wanted to do was not to be found in these books. More advanced literature was needed. My thesis supervisor at the computer science department at Lund Institute of Technology, Mathias Haage, handed me a book that would prove to be very valuable, namely an introductory book to the research field of robotics by John J. Craig [8]. I read and later reread the first six chapters of this book. During this time I found much interesting literature on the internet by following up on the references in different articles. I started by searching on Gamasutra [13] for articles about dynamics in computer games. There I found several interesting magazine articles by Jeff Lander [32] through [35], which led me to Chris Hecker's magazine articles on dynamics [17] through [20]. From Hecker's magazine articles I found some interesting leads. In his articles I got to know about the two dynamicists, Andrew Witkin and David Baraff, who were working for the Oscar award winning Pixar Studios Inc. They applied their dynamics knowledge on computer-animated characters. Feasting upon their articles [61]-[63] and [3]-[5], they taught me of how to do a simulation

of a rigid body. There was still something missing. Nowhere in these articles was there a treatise of how to simulate the dynamics of interconnected jointed bodies. What I had learnt from the slew of articles on dynamics I had read, was that I could now name my problem, which was multibody or multiple body forward dynamics, as opposed to single body forward dynamics, obviously. Searching for documents with this word on my trusted search engine google, at http://www.google.com, I finally was seeing some light at the end of the tunnel. The first document I found, was a publication on the web [22], on molecular dynamics by a Colombian dynamicist Andrés Jaramillo-Botero. Since the document I found on the web had no title, I contacted Andrés Jaramillo-Botero about it, and I got a swift reply, in which he wondered if the document had been of use to me. The text had been of use to me, since he in the article described different solution classes of how to solve for multibody forward dynamics, and the special problems he had been exposed to. The solution of Jaramillo-Botero was too complex for my needs since I would not solve for closed chains of bodies, like those found in molecular structures. His article introduced me to Roy Featherstone, a robot dynamicist that had studied the problem of chains of multiple rigid bodies in [12], terming them articulated bodies. I had now two big leads, which were Featherstone and articulated body dynamics, which gave me a lot of hits on articles on IEEE and ACM. Fortunately the university of Lund had a license for IEEEs digital library so that I could search and download articles. I became a student member of ACM to get access to the vast number of research papers online. I found out that dynamics in human bodies was a hot topic from reading different articles from these resources on human motion control [26], [36], [38], [43] and [45]. All of these articles mixed three disciplines of engineering, which were computer science, dynamics or physics and automatic control theory. I got feedback on this from the robotics book of Craig [8], where he mentions that the field of robotics is a cross breed of many disciplines. While I was pondering this, I found an interesting doctoral thesis on my computer, which I had neglected since the title had not attracted my attention before. It was the dissertation of Brian Vincent Mirtich [42], where I found chapter four to be of outmost interest. Chapter four treated the multibody forward dynamics method of Featherstone in a tree structure of rigid bodies. It is like Zen monks tell you to look inside yourself, you should look into your own computer for answers, treasures can be found. The whole section about linked rigid body mechanics is very influenced by Brian Mirtich's dissertation [42]. If I was not lucky enough, Mirtich had also published a complete implementation in C++ on the internet for this specific problem. I was very happy. Months of hunting had finally bore fruit.

What I will do in this chapter is to try to flatten the learning curve I had

for understanding the Featherstone solution for multibody dynamics. Starting from point mass dynamics going to single body rigid body dynamics to the grand finale in the Featherstone section using multibody dynamics. The three sections have in common in how they update the velocities and forces acting upon the bodies during time. This is discussed in the section on propagation of dynamic state. The dynamic system developed here was not to be found in the most advanced physics game engines such as MathEngine^{TM} [39] or the Havok Engine^{TM} [16] when I started out a year ago, but now both of them have articulated body dynamics implemented. Bear in mind though, that my implementation is not a complete physics engine, merely a subset of one, that deals with the force distribution of trees of linked rigid bodies. If MathEngine^{TM} had implemented multibody dynamics at the start of my thesis, I would not have blinked to use it, but alas there was no alternative than to roll ones own at the time. It turned out to be a very interesting experience.

2.1 Point Mass Dynamics

Before we plunge into the waters of theory, let us first build a simple dynamics model, without giving special notice to why it is done. In high school dynamics all objects that can interact with forces are treated as particles. A particle in dynamics is a point mass. Particles can be bullets, balls or other such simple objects. Let's start with Newton's Second Law:

$$F = ma \tag{2.1}$$

Breathing some 3D life into this formula is easily done by treating the force F and acceleration a as vectors. The force F is the sum of all the forces incident on the particle which makes it easy to add gravitational pull and wind. To find the particle's next location in space we need to express it with some differentials. The position depends on the velocity that in turn depends on the acceleration of the particle.

$$x(t + dt) = x(t) + v(t)dt \tag{2.2}$$

$$v(t + dt) = v(t) + a(t)dt \tag{2.3}$$

$$a(t + dt) = \frac{F}{m} \tag{2.4}$$

Inserting (2.4) in (2.3) gives:

$$v(t + dt) = v(t) + \frac{F}{m}dt \tag{2.5}$$

Then inserting (2.5) in (2.2) gives:

$$x(t + dt) = x(t) + v(t)dt + \frac{F}{m}dt^2 \tag{2.6}$$

Using this differential form a computer can get the next position simply by evaluating (2.6) and (2.5) exchanging dt for a time step Δt. For clarity:

$$x(t + \Delta t) = x(t) + v(t)\Delta t + \frac{F}{m}\Delta t^2 \tag{2.7}$$

$$v(t + \Delta t) = v(t) + \frac{F}{m}\Delta t \tag{2.8}$$

The integration method used is called Euler's method. Euler's method is the simplest numerical solver for differential equations. It works by taking small steps in the derivative's direction. The step size Δt determines how well the method approximates the derivative. This and other approximation methods will be discussed in the next section on state propagation.

2.2 Propagation of Dynamic State

A simple physics engine is essentially a small loop which, initialises the dynamic state of all dynamic objects, let the objects interact with forces, updates the dynamic state of all objects and lastly resolves penetrations. This is of course oversimplified but holds generally. What then is a dynamic state, and how do you update one? Let us look at the previous section on point mass dynamics and try to describe it, shall we. The dynamics state is the characteristics that change with time for the point mass, which are position *pos* and velocity *vel*. To simplify notation we put these two into a state vector Y which is a column vector of the column vectors *pos* and *vel*

$$Y = \begin{pmatrix} pos \\ vel \end{pmatrix} \tag{2.9}$$

The point mass also has mass and force that acts upon it, but for now we do not set them in the state vector, since mass is constant and force is something that acts on the point mass. Initialising the state vector is trivial, just set it to reflect the point mass characteristics. The state update step, is the interesting one. What are we trying to update? Well position and velocity, using acceleration or force to update the velocity, and using

the previous velocity to update the position, as seen in equations (2.2) and (2.5). This method of updating the state vector is called Euler integration. The system is based on ordinary differential equations or ODE for short, which means that the derivatives that are computed only depend on one independent variable. Writing the Euler method down using state vectors we get

$$Y_{new} = Y_{old} + \Delta t \frac{d}{dt} Y \tag{2.10}$$

Eulers method is an explicit method for solving ODEs, as opposed to an implicit method which runs backwards as

$$Y_{new} = Y_{old} + \Delta t f(Y_{new}) \tag{2.11}$$

Implicit methods are more complicated but are needed when dynamics systems become stiff, which means that the solving method is in a singularity. Instead of converging on a solution a stiff system's solution explodes. Sometimes it helps taking smaller steps in stiff systems, but in some degenerate cases this is not enough, and an implicit solution is necessary. For more information on implicit methods I refer to [3]. We will continue discussing only explicit methods. In (2.10) how is $\frac{d}{dt}Y$ computed? It is a derivative vector of the individual components expressed as

$$\frac{d}{dt} Y = \frac{d}{dt} \begin{pmatrix} pos \\ vel \end{pmatrix} = \begin{pmatrix} vel \\ acc \end{pmatrix} = \begin{pmatrix} vel \\ f/m \end{pmatrix} \tag{2.12}$$

To be able to have greater step sizes when evaluating a new state, Euler's method is not enough. Before we can improve on the Euler method, we need to investigate it. The basis for the Euler method is the Taylor series

$$f(taylor)_n = f_{old} + \frac{\Delta t^1}{1!} \frac{d}{dt} f_{old} + \frac{\Delta t^2}{2!} \frac{d^2}{dt^2} f_{old} + \ldots + \frac{\Delta t^n}{n!} \frac{d^n}{dt^n} f_{old} \tag{2.13}$$

As can be seen, Euler's method is the first two terms of the Taylor series. The series tries to approximate the real function at a certain time. Assume the sought function is $f(new)$ then it is a function of $f(taylor)_n$, and the error in approximation called the Lagrangian rest term. The rest term is dominated by the last term $n+1$ in the Taylor series that was not included. How many terms should be included then? A five term approximation is enough, such as the Runge-Kutta order four method which has an error of only $O(\Delta t^5)$. Recently I visited the home page of MathEngine^{TM} [39] and noted that their articulated body solution is based on an implicit solver.

Implicit solvers are always of interest, as they can handle much larger step sizes than explicit methods.

2.3 Particle System Dynamics

If we put several particles together and say that they belong together, so that they interact with each other, we have a particle system. Examples of particle systems are gels and cloth simulations. They are good for describing non-rigid structures, but are rather hard to define. Essentially a particle system is a lump of particles where each particle has mass, position, velocity and force or acceleration. Andrew Witkin has a good survey of how to build a general particle system in [62]. Euler's differential method is actually very simple to apply to a whole system. A loop through the system is done by first computing the forces on each particle and deriving a new acceleration and velocity for it. Then using the time step every particles state is updated using the derived state and the old state. The only problem in the method is how to compute the forces for each particle. This function is called a force law function. Deriving such a function from a certain behaviour is a pure cookbook method. For example gels need to keep a constant volume and thus the force law function should account for this. Even though the method of deriving force law functions is mystical, good results have been achieved by extensive experimentation. Particle system dynamics is then an art form.

In Figure 2.1 a table napkin is dropped onto a table. The picture is from the home page of Hugo Elias [11] where he describes how to build force law functions for cloth, gels and strings. A cloth is a grid of interconnected particles much like a fisherman's net. To have a realistic looking simulation of the cloth each particle depends on 24 of its neighbours. The number of neighbours a particle depends on determines how accurate and how fast the computations will be. Too many neighbours and the cloth will behave stiffly. Each particle is a mass point with springs to its neighbours and the force law function simply computes how these springs drag the particle. Setting the spring stiffness and particle mass to a good value is an empirical problem, depending on how one wants the cloth to behave.

2.4 Rigid Body Dynamics

If an object has an extended mass such as a cylinder or a cube, that object is called a rigid body. Rigid bodies have a centre of mass and a mass distribution. They can tumble and rotate about their own axes. For a great

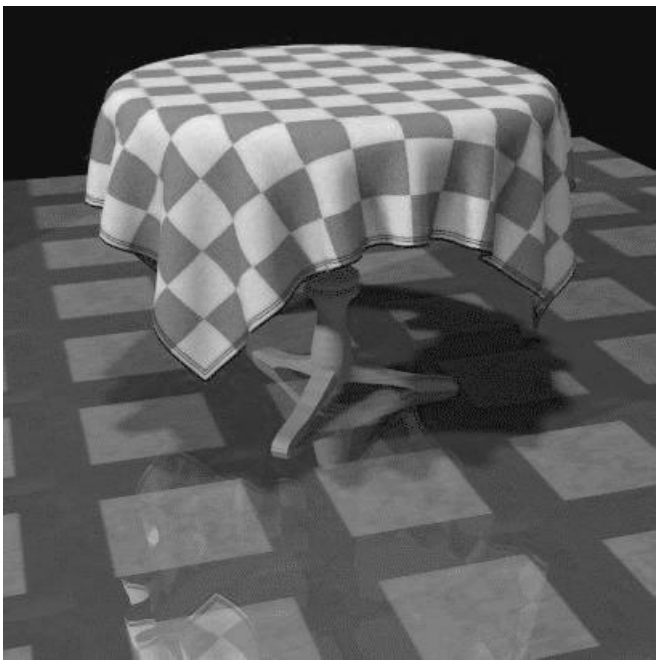

Figure 2.1: A cloth simulation of a napkin on a table. From [11].

survey of rigid body dynamics read [4]. Following the notation of the section on propagation of dynamic state, a rigid body's state becomes

$$Y(t) = \begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} \tag{2.14}$$

The position $x(t)$ vector is straightforward. Orientating the body is done using the rotation matrix $R(t)$. Linear momentum $P(t)$, is the mass of the body times the linear velocity $v(t)$.

$$P(t) = mv(t) \tag{2.15}$$

Angular momentum $L(t)$ is a bit more complicated, and is the inertia tensor $I(t)$ times the angular velocity $\omega(t)$.

$$L(t) = I(t)\omega(t) \tag{2.16}$$

An inertia tensor $I(t)$ is a 3×3 matrix that describes how the mass of a rigid body is extended. It only has to be computed once in the local frame of the rigid body, then orienting it using the rotation R(t).

$$I(t) = R(t)I_{body}R(t)^T \tag{2.17}$$

To be able to use Euler integration we need to express the derivative of the state vector $Y(t)$.

$$\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \tilde{\omega(t)}R(t) \\ F(t) \\ \tau(t) \end{pmatrix} \tag{2.18}$$

Usage of the (˜) sign is explained in the section on spatial algebra but in short it means that we express a cross product of $\omega(t)$ and $R(t)$. The new dynamic state simply becomes

$$Y(t_0 + \Delta t) = Y(t_0) + \Delta t\frac{d}{dt}Y(t_0) \tag{2.19}$$

What remains to be explained is the linear velocity $v(t)$ and the angular velocity $\omega(t)$. They are computed afterwards as

$$v(t) = \frac{P(t)}{m} \tag{2.20}$$

$$\omega(t) = I(t)^{-1}L(t) \tag{2.21}$$

To be perfectly clear let us not forget about the external torque $\tau(t)$ which is the sum of the incident torques. An incident torque is the cross product of the incident force $F_i(t)$ and the distance from the centre of mass $d_i(t)$, which is the moment arm of the incident force.

$$\tau(t) = \sum \tau_i(t) = \sum d_i(t) \times F_i(t) \tag{2.22}$$

Until now the rigid body has not been linked to any other object. This will change in the next section.

2.5 Multibody Forward Dynamics Algorithms

When we have several rigid bodies connected to each other we have a multibody dynamics system. Such a system is for example a robot arm where every limb is a rigid body. Over the years several methods for solving the forward dynamics problem have been proposed. Solving means calculating new angular and linear velocities, accelerations and forces for the links that define the multibody. Just as with point mass and extended mass single

body dynamics, there are two problems. The first problem is to compute the acceleration of the system, and the second is to integrate the next dynamic state. In general the multibody dynamics problem is formulated in the following ODE equation found in [1]

$$\tau = \mathcal{M}(q)\ddot{q} + c(q, \dot{q}) \tag{2.23}$$

On the left hand side, we have the N-element column vector which are the torques applied to the joint actuators. Inspecting the right hand side of the equation, q is the N-element column vector of joint parameters. A joint parameter is either an angular displacement if the joint is revolute, or a distance displacement if the joint is prismatic. Consequently $\dot{q}$ and $\ddot{q}$ are joint parameter velocity and acceleration N-element column vectors. Separating the first product from the first term $\mathcal{M}$ is given, which is a $N \times N$ matrix called the joint-space inertia matrix or the generalized inertia matrix. The second term on the right is the N-element column vector of torques, induced by gravity, centrifugals and Coriolis accelerations. Here N is the number of degrees or equivalently the number of joints, since joints with three degrees of freedom can be described by three joints where two joints have zero length. If the problem is that of forward dynamics, we need to calculate the joint acceleration given the forces and torques induced onto the manipulator, then integrate the new dynamic state of joint parameters and their velocities.

The following exposition of algorithms was found in [22], which makes several references to Walker and Orin 1982 [58] a text given to me by Klas Nilsson at the computer science department at Lund Institute of Technology. The direct method is a $O(n^3)$ method that considers a column at a time of the mass matrix. The kth column of the mass matrix $\mathcal{M}$ is $\mathcal{M}\ddot{\theta}$ for a vector of joint accelerations given by

$$\ddot{\theta}_i = \begin{cases} 0 & for\ i \neq k \\ 1 & for\ i = k \end{cases} \tag{2.24}$$

It follows that the kth column of $\mathcal{M}$ can be computed by applying the Newton-Euler method, to find the forces for this set of joint accelerations. The Newton-Euler method is a recursive inverse dynamics algorithm with $O(n)$ complexity. See [8] pages 196-200 for the details of the Newton-Euler algorithm. Another $O(n^3)$ forward dynamics algorithm is the symmetry based method. The mass matrix is symmetric for serial chains. Thus finding the upper triangular portion of the mass matrix determines the complete $\mathcal{M}$. Yet another $O(n^3)$ method that deserves mentioning is the composite rigid body method. As it is rather elaborate I simply refer to a good description found in Dinesh K. Pai et al [44], where they also point out that it is faster

than $O(n)$ methods for small n. They base their description on the one found in [58].

The second family of algorithms have $O(n^2)$ complexity. An iterative procedure for solving a linear set of equations on the form $Ax = b$ is used by the conjugate gradient method. It assumes the coefficient matrix A to be symmetric definite positive. By using a minimization function an approximate solution is found. See [58] for details. The last $O(n^2)$ algorithm is the triangularised equations method. It is a recursive application of the LDL^T factorisation method using an alternate square factorisation of $\mathcal{M}$.

The Featherstone algorithm is part of a family of algorithms that are called articulated body methods. They all are variations on the same idea and their complexity is $O(n)$.

2.6 Serial Link Forward Dynamics

Before we can describe the Featherstone algorithm we need to set the foundation with a lot of definitions. In this section we will develop an algorithm that calculates the linear and angular velocities, of a serial link of rigid bodies.

Begin by considering the serial link chain in Figure 2.2. The links are numbered 1 to n, where link 1 is attached to a fix base, called link 0. A link i has an inboard joint i and an outboard joint $i + 1$ $(1 \leq i \leq n)$. If every joint has only one degree of freedom we can do a compact description of the links as a vector of joint parameters $q = (q_1, ..., q_1)^T$. A specific parameter q_i is a radian angle about the joint axis if the joint is revolute, while it is a translation along the joint axis if the joint is prismatic. The problem now becomes

Problem 1 *Find the joint parameter accelerations $\ddot{q}$, given the joint parameters q, the joint parameter velocities $\dot{q}$, the external forces acting upon the links and the forces and the torques acting upon the joints.*

2.6.1 Propagation of Velocity and Acceleration

To be able to solve the forward dynamics problem stated in Problem 1 we have to determine the absolute motion of all the links in the chain. The links' linear velocities and angular velocities depend only on q and $\dot{q}$, that is the state of the joints. The linear and angular accelerations of the links depend on q, $\dot{q}$ and $\ddot{q}$ even though $\ddot{q}$ is not yet determined. Let us first define what a body frame for a link i is.

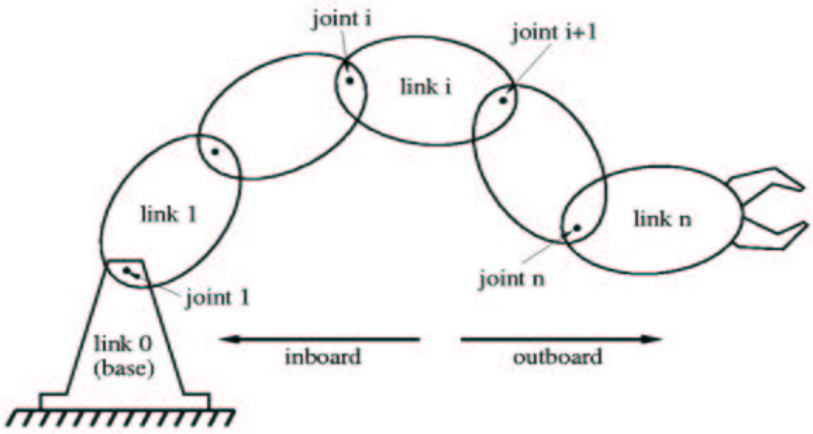

Figure 2.2: A serial link chain. From [42].

Definition 1 *Every link i is expressed in a body frame $\mathcal{F}_i$ with the origin in the links centre of mass. The axes of $\mathcal{F}_i$ are in line with the principal axes of the link. Vectors expressed in the link coordinates are relative to this body frame.*

We can now express Problem 1 using this new definition.

Problem 2 *For each link compute, relative the frame of the link, the linear velocity v_i, angular velocity ω_i, linear acceleration a_i and angular acceleration α_i. Given the link parameters q, link velocities $\dot{q}$ and accelerations $\ddot{q}$.*

The quantities v_i, ω_i, a_i and α_i in Problem 2 describe the motion of frame $\mathcal{F}_i$ relative the inertial frame $\mathcal{O}$. The velocities and accelerations for link i are completely determined by the velocities and accelerations of the link $i-1$ and the motion of the joint i. Since the velocities and accelerations of the base link are 0, which is also the inertial frame, we can start in this one and go from it inductively. From Figure 2.3 we see that u_i is the unit vector in the direction of the joint i axis, perpendicular to the direction axis d_i that is the vector from the joint i origin, to the origin of frame $\mathcal{F}_i$. The vector r_i is the vector from the origin of frame $\mathcal{F}_{i-1}$ to the origin of frame $\mathcal{F}_i$. The motion of link i can be subdivided into two parts. One part that depends on link $i-1$ and one part that depends on the motion of joint i, where the latter is called the relative motion. We need to define the relative quantities first.

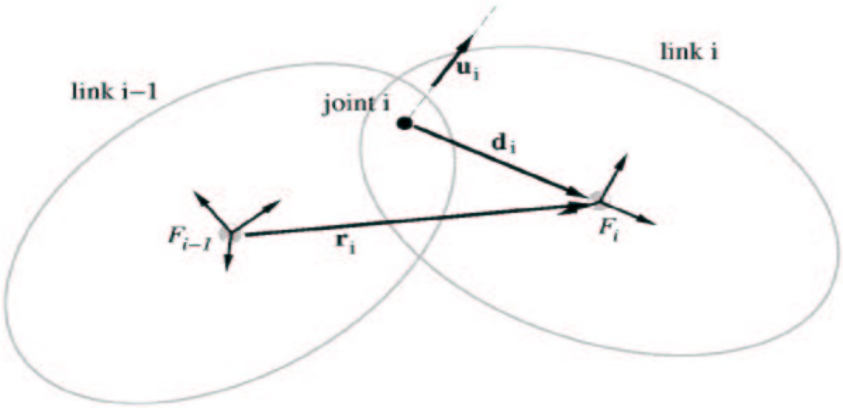

Figure 2.3: A serial link chain. From [42].

Definition 2 *The linear relative velocity* v_{rel} *is the linear velocity of the centre of mass for link* i*. It is the linear velocity link* i *would have if it was in isolation. The relative angular velocity of link* i *is* ω_{rel}*, and it is the angular velocity link* i *would have if it was in isolation.*

Using Definition 2 we can now describe how angular velocity is propagated between links. The angular velocity of link i is the angular velocity of its inboard link $i-1$ and its own relative angular velocity.

$$\omega_i = \omega_{i-1} + \omega_{rel} \tag{2.25}$$

The linear velocity is transformed to other links like the angular velocity except that the previous link's angular velocity is added. You can think of it as if the inboard link slings it forward.

$$\mathrm{v}_i = \mathrm{v}_{i-1} + \omega_{i-1} \times r_i + \mathrm{v}_{rel} \tag{2.26}$$

Since acceleration is the derivative of the velocity, the propagation of acceleration is merely the derivative of the velocity propagation. For angular acceleration we derive the angular velocity propagation (2.25), and get

$$\alpha_i = \alpha_{i-1} + \dot{\omega}_{rel} \tag{2.27}$$

For linear acceleration we derive the linear velocity propagation (2.26), which gives

$$\mathrm{a}_i = \mathrm{a}_{i-1} + \alpha_{i-1} \times r_i + \omega_{i-1} \times \dot{r}_i + \dot{\mathrm{v}}_{rel} \tag{2.28}$$

To loose the derivative of the vector r_i we observe that it is $\mathrm{v}_i - \mathrm{v}_{i-1}$. Thus expressing $\dot{r}_i$ as $\omega_{i-1} \times r_i + \mathrm{v}_{rel}$ and inserting this in 2.28, we get

$$\mathrm{a}_i = \mathrm{a}_{i-1} + \alpha_{i-1} \times r_i + \omega_{i-1} \times (\omega_{i-1} \times r_i) + \omega_{i-1} \times \mathrm{v}_{rel} + \dot{\mathrm{v}}_{rel} \tag{2.29}$$

Now we have expressed a link i's linear and angular velocity and acceleration in respect to the father link or its inboard link $i-1$ and the link's own relative motion. However the relative quantities have not yet been defined. A first step in doing so is to simplify the notation by defining two vectors in the same direction as the joint directional vector u_i. The lengths of these two vectors depend on the velocity and acceleration of the parameters of the joint. Remember that q can be either translation along the joint axis u_i for a prismatic joint i or an angle of rotation about the joint axis u_i for a revolute joint i.

$$\nu_i = \dot{q}_i u_i \tag{2.30}$$
$$\xi_i = \ddot{q}_i u_i \tag{2.31}$$

With these two definitions let us first consider a link with a prismatic joint. Its relative angular velocity must be zero since it does not rotate about the joint, but it has a linear velocity component which is of course in the direction u_i, of the prismatic axis of link i with a velocity of $\dot{q}_i$. It is conveniently denoted using ν_i.

$$\omega_{rel} = 0 \tag{2.32}$$
$$\mathrm{v}_{rel} = \nu_i \tag{2.33}$$

The relative velocities of a revolute joint, that is the angular velocity, is simply the velocity about the joint, while the linear component v_{rel} of link i, is the orthogonal vector to the joint axis velocity vector and the length d_i vector to the centre of mass of link i.

$$\omega_{rel} = \nu_i \tag{2.34}$$
$$\mathrm{v}_{rel} = \nu_i \times d_i \tag{2.35}$$

When I first read this in Mirtich's thesis I was caught in a logical loop since I did not remember that q can either mean translation or angle depending on the joint characteristics. Actually I had to reread Walkers and Orins text

from 1982 [58], where I found on page 143 a list of terms. In Mirtichs thesis the q vector is simply mentioned as the configuration space of the joints, which did not light any bulbs for me. In retrospect it is obvious what he meant. What is now missing is a definition of the relative angular and the relative linear acceleration of a link. The following lemma can be found on page 98 in Mirtich [42], which describes how to derive the quantities in (2.32) through (2.35).

Lemma 1 *For prismatic or revolute joints*

$$\dot{\nu}_i = \xi_i + \omega_{i-1} \times \nu_i \tag{2.36}$$

Revolute joints also have a cross product that needs to be derived.

$$\frac{d}{dt}(\nu_i \times d_i) = \omega_{i-1} \times (\nu_i \times d_i) + \xi_i \times d_i + \nu_i \times (\nu_i \times d_i) \tag{2.37}$$

Proof 1 *Proving Lemma 1 is of interest, and it is a short proof too. Starting by deriving the joint axis velocity $\dot{\nu}_i$, using the rule of derivation of product, then exchanging the first term for the acceleration about the joint axis, we get*

$$\dot{\nu}_i = \ddot{q}_i u_i + \dot{q}_i \dot{u}_i = \xi_i + \dot{q}_i \dot{u}_i \tag{2.38}$$

The second product of the term $\dot{q}_i \dot{u}_i$, which is $\dot{u}_i$, is how the joint axis changes direction. The direction is dependent on the previous link's position, which gives that the direction changes depending on the rotational velocity ω_{i-1} of the previous link. This gives $\dot{u}_i = \omega_{i-1} \times u_i$. Multiplying this with $\dot{q}$, and noting that $\dot{\nu}_i$ is $\dot{q}u_i$ we get the second term of (2.36). Proving (2.37) is done deriving, using the rule of product on it.

$$\frac{d}{dt}(\nu_i \times d_i) = \dot{\nu}_i \times d_i + \nu_i \times \dot{d}_i \tag{2.39}$$

We have already proven $\dot{\nu}_i$ so what is left is left is to prove $\dot{d}_i$. As opposed to the proof of (2.36) $\dot{d}_i$ is dependent on the link i rotation. Since we cannot have the link rotational velocity before we have computed the previous link's rotational velocity, we have to express it in the previous link's rotational velocity.

$$\dot{d}_i = \omega_i \times d_i = (\omega_{i-1} + \nu_i) \times d_i \tag{2.40}$$

Inserting $\dot{d}_i$ along with $\dot{\nu}_i$ into (2.39) we get

$$\frac{d}{dt}(\nu_i \times d_i) = (\xi_i + \omega_{i-1} \times \nu_i) \times d_i + \nu_i \times (\omega_{i-1} + \nu_i) \times d_i \tag{2.41}$$

Expanding the cross products gives

$$\frac{d}{dt}(\nu_i \times d_i) = \xi_i \times d_i + (\omega_{i-1} \times \nu_i) \times d_i + \nu_i \times (\omega_{i-1} \times d_i + \nu_i \times d_i) \tag{2.42}$$

Expanding even more gives

$$\frac{d}{dt}(\nu_i \times d_i) = \xi_i \times d_i + (\omega_{i-1} \times \nu_i) \times d_i + \nu_i \times (\omega_{i-1} \times d_i) + \nu_i \times (\nu_i \times d_i) \tag{2.43}$$

To have a more elegant expression we try to collect the terms to have ω_{i-1} in one term. With the identity below in mind, the expression has to be somewhat manipulated

$$A \times (B \times C) + B \times (C \times A) = -C \times (A \times B) \tag{2.44}$$

The manipulated expression becomes

$$\frac{d}{dt}(\nu_i \times d_i) = \xi_i \times d_i + -d_i \times (\omega_{i-1} \times \nu_i) + \nu_i \times (-d_i \times \omega_{i-1}) + \nu_i \times (\nu_i \times d_i) \tag{2.45}$$

To be able to use the identity we have to invert all occurrences of ω_{i-1} first

$$\frac{d}{dt}(\nu_i \times d_i) = \xi_i \times d_i + -d_i \times (\nu_i \times -\omega_{i-1}) + \nu_i \times (-\omega_{i-1} \times -d_i) + \nu_i \times (\nu_i \times d_i) \tag{2.46}$$

Now if $-d_i$ is A, ν_i is B and $-\omega_{i-1}$ is C we get

$$\frac{d}{dt}(\nu_i \times d_i) = \xi_i \times d_i + -(-\omega_{i-1}) \times (-d_i \times \nu_i) + \nu_i \times (\nu_i \times d_i) \tag{2.47}$$

And finally cleaning the negations up, we get

$$\frac{d}{dt}(\nu_i \times d_i) = \omega_{i-1} \times (\nu_i \times d_i) + \xi_i \times d_i + \nu_i \times (\nu_i \times d_i) \tag{2.48}$$

This concludes the proof of Lemma 1. □

Finally we have enough information to complete the equations (2.25) to (2.29). If joint i is prismatic, the equations become

$$\begin{aligned}
\omega_i &= \omega_{i-1} && (2.49)\\
\mathrm{v}_i &= \mathrm{v}_{i-1} + \omega_{i-1} \times r_i + \nu_i && (2.50)\\
\alpha_i &= \alpha_{i-1} && (2.51)\\
\mathrm{a}_i &= \mathrm{a}_{i-1} + \alpha_{i-1} \times r_i + \xi_i + \omega_{i-1} \times (\omega_{i-1} \times r_i) + 2\omega_{i-1} \times \nu_i && (2.52)
\end{aligned}$$

If joint i is revolute, the equations become

$$\begin{aligned}
\omega_i &= \omega_{i-1} + \nu_i && (2.53)\\
\mathrm{v}_i &= \mathrm{v}_{i-1} + \omega_{i-1} \times r_i + \nu_i \times d_i && (2.54)\\
\alpha_i &= \alpha_{i-1} + \xi_i + \omega_{i-1} \times \nu_i && (2.55)\\
\mathrm{a}_i &= \mathrm{a}_{i-1} + \alpha_{i-1} \times r_i + \xi_i \times d_i + \\
&\quad \omega_{i-1} \times (\omega_{i-1} \times r_i) + 2\omega_{i-1} \times (\nu_i \times d_i) + \nu_i \times (\nu_i \times d_i) && (2.56)
\end{aligned}$$

The equations (2.49) to (2.56), express the quantities relative to the inertial frame of link 0 or frame $\mathcal{O}$. This means all the velocities and accelerations are absolute. Not really necessary to mention but frame $\mathcal{O}$ can be set to any arbitrary frame of choice, although the system will still regard it as inertial. We have enough information to compute actually only the velocity components of every frame in a linkage. By propagating the velocities from the first frame to the last frame all absolute velocities are determined. This is done using equations (2.49) and (2.50) for links with prismatic joints and equations (2.53) and (2.54) for links with revolute joints. See Figure 2.4 for pseudo code. Hold your horses, we have not yet discussed how to eliminate the unknown ξ from the acceleration equations, but we will do so using the Featherstone algorithm, discussed in the Featherstone section. Actually Figure 2.4 describes the first step in the algorithm.

2.6.2 Expressing Quantities in Different Frames

Let us model an example of a one link articulated body, called link 1, attached to an inertial frame $\mathcal{O}$ as in Figure 2.5. Assume joint 1 rotates counter-clockwise with a constant angular velocity $\dot{q}_1$. The equations for linear velocity v_1 and linear acceleration a_1 expressed in the inertial body frame then become

$$\mathrm{v}_1 = \begin{bmatrix} -\dot{q}_1 d_1 \sin(q_1) \\ \dot{q}_1 d_1 \cos(q_1) \end{bmatrix} \qquad (2.57)$$

compSerialLinkVelocities

// Initialize the root link state to zero
$\omega_0, \mathrm{v}_0, \alpha_0, \mathrm{a}_0 \leftarrow 0$

for i=0 to n
  $R \leftarrow$ rotation matrix from $\mathcal{F}_{i-1}$ to $\mathcal{F}_i$
  $r \leftarrow$ radius vector from $\mathcal{F}_{i-1}$ to $\mathcal{F}_i$ (in $\mathcal{F}_i$ coordinates)
  $\omega_i \leftarrow R\omega_{i-1}$
  $\mathrm{v}_i \leftarrow R\mathrm{v}_{i-1} + \omega_i \times r$
  if joint i is prismatic
    $\mathrm{v}_i \leftarrow \mathrm{v}_i + \dot{q}_i u_i$
  else /* joint i is revolute */
    $\omega_i \leftarrow \omega_i + \dot{q}_i u_i$
    $\mathrm{v}_i \leftarrow \mathrm{v}_i + \dot{q}_i(u_i \times d_i)$

Figure 2.4: **compSerialLinkVelocities**

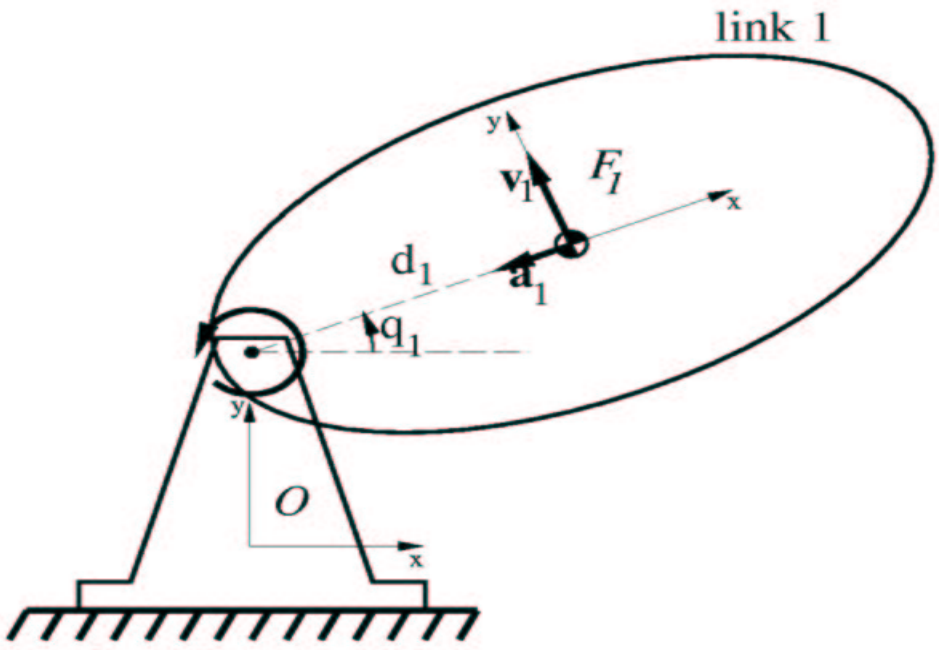

Figure 2.5: Link 1 attached to an inertial body frame $\mathcal{O}$. From [42]

$$\mathrm{a}_1 = \begin{bmatrix} -\dot{q}_1^2 d_1 \cos(q_1) \\ -\dot{q}_1^2 d_1 \sin(q_1) \end{bmatrix} \tag{2.58}$$

Actually these equations become much more manageable if expressed in the body frame $\mathcal{F}_1$

$$\mathrm{v}_1 = \begin{bmatrix} 0 \\ \dot{q}_1 d_1 \end{bmatrix} \tag{2.59}$$

$$\mathrm{a}_1 = \begin{bmatrix} -\dot{q}_1^2 d_1 \\ 0 \end{bmatrix} \tag{2.60}$$

Why are they simpler? Well firstly joint 1 is not rotated in respect to frame $\mathcal{F}_1$ as it is to frame $\mathcal{O}$. Also all quantities are constant in the two latter matrices. Since these quantities do not depend on an inboard link nor an outboard link, it is a trivial articulated body, a term defined in the Featherstone section, which means that the link is regarded in isolation. Imagine we add one more link with an attached body frame $\mathcal{F}_2$. To be able to use equations (2.49) to (2.56) on quantities expressed in frame $\mathcal{F}_2$, we would need to transform them from frame $\mathcal{F}_1$ first. This only requires a multiplication by a transformation matrix. We will develop matrices that do this transformations for vectors of the 6th dimension, in the spatial algebra section that follows.

2.7 Spatial Algebra

The spatial notation developed in this section has been described by several dynamicists, such as Featherstone 1983 [12] and Rodriguez 1991 [49].

Spatial notation simplifies the description of the dynamics relations in three dimensions. The operators make it easier to identify complicated mathematical relations in the mass matrix, mainly because it expresses the relations with a fewer number of symbols. A spatial vector is six dimensional and substitutes two vectors in the third dimension. The spatial velocity vector for example, describes both the linear velocity and the angular velocity in three dimensions respectively of a rigid body. A spatial force applied to a rigid body describes both the linear three dimensional force and the angular force or torque in three dimensions. Spatial vectors are denoted with a caret ($\hat{}$) following the notation of Featherstone.

2.7.1 Transformation of Velocity and Force

We will develop a way to transform velocity, acceleration and force from one body frame into another using spatial algebra. Firstly we need to define what we mean by spatial velocity and acceleration.

Definition 3 *For a rigid body that moves in space with a frame $\mathcal{F}$ attached to it, let* v *be the linear velocity, ω the angular velocity,* a *be the linear acceleration and α be the angular acceleration. These quantities are expressed in the frame $\mathcal{F}$ relative to an inertial frame $\mathcal{O}$, which means that they are absolute quantities. Expressing these four vectors in a spatial notation gives a spatial velocity* $\hat{\mathrm{v}}$ *and a spatial acceleration* $\hat{\mathrm{a}}$*.*

$$\hat{\mathrm{v}} = \begin{bmatrix} \omega \\ \mathrm{v} \end{bmatrix} \tag{2.61}$$

$$\hat{\mathrm{a}} = \begin{bmatrix} \alpha \\ \mathrm{a} \end{bmatrix} \tag{2.62}$$

In animation, rotation matrices are used to rotate vectors in one frame into another. Now we have both acceleration and velocity defined spatially. How do we go about relating the spatial quantities in a frame $\mathcal{F}$ to a frame $\mathcal{G}$? First assume these two frames are connected to a rigid body, where $\mathcal{G}$ is translated relative $\mathcal{F}$. Let r be the vector that translates $\mathcal{G}$ from the origin of $\mathcal{F}$. If $\mathrm{v}_{\mathcal{F}}$ and $\mathrm{v}_{\mathcal{G}}$ are the linear velocities of the frames, not the spatial ones, and ω the shared angular velocity. Then $\mathrm{v}_{\mathcal{G}}$ becomes

$$\mathrm{v}_{\mathcal{G}} = \mathrm{v}_{\mathcal{F}} + \omega \times r \tag{2.63}$$

This can be written in spatial form using a trick which makes a matrix express a cross product and is denoted using a tilde (˜). Such a matrix is composed as follows. Given two vectors a and b in three dimensional space, the cross product between the two is defined as

$$a \times b = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix} \tag{2.64}$$

It can be written also as

$$a \times b = \tilde{a} b \tag{2.65}$$

Where $\tilde{a}$ is the skew-symmetric 3×3 matrix

$$\tilde{a} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \tag{2.66}$$

Since (2.63) has both a linear component and an angular component it is natural to express it using spatial quantities.

$$\hat{\mathrm{v}}_{\mathcal{G}} = \begin{bmatrix} \omega \\ \mathrm{v}_{\mathcal{G}} \end{bmatrix} = \begin{bmatrix} \omega \\ \mathrm{v}_{\mathcal{F}} + \omega \times r \end{bmatrix} = \begin{bmatrix} I & 0 \\ -\tilde{r} & I \end{bmatrix} \begin{bmatrix} \omega \\ \mathrm{v}_{\mathcal{F}} \end{bmatrix} \tag{2.67}$$

The matrix on the right side with $\tilde{r}$ in it is called a spatial matrix and is a 6×6 matrix. We have only considered if $\mathcal{G}$ was not rotated relative $\mathcal{F}$. To do this we have to transform or rotate the vectors in $\mathcal{F}$ to vectors in $\mathcal{G}$ using the rotation matrix R. The matrix R rotates vectors in frame $\mathcal{F}$ into vectors in frame $\mathcal{G}$. Doing this for spatial vectors we need to express the rotation matrix in a 6×6 spatial matrix.

$$\begin{bmatrix} R & 0 \\ 0 & R \end{bmatrix} \tag{2.68}$$

We can now incorporate this rotational spatial matrix into a new definition.

Definition 4 *Let $\mathcal{F}$ and $\mathcal{G}$ be two frames, r the vector that translates the origin of frame $\mathcal{G}$ from frame $\mathcal{F}$ and R be the 3×3 rotational matrix that transforms three dimensional vectors from $\mathcal{F}$ to $\mathcal{G}$. Then the spatial transformation matrix that transforms from $\mathcal{F}$ to $\mathcal{G}$ is defined as*

$${}_{\mathcal{G}}\hat{X}_{\mathcal{F}} = \begin{bmatrix} I & 0 \\ -\tilde{r} & I \end{bmatrix} \begin{bmatrix} R & 0 \\ 0 & R \end{bmatrix} = \begin{bmatrix} R & 0 \\ -\tilde{r}R & R \end{bmatrix} \tag{2.69}$$

Using the spatial transformation matrix, the spatial velocity $\hat{\mathrm{v}}_{\mathcal{G}}$ can be written as

$$\hat{\mathrm{v}}_{\mathcal{G}} = {}_{\mathcal{G}}\hat{X}_{\mathcal{F}} \hat{\mathrm{v}}_{\mathcal{F}} \tag{2.70}$$

The spatial transformation matrix ${}_{\mathcal{G}}\hat{X}_{\mathcal{F}}$ can be seen as the analog to the homogenous transformation matrix in three dimensions that translates and rotates vectors expressed in one frame into vectors expressed in another frame. The spatial matrix ${}_{\mathcal{G}}\hat{X}_{\mathcal{F}}$ transforms spatial vectors from frame $\mathcal{F}$ to frame $\mathcal{G}$. Can this spatial matrix also be used to propagate spatial force vectors? Well yes, the discussion following the next definition clarifies this.

Definition 5 *Given a rigid body with a frame $\mathcal{F}$, let the total external forces on the body be given by a linear force* f *that acts on the body with a vector through the origin of $\mathcal{F}$, and a torque τ. The spatial force is then*

$$\hat{\mathrm{f}} = \begin{bmatrix} \mathrm{f} \\ \tau \end{bmatrix} \tag{2.71}$$

Let $\mathcal{G}$ be the frame we have been using that is related to $\mathcal{F}$ using a translation vector r and a rotational matrix R. The force that acts through the origin of $\mathcal{F}$ is translated to act onto the origin of $\mathcal{G}$. If a correction torque of $-r \times \mathrm{f}$ is added, then the torque τ needs no adjustment according to [40]. The forces in $\mathcal{F}$ are now expressed in $\mathcal{G}$ by

$$\hat{\mathrm{f}}_{\mathcal{G}} = \begin{bmatrix} \mathrm{f} \\ \tau_{\mathcal{G}} \end{bmatrix} = \begin{bmatrix} \mathrm{f} \\ \tau_{\mathcal{F}} - r \times \mathrm{f} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\tilde{r} & 1 \end{bmatrix} \begin{bmatrix} \mathrm{f} \\ \tau_{\mathcal{F}} \end{bmatrix} \tag{2.72}$$

This is analogous to the spatial velocity transformation, thus the same transformation matrix can be used to transform spatial force vectors expressed in frame $\mathcal{F}$ into spatial force vectors expressed in frame $\mathcal{G}$. That is

$$\hat{\mathrm{f}}_{\mathcal{G}} = {}_{\mathcal{G}}\hat{X}_{\mathcal{F}} \hat{\mathrm{f}}_{\mathcal{F}} \tag{2.73}$$

For the sake of completeness let us define the transpose of a spatial vector and the inner product for two spatial vectors.

Definition 6 *If $\hat{x}$ is a spatial vector composed of two three dimensional vectors a and b, written as*

$$\hat{x} = \begin{bmatrix} a \\ b \end{bmatrix} \tag{2.74}$$

then the spatial transpose of $\hat{x}$ is denoted $\hat{x}'$ and is written

$$\hat{x}' = \begin{bmatrix} b^T, a^T \end{bmatrix} \tag{2.75}$$

Definition 7 *The spatial inner product of two spatial vectors $\hat{x}$ and $\hat{y}$ is given by $\hat{x}'\hat{y}$*

2.7.2 Transformation of Acceleration

With the previous definitions we are ready to tackle the problem of describing the transformation of acceleration between two frames. The non spatial notation equations for the prismatic joint i in (2.51) and (2.52) can be written more compactly as

$$\begin{bmatrix} \alpha_i \\ \mathrm{a}_i \end{bmatrix} = \begin{bmatrix} \alpha_{i-1} \\ -r_i \times \alpha_{i-1} + \mathrm{a}_{i-1} \end{bmatrix} + \ddot{q} \begin{bmatrix} 0 \\ u_i \end{bmatrix} + \begin{bmatrix} 0 \\ \omega_{i-1} \times (\omega_{i-1} \times r_i) + 2\omega_{i-1} \times \nu_i \end{bmatrix} \tag{2.76}$$

Doing the same for the non spatial equations for the revolute joint i in (2.55) and (2.56) we get

$$\begin{bmatrix} \alpha_i \\ \mathrm{a}_i \end{bmatrix} = \begin{bmatrix} \alpha_{i-1} \\ -r_i \times \alpha_{i-1} + \mathrm{a}_{i-1} \end{bmatrix} + \ddot{q} \begin{bmatrix} u_i \\ u_i \times d_i \end{bmatrix} + \begin{bmatrix} \omega_{i-1} \times \nu_i \\ \omega_{i-1} \times (\omega_{i-1} \times r_i) + 2\omega_{i-1} \times (\nu_i \times d_i) + \nu_i \times (\nu_i \times d_i) \end{bmatrix} \tag{2.77}$$

We would now want to do the same for acceleration transformations as we did for force and velocity transformation using the spatial transformation matrix. For this we need some more definitions.

Definition 8 *The spatial joint axis of joint i, for a prismatic joint is defined as*

$$\hat{s}_i = \begin{bmatrix} 0 \\ u_i \end{bmatrix} \tag{2.78}$$

and for a revolute joint it is defined as

$$\hat{s}_i = \begin{bmatrix} u_i \\ u_i \times d_i \end{bmatrix} \tag{2.79}$$

Definition 9 *The spatial Coriolis force for a link i, given a prismatic inboard joint, is the spatial vector*

$$\hat{c}_i = \begin{bmatrix} 0 \\ \omega_{i-1} \times (\omega_{i-1} \times r_i) + 2\omega_{i-1} \times \nu_i \end{bmatrix} \tag{2.80}$$

and given a revolute inboard joint it is

$$\hat{c}_i = \begin{bmatrix} \omega_{i-1} \times \nu_i \\ \omega_{i-1} \times (\omega_{i-1} \times r_i) + 2\omega_{i-1} \times (\nu_i \times d_i) + \nu_i \times (\nu_i \times d_i) \end{bmatrix} \tag{2.81}$$

Now let us analyse the equations for spatial acceleration propagation (2.76) and (2.77). As can be noted the first vector in each equation resembles the equations for velocity and force transformation in (2.63) and (2.72). This vector is the transformed acceleration from frame $\mathcal{F}_{i-1}$ to frame $\mathcal{F}_i$. Thus we can use the spatial transformation matrix ${}_{\mathcal{F}_i}\hat{X}_{\mathcal{F}_{i-1}}$ to transform this acceleration vector. Further we can see that the spatial joint axis is the second vector multiplied by $\ddot{q}$ and that the last vector is the spatial Coriolis force. The general acceleration propagation then becomes

$$\hat{\mathrm{a}}_i = {}_{\mathcal{F}_i}\hat{X}_{\mathcal{F}_{i-1}}\hat{\mathrm{a}}_{i-1} + \ddot{q}\hat{s}_i + \hat{c}_i \tag{2.82}$$

Now we have developed the spatial notation for acceleration propagation. Is it not neat?

2.8 The Featherstone Algorithm

The Featherstone algorithm is a structurally recursive algorithm based on the articulated body methodology. Featherstone's algorithm is called a generalized coordinate approach, which means that it has as many state variables as there are degrees of freedom in the system. This means that we can never have an invalid state configuration. The opposite method is the maximal coordinate approach. It is in a family of multiplier methods, which means it has more state variables than degrees of freedom. Multiplier methods are more complex and can be applied to looped chains, which Featherstone's cannot. Firstly we need to define what an articulated body is.

Definition 10 *An articulated body is a subset of a chain of links in isolation, that starts in a link called the handle and stops in the last link of the chain. The last link in the chain has number n, thus the articulated body consist of the links from the handle i to the last link, or links i...n. The trivial articulated body consists of the last link in isolation where the handle $i = n$.*

The Featherstone algorithm begins with treating the trivial articulated body by adding inboard links one by one. The idea is to relate the spatial acceleration $\hat{\mathrm{a}}_i$ for the handle in the articulated body i, to the spatial force applied at its inboard joint. The outboard links for the handle i will have an effect on this relation.

Definition 11 *The spatial acceleration of link i is denoted $\hat{\mathrm{a}}_i$, the spatial force that its inboard link influences it with is $\hat{\mathrm{f}}_i^I$, the spatial force that its outboard link influences it with is $\hat{\mathrm{f}}_i^O$. These spatial vectors are expressed in the frame of link i which is $\mathcal{F}_i$.*

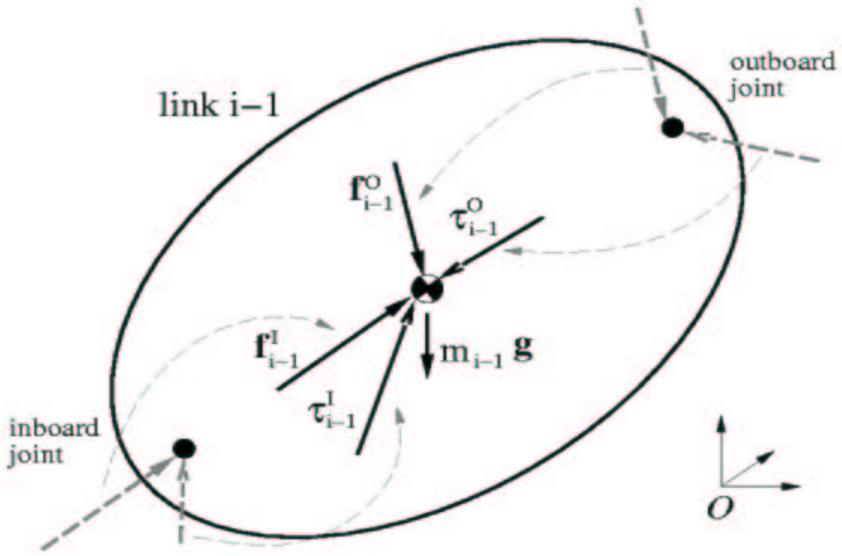

Figure 2.6: A link influenced by its inboard and outboard links. From [42].

An interesting thing to note is that the spatial forces from the inboard and outboard links that influence a link at its inboard and outboard joints are the same forces simply translated to the origin of the frame $\mathcal{F}_i$. This is illustrated in Figure (2.6).

Theorem 1 *Consider the articulated body of serial linkage that has link i as a handle ($1 \leq i \leq n$). There exists a spatial matrix $\hat{I}_i^A$ and a spatial vector $\hat{Z}_i^A$ such that*

$$\hat{\mathrm{f}}_i^A = \hat{I}_i^A \hat{\mathrm{a}}_i + \hat{Z}_i^A \tag{2.83}$$

$\hat{I}_i^A$ is called the spatial articulated inertia of link i and is independent of the joint velocities and accelerations. $\hat{Z}_i^A$ is called the spatial articulated zero-acceleration force of link i, and is independent of the joint accelerations.

The zero-acceleration force $\hat{Z}_i^A$ is termed so since it is the force that the inboard joint must act upon link i so that it does not accelerate. Featherstone calls these forces bias forces. I refer to Mirtich [42] pages 106-111 for the proof of Theorem 1.

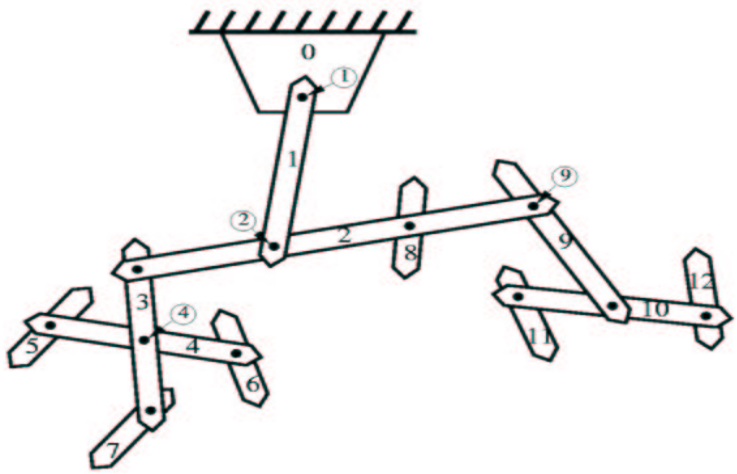

Figure 2.7: The numbering of the links and joints. From [42].

2.8.1 Tree Topologies

Instead of writing the algorithm down for serial links, I would want to extend it to tree-like linkages first. In a tree topology a link has several outboard links, as apposed to a serial chain where links only have one outboard link. Links have only one inboard link though, called link h or the father link. The links are numbered depth-first, starting on the inertial link that is numbered 0 as seen in Figure 2.7.

The algorithm for tree topologies is an extension of the one for links in a series. Instead of traversing the tree using the links we now traverse it using the joints. In a serial link chain it does not matter but in a tree the links can have several children. A joint is only coupled to one inboard link and one outboard link. This means that the velocity propagation equations (2.51), (2.52), (2.55) and (2.56) can be kept as they are by exchanging the inboard index link from $i-1$ to h. The velocity of a link must be computed before it is used, which means that we have to go through the tree over the joints, propagating the velocity from the joint's inboard link to the joint's outboard link.

The first step in the algorithm is to determine all the absolute velocities of all the links in the tree, which it determines using the absolute velocity of the link itself, its parent link and its outboard joint. Using the joint-centric approach previously discussed this step is the same as the one for serial links, with minor modifications.

Propagating acceleration through the tree is analogous to the case for

serial links. If link h is the father link to link i then the acceleration of link i is computed from the positions, velocities and acceleration from link h and joint i. The derivation of the propagation of acceleration is the same for the one in the serial link case, thus the acceleration of link i is

$$\hat{\mathrm{a}}_i = {}_i\hat{X}_h\hat{\mathrm{a}}_h + \ddot{q}_i\hat{\mathrm{s}}_i + \hat{c}_i \tag{2.84}$$

The next step in the algorithm is to calculate the spatial articulated inertias and the spatial articulated zero-acceleration forces for all the links. In a serial linkage this is done iterating from tip to base. In a tree topology we must first compute the leaf values. The inner nodes are determined by combining the children, that is depth first.

Definition 12 *For a linked tree, consider the tree rooted in link i in isolation. This sub tree is an articulated body of the original tree. Link i is the handle of the articulated body. If the link is a leaf in the tree it is trivial, and is simply the isolated rigid body of link i.*

Armed with this definition that resembles Definition 10 we can now formulate the tree link counterpart of Theorem 1.

Theorem 2 *Consider the articulated body for a tree structure that has link h as handle ($1 \leq h \leq n$). There exists a spatial matrix $\hat{I}_h^A$ and a spatial vector $\hat{Z}_h^A$ such that*

$$\hat{\mathrm{f}}_h^I = \hat{I}_h^A\hat{\mathrm{a}}_h + \hat{Z}_h^A \tag{2.85}$$

$\hat{I}_h^A$ is independent of the joint velocities and accelerations. $\hat{Z}_h^A$ is independent of the joint accelerations.

For the proof of Theorem 2 see Mirtich [42] pages 116-118.

2.8.2 Forward Dynamics Algorithm

Finally we can formulate the solution to the forward dynamics problem for a tree structure of linked rigid bodies. The forward dynamics algorithm for tree linkages consists of four steps.

1. Iterating over all the joints in increasing order of index, compute the absolute spatial velocities of all the links. See Figure 2.8.

2. Initialising every link's articulated inertia and articulated zero-acceleration force for the isolated parts, compute the Coriolis vector for each link. See Figure 2.9.

compTreeLinkVelocities

// Initialize the root link state to zero
$\omega_0, \mathrm{v}_0, \alpha_0, \mathrm{a}_0 \leftarrow 0$

for i=1 to n
  $h \leftarrow$ index of link inboard to joint i
  $R \leftarrow$ rotation matrix from $\mathcal{F}_h$ to $\mathcal{F}_i$
  $r \leftarrow$ radius vector from $\mathcal{F}_h$ to $\mathcal{F}_i$ (in $\mathcal{F}_i$ coordinates)
  $\omega_i \leftarrow R\omega_h$
  $\mathrm{v}_i \leftarrow R\mathrm{v}_h + \omega_i \times r$
  if joint i is prismatic
    $\mathrm{v}_i \leftarrow \mathrm{v}_i + \dot{q}_i u_i$
  else /* joint i is revolute */
    $\omega_i \leftarrow \omega_i + \dot{q}_i u_i$
    $\mathrm{v}_i \leftarrow \mathrm{v}_i + \dot{q}_i (u_i \times d_i)$

Figure 2.8: **compTreeLinkVelocities**

3. Iterating over the joints in decreasing order of index, back propagate the articulated inertia and zero-acceleration force from the outboard link to the inboard link of the joint. See Figure 2.10.

4. Iterating over the joints in increasing order of index, compute the acceleration of every joint and the spatial acceleration of the outboard link of the joint. See Figure 2.11.

initTreeLinks

for i=1 to n
$\hat{Z}_i^A \leftarrow \begin{bmatrix} -m_i g \\ \omega_i \times I_i \omega_i \end{bmatrix}$
$\hat{I}_i^A \leftarrow \begin{bmatrix} 0 & M_i \\ I_i & 0 \end{bmatrix}$
$h \leftarrow$ index of link inboard to joint i
if joint i is prismatic
$\hat{c}_i \leftarrow \begin{bmatrix} 0 \\ \omega_h \times (\omega_h \times r_i) + 2\omega_h \times \nu_i \end{bmatrix}$
else /* joint i is revolute */
$\hat{c}_i \leftarrow \begin{bmatrix} \omega_h \times \nu_i \\ \omega_h \times (\omega_h \times r_i) + 2\omega_h \times (\nu_i \times d_i) + \nu_i \times (\nu_i \times d_i) \end{bmatrix}$

Figure 2.9: **initTreeLinks**

backPropagate

for i=n downto 2
$h \leftarrow$ index of link inboard to joint i
$\hat{I}_h^A \leftarrow \hat{I}_h^A + {}_h\hat{X}_i \left[\hat{I}_i^A - \frac{\hat{I}_i^A \hat{s}_i \hat{s}_i' \hat{I}_i^A}{\hat{s}_i' \hat{I}_i^A \hat{s}_i} \right] {}_i\hat{X}_h$
$\hat{Z}_h^A \leftarrow \hat{Z}_h^A + {}_h\hat{X}_i \left[\hat{Z}_i^A + \hat{I}_i^A \hat{c}_i + \frac{\hat{I}_i^A \hat{s}_i \left[Q_i - \hat{s}_i' (\hat{Z}_i^A + \hat{I}_i^A \hat{c}_i) \right]}{\hat{s}_i' \hat{I}_i^A \hat{s}_i} \right]$

Figure 2.10: **backPropagate**

accPropagate

$\hat{\mathrm{a}}_0 \leftarrow \hat{0}$
for i=1 to n
$h \leftarrow$ index of link inboard to joint i
$\ddot{q}_i = \frac{Q_i - \hat{s}_i' \hat{I}_i^A \, {}_i\hat{X}_h \hat{\mathrm{a}}_h - \hat{s}_i' (\hat{Z}_i^A + \hat{I}_i^A \hat{c}_i)}{\hat{s}_i' \hat{I}_i^A \hat{s}_i}$
$\hat{\mathrm{a}}_i = {}_i\hat{X}_h \hat{\mathrm{a}}_h + \hat{c}_i + \ddot{q} \hat{s}_i$

Figure 2.11: **accPropagate**

Chapter 3

Animation

The part of computer animation that I have been interested in is the one of human beings. Since to animate means to give life or to make alive, what is then better than human animation? Some history of animation was treated in the introductory chapter. I described how a human could be built from separate parts just as with robots. The method is a branch of rigid body animation and can be called hierarchical rigid body animation or articulated figure animation. It is called hierarchical since the different parts have a relative order. This method has been replaced lately by a deformed body animation method called skinning. Hierarchical rigid body animation is the basic building block in skinning and is crucial for the understanding of the latter method.

The most important part of this chapter is the one on parametric animation, which is then used in the parametric dynamics chapter. To be able to understand what it means and what it does we need to know the most basic character animation techniques first.

3.1 Rigid Body Animation

Let us define the animation of a rigid body to be composed of two transforms or a hierarchy of transforms. A transform is a way to position and orientate an object. It is done most easily using a matrix that contains both of the operators on an object. If we define that matrix concatenation is done left to right, the transforms that are done first are the rightmost ones. Thus we get a right to left concatenation order of the operators.

$$positionMatrix * rotationMatrix \tag{3.1}$$

If we use column-major ordering of the matrices we get two different

matrices for the operators, where notation follows the one in OpenGL^{TM} found in the "Red Book" [65].

$$positionMatrix = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

The rotation operator is usually expressed as a rotation with an angle θ about a certain axis. This was found in Graphics Gems I page 466 [14].

$$rotationMatrix = \begin{bmatrix} tx^2 + c & txy + sz & txz - sy & 0 \\ txy - sz & ty^2 + c & tyz + sx & 0 \\ txz + sy & tyz - sx & tz^2 + c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

Where x, y and z are the components of the unit vector along the axis and where $s = \sin(\theta)$, $c = \cos(\theta)$ and $t = 1 - c$. See Craigs robotics book [8] chapter 2 for the derivation, well actually you will have to derive it yourself in exercise 2.6 in said book.

What we do to get the transform of these two operators is to concatenate them into a single matrix. Such a concatenation is called a frame. The name frame comes from the research field of robotics. We can analyse this transform more deeply, and denote a frame ${}^{\mathrm{A}}_{\mathrm{B}}X$ as one that relates B's coordinates $\mathrm{p_B}$ into A's coordinates $\mathrm{p_A}$ as

$$\mathrm{p_A} = {}^{\mathrm{A}}_{\mathrm{B}}X\mathrm{p_B} \tag{3.4}$$

If we take a closer look at the transform it is a rotation and a translation, but what do they mean? Setting the transform as

$${}^{\mathrm{A}}_{\mathrm{B}}X = {}^{\mathrm{A}}_{\mathrm{B}}\mathrm{T}\ {}^{\mathrm{A}}_{\mathrm{B}}\mathrm{R} \tag{3.5}$$

The translation matrix ${}^{\mathrm{A}}_{\mathrm{B}}\mathrm{T}$ contains the vector offseting the origin of the coordinate system B, expressed in A. Inspecting the rotation matrix ${}^{\mathrm{A}}_{\mathrm{B}}\mathrm{R}$ it is the basis vectors of coordinate system B expressed in A. Some other aspects of rigid body systems are the fast inversions, inspired by Möller and Haines' book [41]. Say we want ${}^{\mathrm{B}}_{\mathrm{A}}X$ instead of ${}^{\mathrm{A}}_{\mathrm{B}}X$, and set them up as

$${}^{\mathrm{B}}_{\mathrm{A}}X = ({}^{\mathrm{A}}_{\mathrm{B}}X)^{-1} \tag{3.6}$$

expanding the last expression

$$({}^{\mathrm{A}}_{\mathrm{B}}X)^{-1} = ({}^{\mathrm{A}}_{\mathrm{B}}\mathrm{T}\ {}^{\mathrm{A}}_{\mathrm{B}}\mathrm{R})^{-1} = {}^{\mathrm{A}}_{\mathrm{B}}\mathrm{R}^{-1}\ {}^{\mathrm{A}}_{\mathrm{B}}\mathrm{T}^{-1} \tag{3.7}$$

Rigid body transforms are thus very fast to invert, since the rotation matrix inversion is a transposition, and a translation matrix inversion is a negation of the translating component.

3.1.1 Parent-Child Hierarchy

In a hierarchical rigid body animation every object has its own local frame which is a transform. Let us denote the transform that relates the local frame i to $i-1$ with ${}^{i-1}_{i}X$. To get an object's world coordinates or to transform it into the world frame one has to concatenate all the frames of its ancestors. You start by defining the frame that relates frame 0 to the inertial frame $\mathcal{O}$ as the root ${}^{\mathcal{O}}_{0}X$ and add children frames to it that in turn have children and so forth. Using our notation this is equivalent to get the transform ${}^{\mathcal{O}}_{i}X$. This tree of transforms is a directed acyclic graph. An example of a hierarchic rigid body character is seen Figure 3.1. To get the transform of a certain target frame that is to transform it into a certain frame, you start in the transform ${}^{\mathcal{O}}_{0}X$ and multiply all the matrices for the path down to the target frame i. This hierarchic structure is called a parent-child hierarchy. Expressed in our notation the world transform ${}^{\mathcal{O}}_{i}X$ is computed as

$$ {}^{\mathcal{O}}_{i}X = {}^{\mathcal{O}}_{0}X \; {}^{0}_{1}X \; \ldots \; {}^{i-1}_{i}X \tag{3.8} $$

It is not very efficient to calculate all of these matrix multiplications every time you need a transform a child relative a specific frame. Instead you can update the whole tree at once, using a depth first traversal from the root of the tree. When doing this traversal, and arrive at a specific child frame you multiply this with the current parent matrix, and for all its children you pass this matrix on as the new parent matrix. The passing of the parent matrix could either be done implicitly, using the stack of the programming language running environment, or it could be done explicitly using a specific matrix stack. The parent matrix needs to be untouched or untransformed, otherwise the other children will get the wrong transform. Now we have enough information to go to the next step, namely bones.

3.2 Skinning Algorithms

One problematic area in character animation is the one on how to represent the skin of a human or an animal. It is not a trivial problem and the solutions range from systems that incorporate anatomical systems, that build animated characters from muscles and fat, to solutions based solely on the geometric characteristics of the hull of the body. Algorithms that try to

Figure 3.1: A hierarchic rigid body character. From [9].

simulate human looks using physics are called deep algorithms, while the ones that use geometric properties are called shallow. This division in deep and shallow also describe the computational requirements where the former is more complex. An example of a deep algorithm is the Layered Elastic Model Animation system or LEMAN for short by Russel Turner [57]. The system is based on the tissue configuration of vertebrate animals. Bones build a skeleton to which muscles are attached, to which a fatty tissue layer is applied, as seen in Figure 3.2. The system is quite complex using particle dynamics systems, for the different layers. Skin will crease and stretch with time, giving very natural looks to the characters, as seen in Figure 3.3.

On the other hand we have the shallow algorithms. Two examples of such algorithms are Pose Space Deformation, PSD for short and Skeleton Subspace Deformation, which is SSD for short. Both models describe how a hull like the human skin, is deformed by a set of bones. Skeleton Subspace Deformation has problems with collapsing vertices when the bones are in certain configurations as described in the next section. Pose Space Deformation tries to correct these problems using a volume preserving metric along with several subspace deformation parameters. The subspace parameters are such that when the mesh is deformed into a certain position these parameters try to correct the mesh, into a non-collapsed state. The parameters are human

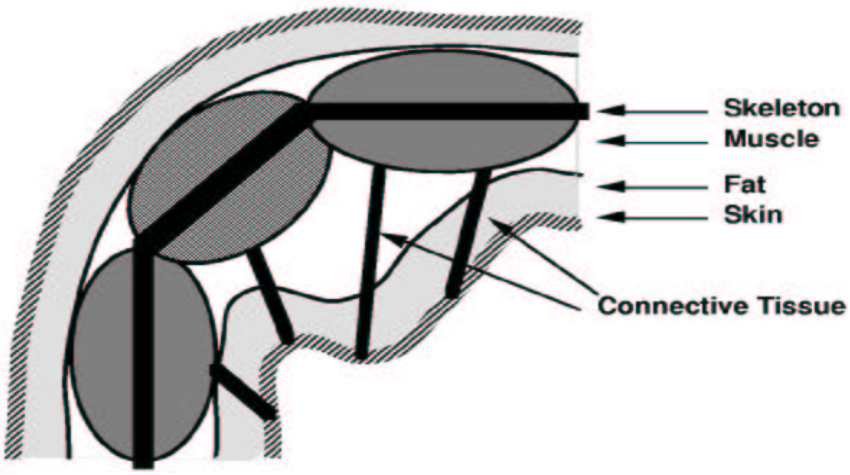

Figure 3.2: The LEMAN layers. From [57].

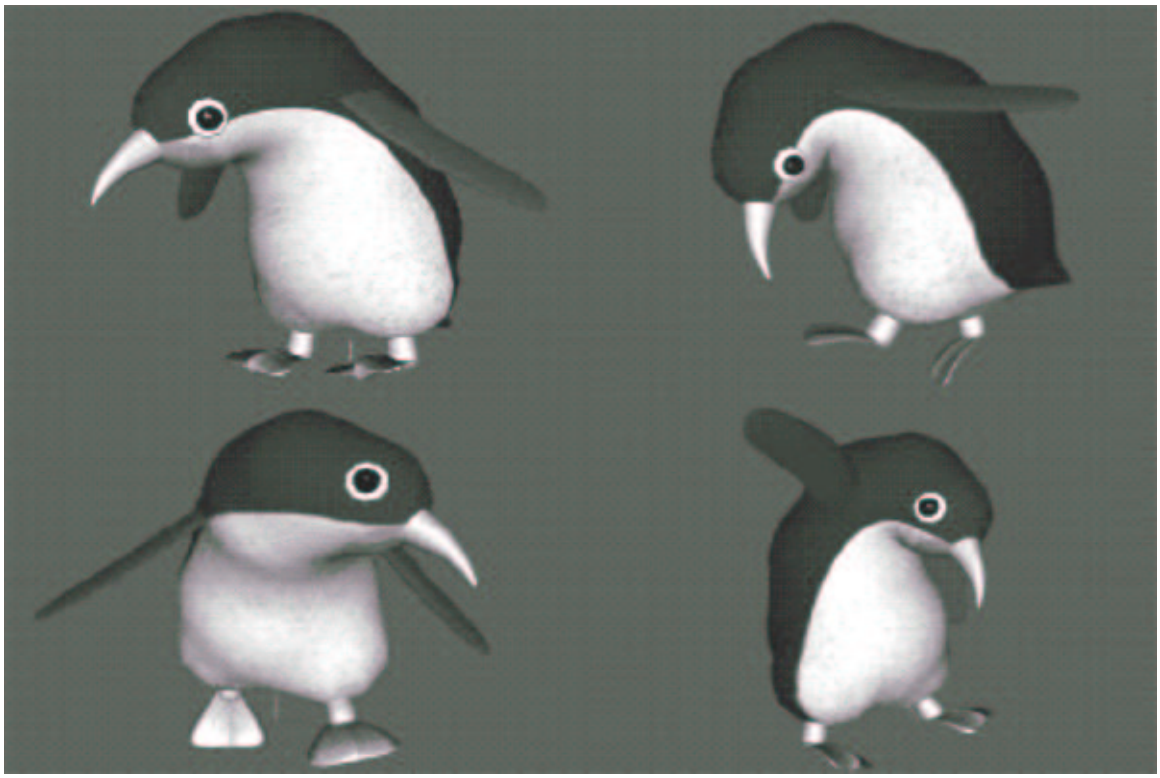

Figure 3.3: A penguin built using the LEMAN system. From [57].

driven, so the animators have to set them by hand. There is no publicly available implementation of PSD, but the authors of the original article [37] report that animators are handling the configurable parameters with ease. I have based my prototype on the Skeleton Subspace Deformation algorithm which is described in the following section.

3.2.1 Skeleton Subspace Deformation or "Skinning"

The inspiration to write about skinning comes from Jeff Lander's article [30], while the actual implementation technique was derived from an example exporter for 3D Studio Max^{TM} by Discreet Inc. If you want computer animated figures that have more of a real skin, hierarchic rigid body animation isn't enough. In 3D Studio MAX^{TM} there has been for quite some time a tool called bones. They facilitate the building of humanoid figures and animals. What are they then? In our body the skeleton works as a supporting structure for the tissue and as levers for the muscles. Bones in 3D Studio MAX^{TM} make up both the skeleton and the muscles, that is the bones move by themselves and are not affected by any muscles. If the animators had to define the muscles too, it would be much more complicated although such research has been done as was discussed in the previous section.

To these bones a skin is attached, so that when the bones move, the skin moves also. This means that we can regard 3D characters in computer games as paper figures, driven by a mechanic skeleton. Every vertex in the skin is attached to a few bones, mostly two or three with a weight for every bone. The bones affect the vertices with a transformation. By weighting every transformation with the weights for these bones we get a kind of averaged transform for the vertex. I will explain how this is done. Every such transform assumes that the vertex is expressed in the local frame of the bone. This is done by transforming the vertex by the inverse world transform for the bone in its reference position. Then the vertex is transformed by the current world transform of the bone. We multiply the resulting vertex by the weight ($0 \leq w \leq 1$) for that bone. By summing all of these weighted vertices we get the final vertex. This can be expressed using a sum over the weights. If v^{Init} is the initial vertex, w_i^v the weight for bone i for the vertex v, X_i^t the world transform of bone i at time t and $(X_i^{Init})^{-1}$ the inverse of the initial world transform of bone i, $v(t)$ the vertex at time t becomes

$$v(t) = \sum_{i=1}^{n} w_i^v X_i^t (X_i^{Init})^{-1} v^{Init} \tag{3.9}$$

The sum of the weights to the bones for a specific vertex is one

$$\sum_{i=1}^{n} w_i^v = 1 \tag{3.10}$$

What we do conceptually, is to create as many rigid body transforms of the whole mesh as there are bones, and calculate a weighted final mesh. The resulting animation looks convincing enough.

It is important to have proper tools to make it easier to do a good weighting of the bones to the vertices. In 3D Studio MAX^{TM} this is done using envelopes. An envelope is a kind of force field where the weight to a vertex decreases with the distance from the bone. If you don't weight a vertex properly to the bones it will have artefacts such as when the bone moves into certain angles the mesh surface will be distorted. Ensuring that a vertex close to two bones is equally weighted to both is important. These and more things are found when dealing with bones in practice. Good tools will however never make up for the flawed architecture of the algorithm. A problematic area is in the joints, where the limbs will sink in when bent, as seen in Figure 3.4. Another artefact area is when two limbs are extended along a common axis, and one of the limbs is rotated around the common axis. An example of this last problem is when a character needs to open up a door, which demands that the upper arm is rotated around the elbow. The arm will thin out around the elbow area between the lower arm and upper arm, yielding totally unrealistic visuals. An exaggerated such visual artefact is seen in Figure 3.5, where the arm has been bent from its original position in Figure 3.6.

We want the skin, not to be attached to a specific bone animation, just a specific set of bones. The parameter $(X_i^{Init})^{-1}$ is the inverse transform of the bone animation pose that generated the specific mesh. Ensuring that the transform is saved along with the mesh, we get independence of different bone animations. Exporting the inverse transforms with the mesh is even better as one does not have to invert them at load time. If one does not save the transforms along with the mesh one would have to export a mesh for every bone animation, which requires more storage. For example, the model format of Half-Life^{TM} has a mesh with a bone reference set-up attached to it. I noticed it while I was exporting models from Counter-Strike^{TM}, which is a game built using Half-Life^{TM} technology. Another approach would be to save the mesh, weighting information and the reference transform separately from each other. This way the mesh would not be tied to a specific bone structure, and the animation or bone structure would not be tied to a specific mesh. Ultimate flexibility is not always practical though, since constraints have to be maintained. Such constraints would be configuration management

Figure 3.4: The elbow joint is sunken when it is bent.

Figure 3.5: Upper arm bent 180 degrees around the shoulder.

Figure 3.6: Arm before any transformation.

of the three components.

3.3 Parametric Animation

A couple of interesting ideas emerge when considering that a mesh can use any bone animation, since it is detached from a specific animation. The bones can be manipulated in any way and then they are applied to the mesh. To be able to change the parameters of a bone animation individually the animation of the bones must be local and being divided into translation, rotation and scale. In my work I have chosen only to discuss rotation. Actually it is also useful to animate the scale of the bones. For example if you want an animation where the figures breathe, you would only need to scale up and down the bones affecting the torso, in the scale track of the bones. To keep it simple we will only discuss rotations. One first observation is that we can define some fundamental operators onto rotational hierarchies. Such an operator I have chosen to call addition, which merges two animations. A merge of rotations is done bone by bone. The root bone is merged with the root bone in the second animation, and so on for all bones in the hierarchy. This of course assumes that the trees of bones are the same. Why would you want to add two animations? Well if you have an animation with a man walking and another with a man waving, by adding the two animations a third animation that is the composite of the two is produced. Merging two animations this way saves a lot of extra work for the artists and requires

less memory since we use one less animation. We can now produce all the animations separately and have the composites of the animations by adding the appropriate ones together. Transitions from one animation into another are also handled by adding more of the starting animation than the target animation in the beginning, and in the end add more of the target than the start. This means that we can ramp up and down the addition coefficients. That's why it is called parametric animation.

3.3.1 Interpolation of Rotations - Quaternions

To interpolate rotations about different axes is not trivial. There is a fast and easy way using quaternions. Quaternions were invented in 1843 by Sir William Rowan Hamilton. They are used extensively by robotics researchers. Robotics researchers have to deal with trajectories and rotations of robot parts, which makes quaternions very useful for them. What are they then? They are mathematically an extension of the complex numbers. Sir Hamilton presented a complete algebra for them in 1853. I will use the algebra, defining and deriving only parts of it. The interested reader should look for any book by Sir Hamilton himself. A quaternion can be written as a complex triplet and a real number.

$$\hat{q} = (q_v, q_w) = (q_x, q_y, q_z, q_w) = iq_x + jq_y + kq_z + q_w \tag{3.11}$$

The numbers q_x, q_y and q_z are called the imaginary part and q_w is the real part. For the algebra to hold we need to define some more, namely the i, j and k relations.

$$i^2 = j^2 = k^2 = -1 \tag{3.12}$$

$$jk = -kj = i \tag{3.13}$$
$$ki = -ik = j \tag{3.14}$$
$$ij = -ji = k \tag{3.15}$$

Using these definitions we can derive addition and multiplication which become

$$\hat{q} + \hat{r} = (q_v, q_w) + (r_v, r_w) = (q_v + r_v, q_w + r_w) \tag{3.16}$$

$$\hat{q}\hat{r} = (q_v \times r_v + r_w q_v + q_w r_v, q_w r_w - q_v r_v) \tag{3.17}$$

Note that multiplication is not commutative. To not lose perspectives, let us remind ourselves that what we want, is to interpolate between rotations. Conveniently enough a unit quaternion is the same as a rotation. A unit quaternion is defined as

$$q_x^2 + q_y^2 + q_z^2 + q_w^2 = 1 \tag{3.18}$$

This can be interpreted in the fourth dimension as if it lies on the unit hypersphere. It can also be represented as

$$\hat{q} = \cos(\frac{\theta}{2}) + v\sin(\frac{\theta}{2}) \tag{3.19}$$

This unit quaternion is the same as a rotation in three space about an axis v with angle of θ.

Spherical linear interpolation or slerp interpolates two quaternions along the shortest arc between them, as in

$$Slerp(\hat{q}, \hat{r}, t) = \hat{q}\frac{\sin((1-t)\Omega)}{\sin(\Omega)} + \hat{r}\frac{\sin(\Omega t)}{\sin(\Omega)} \tag{3.20}$$

Here $0 \leq t \leq 1$ is the interpolation coefficient and $\cos(\Omega) = \hat{q} \cdot \hat{r}$. Where $\hat{q} \cdot \hat{r}$ is the inner product which is defined as

$$\hat{q} \cdot \hat{r} = q_v \cdot r_v + q_w \cdot r_w \tag{3.21}$$

Given two quaternions $\hat{q}$ and $\hat{r}$ there are two different rotations that interpolate $\hat{q}$ into $\hat{r}$. A rotation with a quaternion $\hat{r}$ is the same as $-\hat{r}$. Such pairs of quaternions are called antipodal, since they lie opposite on the hypersphere. We want the shortest way from $\hat{q}$ to $\hat{r}$. That can be accomplished by first checking the magnitudes $|\hat{q} - \hat{r}|$ and $|\hat{q} + \hat{r}|$. If $|\hat{q} - \hat{r}|$ is less, then the shortest arc is between $\hat{q}$ and $\hat{r}$, but if $|\hat{q} + \hat{r}|$ is less, then the shortest arc is between $\hat{q}$ and $-\hat{r}$, and we substitute $\hat{r}$ with $-\hat{r}$ before doing a slerp. Thinking about quaternions is not straightforward as the path they make is on a four-dimensional hypersphere. Doing a spherical linear interpolation between more than a pair of quaternions is not trivial. Ken Shoemake tackled the problem in 1985 [53]. He used a cubic Bézier curve to interpolate the quaternions using six slerps, something that is called a de Casteljau construction. A more refined method is the one presented in Barr et al [6], that has better continuity of angular velocity than Shoemakes curves, but the interpolation can take several minutes to perform, ruling out real-time use. A more modest technique of interpolation of quaternions, is using Kochanek-Bartels splines as described in [25]. They use only three slerps and base their continuity on the tangents of the two keys in a spline

section. 3D Studio Max^{TM} uses this kind of interpolation for the rotations. I settled for a straight linear spherical interpolation for the animation system, mainly because it is a simple approach, and since I had enough keys to keep the orientation changes small.

3.3.2 Bone Poses

Using the slerp we can now define addition of two rotational hierarchies. For every bone in the two hierarchies we transform the axis-angle representation into quaternions, then do a slerp between them, giving an in between value. This in between value is a new quaternion. To keep a simple way of thinking and to facilitate the programming, I have developed a structure that I call a *bone pose*. A bone pose is a set of rotations of a bone animation at a certain time. You could also call a bone pose a photo of an animation frame that is not further interpolated. This photo is a quaternion representation of the rotations. With this way of thinking it is easier to treat a bone animation addition since we now work in quaternion space all the time. For example an addition of three animations is written as

$$\begin{aligned} &BoneAnim(time1) \rightarrow BonePose1 \\ &BoneAnim(time2) \rightarrow BonePose2 \\ &BoneAnim(time3) \rightarrow BonePose3 \\ &BonePose \leftarrow BonePose1 + BonePose2 \\ &BonePose \leftarrow BonePose + BonePose3 \\ &BoneAnim \leftarrow BonePose \end{aligned} \tag{3.22}$$

The only thing we have to remember is to transform the quaternions into regular rotations when we want to transform geometry. We have just defined an addition operator for quaternion rotational hierarchies or bone poses for short.

3.3.3 Key Reduction

If one has a lot of animation frames, key reduction can save some space. We can use the quaternions for this also. By using the definition of the inner product (3.21) we get the magnitude of the angle between two quaternions. Using this angle one can reduce the number of keys one gets from a motion capture sequence, that are not needed. Those that are not needed are of course keys that have a small enough angle between each other. Most major 3D modelling packages already implement some kind of key reduction. An idea that I have but did not implement, since there was no need for key reduction, is to use a cubic spline on the quaternions. Instead of rejecting or

keeping keys on the relative angular difference, one could refine the rejection process by sampling cubically interpolated quaternions, between the first key and the last key, using the threshold angle as the sampling rule. This would require an integration along the quaternion path.

3.3.4 Animation Blending

The technique of adding animations was inspired by Jeppson [23], actually it is his method that I have explained, but I had to find out the inner workings for myself since I had to implement it as a subsystem. I was the opponent to his master thesis in computer science and engineering at Lund Institute of Technology, which led to that I was greatly inspired by his work. I adopted the term parametric animation from Valve Software Inc who used it as a name for their new motion blending feature in their Half-Life^{TM} engine. The bone pose solution was inspired by Granny^{TM}, a bones SDK I read about on the internet [15]. In the Granny FAQ they explained that their inverse kinematics were applied to a hierarchy and then blended with the animation. I figured that one could use the same approach to animation. That is, to have a specific record with only a single animation frame. Thus bone pose algebra was invented. Someone might argue that this is elementary, - I simply put it in words. Sure, why not, I just like to call it an algebra. It is not an algebra in the strict sense since there is no multiplicative inverse for a bone pose for example.

Bone Importance

A bone pose consists of a hierarchy of bone structures that hold a transform which consists of a quaternion, a translation and a bone importance weighting value for the bone. I have only mentioned that the rotations are merged with the use of the quaternion slerp. If we always wanted that the bones in each animation were equally important we could do with a t-value of 0.5. However consider an animation with a man walking and another with the same man waving his arm. The resulting composite animation should look like a man walking and waving his arm. This will not be the case if we use equal weighting for the arms and legs in the two animations. Instead we can weight the legs in the walking animation more and the waving arm in the waving animation more. Let's do it for a simple case. The quaternions $\hat{q}$ and $\hat{r}$ have weights $w_{\hat{q}}$ and $w_{\hat{r}}$. Here $\hat{q}$ is the from quaternion and $\hat{r}$ is the to quaternion. Expressing this as a combination of the two we get

$$\frac{w_{\hat{q}}}{w_{\hat{q}} + w_{\hat{r}}}\hat{q} + \frac{w_{\hat{r}}}{w_{\hat{q}} + w_{\hat{r}}}\hat{r} \tag{3.23}$$

We can now calculate the interpolation t-value.

$$t = \frac{w_{\hat{r}}}{w_{\hat{q}} + w_{\hat{r}}} \tag{3.24}$$

Using weights, bones that are more important are emphasized. I tried to calculate the bone importance weights algorithmically. My presumption was that limbs that move a lot, are more important. Using the quaternion inner product (3.21), I calculated the sum of the relative angular displacements. The result was not convincing but adequate for my initial tests, where automation was my main concern. For example the arms in the walking animation were swinging a lot, although they are not as important in the composite as the waving arm. A remedy would be to also consider the angular derivative, which I did not test, since I am not convinced that subtle movements would be emphasized enough. When I was an opponent to Jeppson [23] I asked him the question if he did not automate the bone importance weighting somehow. He didn't at the time, and relied on the artists to do it. He used a weighting scale of unimportant, important and very important. A set of weights could be 10, 50 and 400. It is mainly an editing question. When adding more than two animations an idea is to keep the highest bone importance weight in the bone pose structure. In my implementation I simply created a text file that was parsed to set the weights for different animations. This way editing was very easy.

Animation Transitioning

When adding two animations you have to blend in the new animation gradually to get a smooth transition between the two. For example blending with the man waving and the man walking would become

$$\begin{aligned} &WaveAnim(time) \rightarrow WavePose \\ &WalkAnim(time) \rightarrow WalkPose \\ &CompositePose \leftarrow (1-t)WavePose + (t)WalkPose \\ &CompositeAnim \leftarrow CompositePose \end{aligned} \tag{3.25}$$

The t parameter is the lead in of the walking animation, going from 0 to 1. The characteristic of the function of this lead in, depends on how fast you want the other animation to take charge. A nice function is for example the cosine of u ranging from 0 to π, where t is $(1 - \cos u)/2$. It will have a nice gradual change of animation.

3.4 Inverse Kinematics

As it was described in the introductory chapter, inverse kinematics is to calculate a trajectory given starting and stopping conditions. It is trivial to do it for a single object but not trivial if it is linked to another object. This is why when you talk about inverse kinematics, you implicitly imply a linked structure of two or more links. I encountered two different methods for solving IK in Chris Welmans Master Thesis [60]. One is called the Jacobian transpose method and the other is called Cyclic Coordinate Descent or CCD.

3.4.1 Cyclic Coordinate Descent - CCD

The CCD method is a heuristic iterative method for finding the trajectory of a link in a chain. The method assumes that the outermost link or end effector as it is called in robotics [8], wants to get as close to the end point as possible. Since the links are not able to move relative to each other, the only thing they can do is to rotate relative each other. CCD starts by rotating the end effector so that it gets closer to the end point. If we aren't near enough, we do the same for the next link but we rotate it so that the end effector is as close as possible, not the link itself. Doing this down to the root for all links is one whole iteration. This is done until a maximum number of links have been traversed or the end effector is near enough the end point. The method does not necessarily find the optimal solution, since it is a greedy locally optimising algorithm. CCD has its benefits though, because you as a programmer do not have to care about special cases such as chains that have no chance of reaching the end point. The chain will try to reach the end point. When I first read about how the algorithm went about doing this optimisation, I didn't get why I had not thought of this super simple solution before.

3.4.2 Jacobian Transpose

Also in Chris Welman's thesis [60] I found a treatise of a more refined method called the Jacobian transpose method. Although he used CCD in the running test system of his, the Jacobian transpose method deserves an exposition. Introduced 1984 by Wolovich and Elliot [64] it is based on the idea that an elastic force attracts the end effector to the end point. If the destination point is $x_d(t)$ and the end effector's tip is $x_c(t)$ then the error metric $e(t)$ is simply

$$e(t) = x_d(t) - x_c(t) \tag{3.26}$$

A force F applied on the end effector consists of a pulling force f and torque m.

$$F = [f_x, f_y, f_z, m_x, m_y, m_z]^T \tag{3.27}$$

The force F will cause internal forces and torques on the joints of the chain. Under the simplifying assumption of virtual work introduced by R. P. Paul [48] the relation between F and the generalized forces τ is

$$\tau = j^T F \tag{3.28}$$

If we are only interested in the angular velocity in the link joints, then it is only needed to think of τ as the vector of joint angular velocities.

$$\dot{q} = j^T F \tag{3.29}$$

Here q is the angle of the joint in three dimensions. When $\dot{q}$ has been calculated, the next q can be integrated. This new q brings the end effector closer to the destination. The algorithm iterates until $e(t)$ is small enough or some other criteria is true, for example number of iterations. One can regard the error metric $e(t)$ as a pulling force on the chain. A description of how to calculate the Jacobian and its transpose is found in [60].

3.4.3 Comparison of the Two Methods

Which one of the two inverse kinematics methods is to be preferred then? The Jacobian transpose method yields a more natural appearance to the links, since they are calibrated using force calculations, which makes the chain look more naturally balanced. The Jacobian method and CCD are equally fast during normal operation, but the Jacobian method is slower, when the destination point is in certain positions. Such positions are for example points inside the chain. This will take more iterations and the Jacobian transpose method will not give the chain the natural looking appearance any longer. As a side note, Welman himself used the CCD method in his test system.

3.4.4 The Problem of Control

Other than considering an algorithm, it is equally important to select an appropriate number of links, to be manipulated by the algorithm. The more links the less control of the output. For two-link structures, there is a strict analytical approach using geometrical analysis. Using a method such as CCD is easier and more general. In an articulated figure, separating specific chains of few number of limbs for solving, is preferable. An example by Chris

Welman [60] is how un-intuitive it would be to change the position of the spine by pulling the fingers of a character.

Chapter 4

Parametric Dynamics

A parametric function depends on one or more input values, that can change within specific intervals. What then, is parametric dynamics? The name is taken from the technique, of mixing animations of humanoid figures, which is called parametric animation. What differentiates them is that the animations that are mixed are not all recorded in parametric dynamics, rather some are simulated in real-time. On the other hand they resemble each other in that they both mix animations, which means it is an addition to parametric animation or a hybrid model of static and dynamic animations.

One very important aspect of animation is a predictable or controllable result. While a pre-generated animation is static, a simulated is dynamic. We need to set regulating functions on the simulated animations, to control their behaviour. This can be done by only simulating those body parts that are interesting for the situation, or to simulate only a few set of bones. In this spirit dynamic situations were invented, that besides telling which body parts are interesting for the simulation, also contain more parameters.

Those animations that are produced from the simulation are mixed with the recorded ones in a blending pipeline. The blending pipeline is the same as the one developed in the animation chapter, using the so called bone pose operators.

Ultimately the problem is then to feed the correct information into the simulator, and to extract the right information from the simulator, to be used in the blending pipeline. In the next chapter we will see this knowledge into use and analyse the results.

4.1 Dynamic Situations

In computer games there often arises dynamic situations, for example when a character is affected by outside forces. Such a situation can be that someone is shot, which is often the case, or that someone is pushed, or someone may stumble and fall, or be run over by a car. To take care of all these varying situations, in a single generic system is very complex. Why? To be able to understand the question, we can start to ask ourselves another question. What is it that makes us perceive something, as human behaviour, that is what is it that distinguishes us from robots? We have muscles, sinews, bones, joints, cartilage, skin and reflex behaviours. We have the ability to adapt ourselves to situations that arise more than once, or those that appear similar. To simulate a human is more than to simulate its physical body. You need to take into account the human perception of the surrounding world as well. There is a big difference in how a physically fit person performs its motions when compared to an unfit person. To completely simulate a human being we need to do a complete physical and mental simulation. This demands a lot of knowledge of man, and right now we don't have the knowledge of how the mental processes in man work. It is also probably very costly to do a complete simulation of a person. Martin Rystrand my supervisor at Massive Entertainment AB, told me about the animator of the Paladin character in the game Diablo II^{TM}, by Blizzard Entertainment Inc. The character had taken more than a year and a half to complete, with the animator working on it full time. Tools of today are mostly recorded animations of humans and animals. Such libraries of motion are supplied by several companies.

It hit me one day, that why not let the animator take care of the more complex mental processes and let the computer take care of the easier physical. The physical processes could not run for a long time though, since then the illusion of man, would disappear. A shorter physical influence of a character mixed with a recording of a complex animation, would facilitate the working process of the animator, or so I believed. For example a physical influence could be to be shot, and the complex could for example be to walk or to run. By mixing the bone pose of the simulated animation, with the bone pose of the recorded animation, we have achieved the hybrid model. Is there then a limitation to what can be a dynamic situation, and how many different recorded animations, it can be mixed with? Beforehand we can say that instantaneous and simple motions, are best fit to be dynamic situations. The longer a movement takes to simulate, the less convincing it becomes. Defining what the best recorded animation would be, is a tougher question, and has to be tested. The most attractive aspect of this method, is that you can affect a character by several dynamic situations, at the same

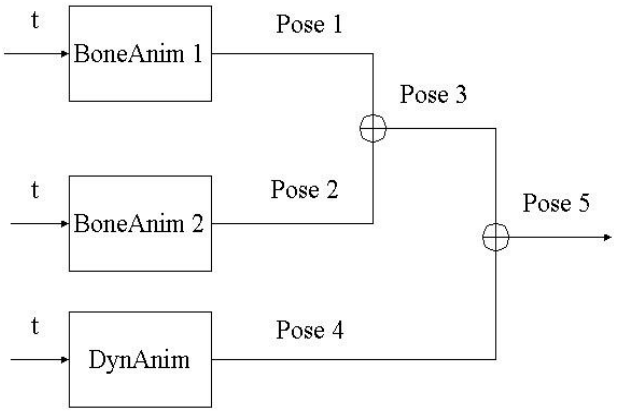

Figure 4.1: An example schematic of a blending pipeline.

time.

4.2 The Blending Pipeline

In Chapter 1 you were introduced to the bone pose operators, unary multiplication by a scalar and a binary addition of bone poses. Using these two operators on a set of of identical hierarchies of quaternions, we can define a bone pose blending pipeline. We can begin to illustrate it with a simple schematic as in Figure 4.1. Observe though that the transitioning units which in this case would be multiplicative units, are not considered in the schematic for the sake of legibility.

If we want to be able to insert dynamic bones into the scheme we need to have a function or black box that gives us a bone pose. What does such a black box look like? Let us take a look inside the DynAnim component, referring to Figure 4.2. What have we here then? The two most important components are the Dynamics Initialiser and the Dynamics to Pose component. We need to know more about the dynamics system and how it relates to bone poses. In the next section we will take a look at the initiating component.

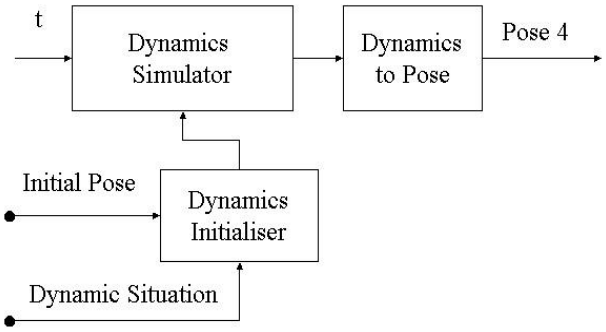

Figure 4.2: An inside look of the DynAnim component.

4.3 Feeding the Dynamics Simulator

The dynamics simulator was originally written in 1998 by Brian Vincent Mirtich and James Kuffner Jr, and is copyrighted by MEITCA[1]. Following the test example that was done by Brian Mirtich and James Kuffner, and by rigorously analysing the calls mathematically, I could understand how the system worked. There was no documentation so this was the only way to do it. But as the saying goes, don't check the teeth of a horse given to you. Porting the test from its original platform of OpenInventor^{TM} to OpenGL^{TM} was quite easy, thanks to the authors who had been farsighted and prepared the system for such a porting. The theory of the system was described in the Chapter 2, that dealt with tree structures of rigid bodies and the reasoning about Featherstone's algorithm, where the dynamic links were connected by joints. But we have only bones in 3D Studio MAX^{TM}. Bones are transforms in a hierarchic structure, so how do they relate to dynamic links? First we need to know what a dynamic link is, and how we can describe one.

4.3.1 A Dynamic Link

A dynamic link is a pendulum, that is attached with its arm to an inboard link, and where the centre of the pendulum ball is in the origin of the outboard link. To visualise the direction of the pendulum system we can use a rhombus. We place the rhombus origin in the outboard bone system, where its direction is such that its z-axis points in the vector from the origin of the inboard bone system to the origin of the outboard bone system. In Figure

[1]Mitsubishi Electric Information Technology Center America

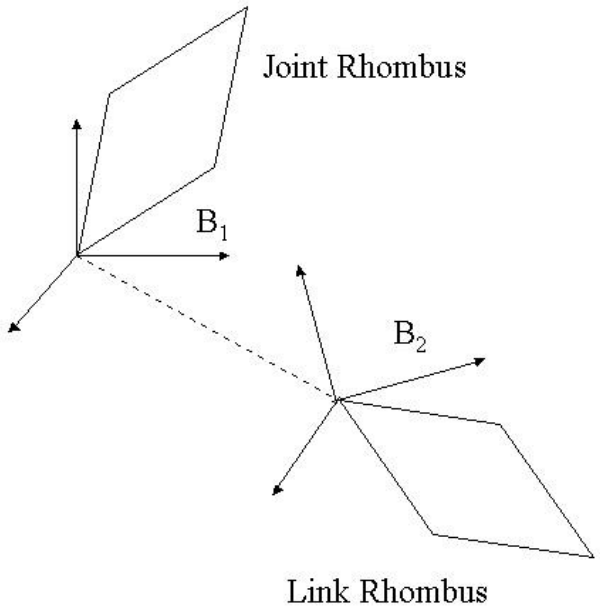

Figure 4.3: A dynamic link visualised using rhombi.

4.3 the Link Rhombus is the pendulum system and the Joint Rhombus is the system of the joint.

The dynamic link's axis of rotation must be orthogonal to the direction of the link system, which also means it is orthogonal to the pendulum arm. It is also rotated about the link system z-axis, to be able to control the plane of motion of the pendulum. In Figure 4.3 the inboard bone system is B_1 and the outboard bone system is B_2. The reason to why I used rhombi, is that you can then easily distinguish the system's direction, which would be harder if we used a sphere for the visualization of the pendulum origin. If we instead describe the rhombi using coordinate system arrows, we can get a clearer picture of how the dynamic link is related to the bone transforms. Before we do that, we can stop to think for a while, and establish why we need to do this analysis. To be able to represent the dynamic link to the dynamics system, we need this analysis. When initiating the hierarchy to be simulated by the system, you describe two dynamic links and their relation to a common joint system, using two transforms. A new link that is connected to the system is called an outboard link, while the link that is the anchor in this relation is called the inboard link. The two transforms are such that they are relative a common joint system. This joint system is local to the outboard link with one transform and relative the inboard link with another

transform. Since this analysis is quite extensive it demands a new section.

4.3.2 Dynamic Links' Relation to Joints

This section and the next ones deal much with transforms. Chiefly the rigid body transform will be used. We use the same notation as Chapter 1.

$$ {}^{i}_{o}X \tag{4.1} $$

The above form means the transform that takes coordinates described in the outboard system, and transforms them into coordinates described in the inboard system. What this whole discussion will be about is the transform from outboard to inboard coordinates and how it relates to the rotational joint of the outboard link. Actually we can describe this relation using three transforms

$$ {}^{i}_{o}X = {}^{i}_{j}X X_{R_j} {}^{j}_{o}X \tag{4.2} $$

The middlemost transform X_{R_j} is the local rotation in the joint, and is simply a rotation about the z-axis. The interesting transforms are ${}^{i}_{j}X$ and ${}^{j}_{o}X$, that are fed into the call that connects two links. Why does the dynamic system need these two transforms? After each update of the dynamics system it computes the transform ${}^{i}_{o}X$, but to be able to do this it needs information of how the links relate to the joint so it needs these representations. Actually ${}^{i}_{j}X$ and ${}^{o}_{j}X$ are fed and not the whole matrices either. Why ${}^{o}_{j}X$ is needed instead of its inverse is a mystery to me, but I guess it makes it easier to input the transforms by hand. I said that not the complete matrices are fed into the connect call, and to know why we can take a closer look at the rigid body transform which is a rotation followed by a translation, as in

$$ {}^{o}_{j}X = {}^{o}_{j}T \, {}^{o}_{j}R \tag{4.3} $$

Here ${}^{o}_{j}T$ is the origin of the joint system described in the outboard system, and ${}^{o}_{j}R$ is the rotation for the joint system relative the outboard system. If we fill the matrices with symbols and concatenate the two we get

$$ {}^{o}_{j}X = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 & Y_1 & Z_1 & 0 \\ X_2 & Y_2 & Z_2 & 0 \\ X_3 & Y_3 & Z_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} X_1 & Y_1 & Z_1 & T_x \\ X_2 & Y_2 & Z_2 & T_y \\ X_3 & Y_3 & Z_3 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4} $$

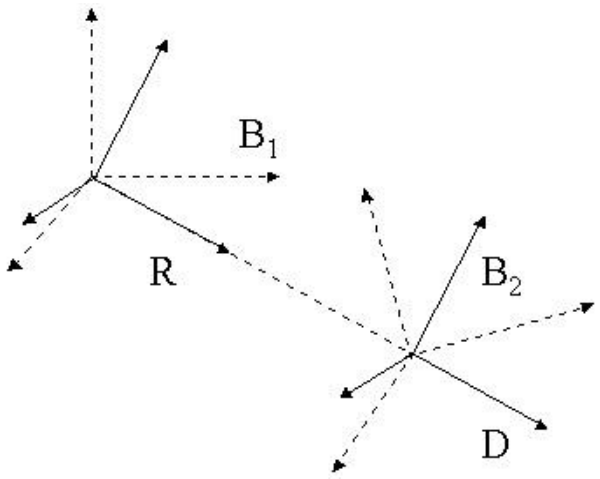

Figure 4.4: A dynamic link's relation to its bones. The dynamic link system D and the rhombic system R. Systems B_1 and B_2 are dotted.

The three rotational axes X, Y and Z are all orthogonal to each other, which means it is enough with knowing two of them to get the third. The dynamics system then needs 9 parameters chosen to be X, Z and the translation T. Naming these parameters as the system does, we have outRef, outAxis and outLoc respectively. Now that we have analysed the matter exhaustively we can go on to computing these relations based on the bone transforms. By the way, I thank Brian Mirtich for ratifying my queries regarding the above transforms.

4.3.3 Dynamic Links' Relation to Bones

In the creation of a dynamic link we originate from the bone systems. An illustration can be of help as in Figure 4.4, where B_1 and B_2 are the bone transforms that create the dynamic link D. How then, is D related to B_1? We are looking for

$$\begin{smallmatrix}\mathrm{B_1}\\ \mathrm{D}\end{smallmatrix} X \tag{4.5}$$

Since the dynamic link D is to be related to B_1 the rhomb transform R is used as the rotation for D and the offset for B_2's origin described in B_1 becomes the translation, which gives us

$$\begin{smallmatrix}\mathrm{B_1}\\ \mathrm{D}\end{smallmatrix} X = \begin{smallmatrix}\mathrm{B_1}\\ \mathrm{D}\end{smallmatrix} T \begin{smallmatrix}\mathrm{B_1}\\ \mathrm{D}\end{smallmatrix} R = \begin{smallmatrix}\mathrm{B_1}\\ \mathrm{B_2}\end{smallmatrix} T \begin{smallmatrix}\mathrm{B_1}\\ \mathrm{R}\end{smallmatrix} R \tag{4.6}$$

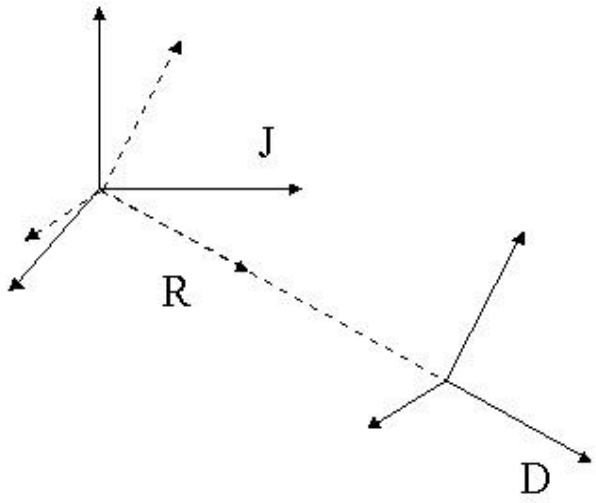

Figure 4.5: The joint system J related to the dotted rhombus system R and the dynamic link system D.

Great then, we now know how the dynamic links are built relative bone transforms. It takes two bone transforms to build a dynamic link, but a dynamic link can only affect one bone transform, otherwise it will not correspond to how 3D Studio MAX^{TM} handles bones.

4.3.4 Dynamic Links' Relation to Joints II

Now what remains is to relate the bones to the rotational joints which we will now term j. This is needed to be able to compute the transforms ${}^{o}_{j}X$ and ${}^{i}_{j}X$. First we need to graph how the joint system is related to the dynamic link, as seen in Figure 4.5. The rotational joint system j is only rotated relative to the rhombus system R. This rotation is such that the R system's z-axis is rotated from R so that it is aligned with its own x-axis. Then the system is rotated around the R systems z-axis with α degrees. As always we can express it with a beautiful transform

$$ {}^{\mathrm{R}}_{j}X = \mathrm{R}(\alpha, 0, 0, 1)\mathrm{R}(90, 0, 1, 0) \tag{4.7} $$

With this knowledge we can now set up the following equation

$$ {}^{\mathrm{D}}_{j}X = {}^{\mathrm{D}}_{\mathrm{B}_1}X \, {}^{\mathrm{B}_1}_{\mathrm{R}}X \, {}^{\mathrm{R}}_{j}X \tag{4.8} $$

We now know how an outboard link is related to its joint. To connect two dynamic links we also need the relation between the inboard link and

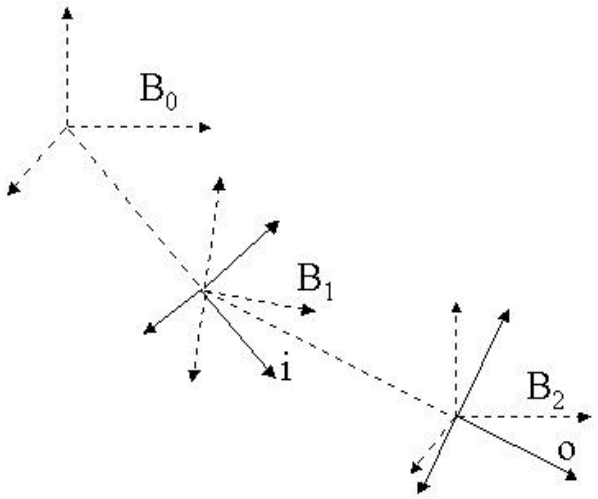

Figure 4.6: Inboard system i and outboard system o. Dotted bone systems are B_0, B_1 and B_2.

the joint. At the connecting moment we have two dynamic links, called the inboard link which is the anchor and the outboard link which is the new link that we want to attach. The rotational joint is part of the outboard link. Let us again please our eyes with a figure, see Figure 4.6. To make the figure more legible I did not include the joint system which is described in the bone system B_1. We already have ${}^{o}_{j}X$ as the outboard link is the dynamic link system D as in the transform ${}^{D}_{j}X$. What remains is ${}^{i}_{j}X$ but we have a number of leads. The bone system B_0 is related to B_1 with the known transform ${}^{B_0}_{B_1}X$, and we know how a bone is related to a dynamic link. This brings us the following equation

$$ {}^{i}_{j}X = {}^{i}_{B_0}X \, {}^{B_0}_{B_1}X \, {}^{B_1}_{j}X \tag{4.9} $$

It is trivial to solve for ${}^{B_1}_{j}X$ since it is

$$ {}^{B_1}_{j}X = {}^{B_1}_{R}X \, {}^{R}_{j}X \tag{4.10} $$

Finally we can put the case with the dynamic link initiation to rest. What awaits now, is how to transform the information we get from the simulator.

4.4 Feeding the Blending Pipeline

When the simulator is given a description of a skeleton it gets the information from the Initial Pose signal in Figure 4.2. From this Initial Pose the Dynamics Initialiser knows how to connect the dynamic links. On the other hand the Dynamic Situation signal, gives which bones that are to be simulated, and with which angles about the dynamic links, the rotational joints are rotated. When the simulator goes through an update we get back the new relations between the dynamic links, in the transform

$$ {}^{i}_{o}X \tag{4.11} $$

The problem is that what we really need is the transform for the bones ${}^{\mathrm{B}_0}_{\mathrm{B}_1}X$. That is the outboard link in Figure 4.6 rotates the bone transform B_1. If we express this transform using dynamic links, and what we know of their relations, we get

$$ {}^{\mathrm{B}_0}_{\mathrm{B}_1}X = {}^{\mathrm{B}_0}_{i}X \, {}^{i}_{o}X \, {}^{o}_{\mathrm{B}_1}X \tag{4.12} $$

We can see clearly that all components are already known from the reasoning about the initiation of dynamic links. Knowing this we can feed a new bone pose into the blending pipeline. The dynamic situation tells us what bones are simulated, so these bones get a high mixing coefficient while the not simulated ones get a mixing coefficient of 0. It is time to put the theory into practice, which we will do in the next chapter.

Chapter 5

Tests

To be able to test something you need to be able to measure something. The problem I was faced with was to make the simulated animation look good, which is a subjective metric thus a subjective scale is needed. Such a metric I have chosen to call the Gestalt Metric or GM for short, which I divide into three categories,

GM Good

- The virtual character has natural motion. This is exhibited by joints that are properly constrained, motion that is not too fast, a sense of restrain in the limbs and a long enough relaxation period for the muscles.
- The limbs cannot penetrate.

GM Less Good

- The virtual character has natural motion.
- The limbs can have some minor penetrations.

GM Bad

- The character has no natural motion and has no sence of balance.
- The limbs penetrate.

5.1 Test Parameters

When I started the test phase I included quite a lot of test parameters, since I did not know which ones were important for the outcome. I divide the parameters into two categories, the static and the dynamic. A test such as being shot in the arm has several static parameters, such as which bones are affected by the simulation, the masses of the links, the initial link parameters which are derived from the bones and the link dampenings. Then we have the dynamic parameters, which are link velocity and link angular plane of rotation. You might wonder why I set the angular plane of rotation into the dynamic parameters, and it is a legitimate wondering. In the case of an arm we do not have many degrees of freedom except for the shoulder which has at least three rotational planes. Since I wanted to have simple manipulators I used one degree of freedom for all the joints, which greatly simplified the implementation. To add more degrees of freedom to a body part you need to add extra bones in its vicinity, preferably with a length of zero. So it is solvable, but for my testing I kept the skeleton as it was and instead changed the plane of rotation whenever a joint with more than one degree of freedom was hit. The static parameters are always the same while the dynamic are dependant on the incident force.

Parameters that do not really fit into neither the static nor the dynamic category are the transition parameters. I use terms from digital music synthesising to define them. The attack is how long time it takes for the simulation to take full effect, using a cosine ramp, of $(1 - \cos u)/2$. When the attack period is over the sustain period is reached, where the simulation is the only rotation which affects the bones. The last period is maybe the most important, and I have chosen to call it relaxation instead of the synthesising equivalent of release. It is necessary to have a long relaxation period to make sure the person doesn't look like a super human whenever he is shot. During the relaxation the muscles should relax from the superimposed simulated animation back into the kinematic animation. This is done using an inverted cosine ramp. A complete transition sequence is seen in Figure 5.1.

An important parameter is how many and what bones are simulated by the system. To simulate a gun shot in the arm, it may be better to also simulate some bones in the torso to get a more dramatic effect. In the running system linear linkages of bones are supported, not trees. This is currently solved by using two separate linkages. The dynamics system supports trees, as was discussed in Chapter 2.

It may be of interest to describe how these parameters were implemented and edited. My philosophy was easy editing and easy parsing, so I used text files where each bone was on a single line along with the parameter value. The

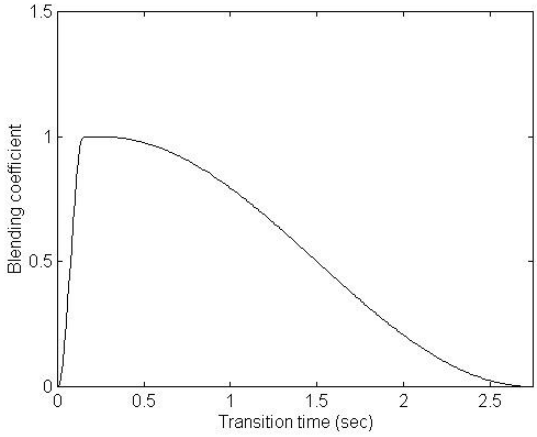

Figure 5.1: A transition function for a shot in the arm. Note the short attack time, and sustain time, followed by the long relaxation time.

parameters that are edited this way are; which bones that are simulated, link mass, joint rotation, link velocity and link dampening. The other parameters were defined similarly in a text file including, attack, sustain, relaxation and simulation time. Using the approach of saving all parameters in text files, fine tuning of individual links was comfortable. An example of such a text file is shown in Figure 5.2

5.2 Tests

The tests describe what static and dynamic parameters were changed and why. The static parameters within each test have to stay the same, because all tests have at least two sub-tests where the character is shot from different angles. All of these sub-tests need to have the same static parameters within a test. These are mass, which bones that are affected and dampenings. Actually dampening is the same for all tests, since it was dicovered that it did not matter what the dampening was. It is not significant since it needs a very long time before it can slow down the velocity of the links considerably. Instead of a high dampening, a short simulation period was chosen. This way when the linkage was going into an abnormal configuration, the simulation was stopped using the simulation time parameter.

The mass parameter is actually radius but they are equivalent, just remember that mass is a cubic function of the radius. Most radiuses were set to 10 meters which gives a mass of

```
Bip01 | 0
  Bip01 Pelvis | 0
    Bip01 Spine | 0
      Bip01 Spine1 | 1
        Bip01 Spine2 | 1
          Bip01 Spine3 | 1
            Bip01 Neck | 1
              Bip01 Head | 0
              Bip01 L Clavicle | 1
                Bip01 L UpperArm | 1
                  Bip01 L Forearm | 1
                    Bip01 L Hand | 0
              Bip01 R Clavicle | 0
                Bip01 R UpperArm | 0
                  Bip01 R Forearm | 0
                    Bip01 R Hand | 0
      Bip01 L Thigh | 0
        Bip01 L Calf | 0
          Bip01 L Foot | 0
            Bip01 L Toe0 | 0
              Bip01 L Toe01 | 0
      Bip01 R Thigh | 0
        Bip01 R Calf | 0
          Bip01 R Foot | 0
            Bip01 R Toe0 | 0
              Bip01 R Toe01 | 0
```

Figure 5.2: An example of a text file for editing of which bones that are to be simulated. This one is for a shot in the shoulder. A value of 1 means it is simulated.

$$mass = \frac{4}{3}\pi r^3 \tag{5.1}$$

which is 4188 kilograms. It is quite humerous when thinking about it, but then most arms are at least 5 and at most 11 meters long. I did not consider the scale of the figures and the corresponding masses until after the testing was done, but it did not matter as they were giants to the simulation system anyway. The arms and legs have this scale in 3D Studio Max^{TM} and I did not change it before exporting the figures into the dynamics engine. This large scale made it necessary to apply high velocities to the limbs in the tests. Something you will probably notice is the use of rhombi for the visualization of the bones, which is a primitive most modelling packages make use of.

5.2.1 Test I - Walking and shot in shoulder, GM Good

For this test sequence I used four different angles which were; front, back, inside and outside. The scene consists of a man being shot by a small caliber gun, once in the shoulder. The static parameters were chosen so that the person would be shot in a dramatical way. Quite a lot of bones were added, as seen in Figure 5.3, which were the spine and arm bones. The reason to include the spine, was that it made the character lean when shot. All radiuses were 10 meters since it did not seem to matter what radius they had. For the shot from the front and the back the dynamic parameters were as in Tables 5.1 and 5.2.

Bone Name	*Angle (deg)*	*Velocity (deg/sec)*
Bip01 Spine1	-90	600
Bip01 Spine2	45	300
Bip01 Spine3	45	300
Bip01 Neck	45	300
Bip01 L Clavicle	-90	300
Bip01 L UpperArm	-90	300

Table 5.1: Shot in the front.

As can be seen they are perfect opposites in velocity. The velocities are high as mentioned in the section on test parameters, to give the shoulder enough energy to react violently. Why then have I chosen to set the "Bip01 Spine1" bone to have a higher velocity? It seemed as if this bone made the animation more dramatic since the figure leaned over more, and as it is a root bone it affects the other ones. This shows that maybe another parameter

Bone Name	*Angle (deg)*	*Velocity (deg/sec)*
Bip01 Spine1	-90	-600
Bip01 Spine2	45	-300
Bip01 Spine3	45	-300
Bip01 Neck	45	-300
Bip01 L Clavicle	-90	-300
Bip01 L UpperArm	-90	-300

Table 5.2: Shot in the back.

such as a velocity multiplier would be a good idea. If there was a velocity multiplier parameter this bone would also have a velocity of 300, which is desirable since the velocities are to be automatically calculated, and there is nothing that says that this bone would be faster. If we continue to the shots from the inside and outside, we can observe some direct changes in the rotational planes for the joints with more than one degree of freedom, as can be read from Tables 5.3 and 5.4.

Bone Name	*Angle (deg)*	*Velocity (deg/sec)*
Bip01 Spine1	0	-600
Bip01 Spine2	-45	-300
Bip01 Spine3	-45	-300
Bip01 Neck	-45	-300
Bip01 L Clavicle	0	-300
Bip01 L UpperArm	90	0

Table 5.3: Shot on the inside.

Bone Name	*Angle (deg)*	*Velocity (deg/sec)*
Bip01 Spine1	0	600
Bip01 Spine2	-45	300
Bip01 Spine3	-45	300
Bip01 Neck	-45	300
Bip01 L Clavicle	0	300
Bip01 L UpperArm	90	0

Table 5.4: Shot from the outside.

That they are the opposite in their angular velocities is very good since

Figure 5.3: Test I - Walking and shot in shoulder. The bones that are dynamically simulated have a black joint rhombus attached. The spine and the left arm are affected.

that scales well for automation. How do they differ from the front and back shots? The limbs with more that one degree of freedom have a rotational plane rotated 90 degrees. The upper arm bone has a velocity of 0 since it only has one degree of freedom and it is shot in parallell with this plane. All the parameters are indeed very good for automation. The transition parameters were the same for all test cases and are summarised in Table 5.5.

Attack (sec)	*Sustain (sec)*	*Relaxation (sec)*	*Sim Time (sec)*
0.1	0.1	2.5	0.15

Table 5.5: The transition parameters.

It seems as if the best choice of velocities are very high velocities, and instead very short simulation times. If the simulation gets too much time it will not look right at all, more likely the character will look like a big wheel at an amusement park. All in all being shot in the shoulder looked very good and got a GM Good.

5.2.2 Test II - Walking and shot in leg, GM Bad

Being shot in the leg or any other extremity is very common in combat situations. I set the right leg to be affected by a force from the back, front, inside and outside as seen in Figure 5.4. Just as in the shoulder shot scenario

I froze the static parameters to have a radius of 10 meters. The rotational joints were very easy to place using the editable text files, and the result was the obvious one, that is a very mechanical behaviour. Being shot in the legs is not very convincing due to the fact that the leg behaves stiffly. The reason for this could be that I could not add more limbs to the animation. For example a good extra situation would be to also add some motion to the spines. Why could I not do this then? Since the root bone in the right leg hierarchy is attached to another root bone with several children, we would then have a tree structure which I did not support in the prototype. Another reason was that when I did add the extra root bone in the spine, the whole body would roll over like a beetle. What is disturbing is that the person is too balanced during the whole procedure. This however is not a surprise as the parametric dynamics method is a local method, working on isolated chains. An idea could be to add a stumbling animation, which would blend over the walking animation and mix with the simulated animation. As can be seen from Tables 5.6 to 5.9 this situation lends very well to automation if we consider the angles and velocities. The most troublesome animation was the shot from the inside as the right leg penetrated the left leg. Penetrations would here have to be calculated in real-time and avoided.

Bone Name	*Angle (deg)*	*Velocity (deg/sec)*
Bip01 R Thigh	90	300
Bip01 R Calf	90	300
Bip01 R Foot	90	300
Bip01 R Toe0	90	300

Table 5.6: Shot from the back.

Bone Name	*Angle (deg)*	*Velocity (deg/sec)*
Bip01 R Thigh	90	-300
Bip01 R Calf	90	-300
Bip01 R Foot	90	-300
Bip01 R Toe0	90	-300

Table 5.7: Shot from the front.

There was no change in the transition parameters. If the relaxation was any shorter it would look like someone that was pulled gently by the leg. It must look like the muscles and sinews had experienced stress and need some

Bone Name	*Angle (deg)*	*Velocity (deg/sec)*
Bip01 R Thigh	0	-900
Bip01 R Calf	90	-600
Bip01 R Foot	90	-600
Bip01 R Toe0	90	-600

Table 5.8: Shot from the inside.

Bone Name	*Angle (deg)*	*Velocity (deg/sec)*
Bip01 R Thigh	0	900
Bip01 R Calf	90	600
Bip01 R Foot	90	600
Bip01 R Toe0	90	600

Table 5.9: Shot from the outside.

time to relax. Being shot in the leg only got a GM bad because of the lack of balance disturbance in the stance of the figure.

Attack (sec)	*Sustain (sec)*	*Relaxation (sec)*	*Sim Time (sec)*
0.1	0.1	2.5	0.15

Table 5.10: The transition parameters.

5.2.3 Test III - Walking and shot in chest, GM Good

The most obvious animation is the one of a person being shot in the chest area. Several things happen when you are shot in the chest, for example you could lose the air or be hit in the spine. The most obvious is that you bend in some direction with the upper body, and because of this the spine is a good candidate for affected bones. I included most spinal bones into the situation as seen in Figure 5.5. The motion looked convincing and lends very well to automation, since the velocities in the back and front are mirrored. Walking and shot in the chest got a GM Good.

Figure 5.4: Test II - Walking and shot in leg. The bones that are dynamically simulated have a black joint rhombus attached. The right leg is affected.

Bone Name	*Angle (deg)*	*Velocity (deg/sec)*
Bip01 Spine1	90	600
Bip01 Spine2	90	600
Bip01 Spine3	90	600

Table 5.11: Shot from the back.

Bone Name	*Angle (deg)*	*Velocity (deg/sec)*
Bip01 R Spine1	90	-600
Bip01 R Spine2	90	-600
Bip01 R Spine3	90	-600

Table 5.12: Shot from the front.

Attack (sec)	*Sustain (sec)*	*Relaxation (sec)*	*Sim Time (sec)*
0.1	0.1	2.5	0.15

Table 5.13: The transition parameters.

Figure 5.5: Test III - Walking and shot in chest. The bones that are dynamically simulated have a black joint rhombus attached. The spine bones are affected.

5.2.4 Tests IV-VI - Waving, GM Good except for Test V

Test I through III were also done on a waving animation to test that these situtations are adaptable to other animations. This was done by only changing what animation the dynamic simulator would use to initialise from and to mix with. No static or dynamic parameters were changed. The exact same results were seen in tests IV to VI as in I to III, which means that at least a waving animation is easily adaptable. This means that tests IV and VI got a GM Good while test V got a GM Bad, because it too had problems with balance in the legs. When someone is shot in the legs, he or she should lose their balance and not stand still or be unaffected. A picture of a waving man being shot in the soulder is seen in Figure 5.6.

5.3 Analysis

Let us analyse the results from the test phase, beginning with a discussion of the static parameters mass, dampening, angle and affected bones, then discussing the dynamic velocity. The mass parameter or radius was never changed from 10 meters, although it does have some significance. If for example a bone has a small mass parameter it will easily be affected by nearby bones. On the other hand when shooting characters mass is of minor

Figure 5.6: Test IV - Waving and shot in shoulder. The bones that are dynamically simulated have a black joint rhombus attached. The spine bones and the left arm are affected.

significance, since the dynamic events are brief. If one would want a bone to move faster the only remedy is to alter its velocity. If an impulse affects a bone with a greater mass, it will not have a high local velocity as a limb with less mass, if they have the same rotational plane. The problem is that if you want to emphasize a certain bone by decreasing its mass, it will be more willing to change direction because of its neighbours. That is why a velocity multiplier independent of the dynamics properties is needed. This velocity multiplier affects how much initial velocity a limb will have, before it is simulated.

An even less significant factor is the dampening, which regulates how much kinetic energy that dissipates. It is not a good approximation of a muscle, and to make it significant it would have to be greatly exaggerated. In the short events used in the tests, energy dissipation was not given enough time to make any difference. Strongly coupled to the velocity is the angle or vector of rotation about which the bone revolves. In my implementation I only used one degree of freedom per bone. Setting this single degree per bone was done very intuitively and gave great results. The problem was in limbs that have more that one inherent degree of freedom. This was solved by manipulating the vector to gain an extra degree of freedom. Freezing a degree of freedom is quite convenient if control is considered, as the less

freedom the limbs have the more control the animator is given. Too many degrees of freedom leads to another type of dynamics for humans, called ragdolling which is the opposite of parametric dynamics. The number of bones that are simulated controls the freedom of the simulation in conjunction with the rotational vectors. If one wants a shot in the arm, one activates these bones to be simulated. This gave good results and it was very intuitive to set the correct bones.

One of the most important parameters is the velocity which tells the dynamics system how fast the bones move. In the beginning I thought that I would have to take into account the initial velocity of the limbs from the kinetic animation. But since they are so small in comparison to the violent velocities of a dramatic shot, I disregarded them. The velocity is dependent on the dynamic event, and probably smaller velocities will make the simulated bones more sensitive to the initial velocities. However setting the velocity should be done dynamically by the event, and as commented in the tests this should be simple to automate since orthogonal velocites were used for orthogonal directions of interacting force.

The transition parameters are dependent on the type of dynamic event. In all the test I used shots, demanding a dramatic transition of events. Using a very short attack period followed by a short sustain and a long relaxation period the character looked as if it was shot. If the attack period was too long, it looked as if something pushed the figure rather than if something hit it. The sustain period was short, since the dynamics in itself is not suited to animate the figure, which is what happens when the blending is in the sustain period. A long relaxation made the character look convincing as it slowly relaxed back into the original animation. If the simulation time was too long the resulting animation looked very strange as the figure would get its limbs in unnatural positions, due to the high velocities involved. If a slower dynamic event would affect the figure, the simulation could run during a longer period.

The overall test results are given in Table 5.14. The Gestalt Metric was a good pruning tool to tune the parameters, since one could define what the end result should look like. Tests II and V were the bad ones and they are both shots in the leg. Since no animation was given GM Less Good, maybe the scale is too crude and needs some more refined criterias. Hopefully though some day the Gestalt Metric will not be used, as we can replace it with an automated model to test against.

All of the test ran in real-time on a Pentium II^{TM} 300 MHz with a Matrox Millenium^{TM} card at approximately 30 frames per second. Due to the experimental physics engine it took more processing power than necessary, as it can be highly optimized.

Test	*Gestalt Metric*
Test I	GM Good
Test II	GM Bad
Test III	GM Good
Test IV	GM Good
Test V	GM Bad
Test VI	GM Good

Table 5.14: The tests and their Gestalt Metrics.

Chapter 6

Further Work

The most obvious enhancement is a penetration system for dealing with limbs that intersect. Supporting tree structured chains of bones was to be added initially but was left out of time constraints. Tree structures are supported by the simulator already.

When I did the tests I was not thinking of how fast the limbs needed to go to be convincing. If the limbs are that fast such as 360 degrees per second or more, which is a whole turn per second, physical rules tend not to matter. A more light weight simulator could be used, such as every limb only rotates with its velocity disregarding the other limbs, and there would be no need for any integration of any sort. This special simulator would only be used for shots or other such dramatical and instant situations and would be very fast.

A detail I disregarded in the tests was how to initiate the velocities of the links, which would be done using a force transform from world force into local force, since all velocites are local. However most part of the work would not be on the computational side, but on the interface side to the animator.

A new parameter that was found during testing was the velocity multiplier, that would help to emphasize certain limbs.The dynamic situation has to be extended to also incorporate extra transitional animations such as a stumbling animation in the case of legs being shot at. Since I have had so many ideas of how to add features and new parameters to the parametric dynamics method, I think it will be used in some hybrid form.

Chapter 7

Conclusion

The resulting simulated animations were quite convincing and it was easy to parametrize them. There was a problem with intersection though in for example test II when the right leg cut into the left leg. Actually it was very fun to parametrize the different situations and test them, which means at least I found the system intuitive, but then again I designed and built it. It should however not be a problem for a skilled animator to use the system, obviously in a more refined form where the velocity parameters are dynamically computed.

The first time I began to simulate the shot in the shoulder I was excited as it looked so real. In the first few moments of a shock such as a bullet that hits a body, the limbs could very well be considered to be pendulums. For more complex interactions such as being shot in the leg merely accelerating the leg is not enough. Another complex pre-recorded animation has to be superimposed. This is now a practical problem, which means that the system has to be built, used and redesigned to fit special needs. I am quite certain that this method has at least brought many new ideas into the field of human animation in computer games, and will be used in some form. Personally I would use the system to add some life - or death which is more accurate - into the characters in a computer game. For example a man being shot can be shot from almost any angle and still look good.

The dynamic interactions that can be achieved using the method are numerous, but it needs a system for dealing with penetrations. Another problem that occurred when the animation was not correctly parametrized was that the bones could take the wrong shortest paths. This happened when a bone such as the spine bone was spun almost all the way around the hip, which made the quaternion interpolation take the shorter path between the legs instead of around the hip. However these extreme situations rarely happen, but if they do a second overriding system that checks legal angles

could be used.

In closing I would say that it is easy to tune the parameters, it is a simple and extendable technique, it can be used on several animations and fits perfectly into a bone blending pipeline. Parametric dynamics takes up very little memory and demands quite small computational costs.

Bibliography

[1] Uri M. Ascher, Dinesh K. Pai and Benoit P. Cloutier, "Forward Dynamics, Elimination Methods and Formulation Stiffness in Robot Simulation", Department of Computer Science, University of British Columbia, Vancouver, Canada, pp. 1-17, May 1996

[2] Amaury Aubel, Boulic Ronan and Daniel Thalmann, "Real-Time Display of Virtual Humans: Levels of Details and Impostors", IEEE Nbr 1051-8215/00, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 10, No. 2, pp. 207-217, March 2000

[3] David Baraff, "Physically Based Modeling: Principles and Practice, Implicit Methods for Differential Equations", SIGGRAPH '97 Course notes, pp. E1-E4, 1997

[4] David Baraff, "Rigid Body Simulation I - Unconstrained Rigid Body Dynamics", SIGGRAPH '97 Course notes, pp. D1-D31, 1997

[5] David Baraff, "Rigid Body Simulation II - Nonpenetration constraints", SIGGRAPH '97 Course notes, pp. D32-D68, 1997

[6] Alan H. Barr, Bena Currin and Steven Gabriel, John F. Hughes, "Smooth Interpolation of Orientations with Angular Velocity Constraints using Quaternions", SIGGRAPH '92, 1992

[7] Shih-kai Chung and James K. Hahn, "Animation of Human Walking in Virtual Environments", pp. 4-15, IEEE nbr 1087-4844/99, 1999

[8] John J. Craig, "Introduction to Robotics: Mechanics and Control", Addison-Wesley, 1989

[9] Darwin3D, http://www.darwin3d.com, home page of Jeff Lander's company Darwin3D

[10] Discreet, "Exporting Character Studio R2 Biped Information from 3D Studio MAX R2.X", 3D Studio MAX R3 - Character Studio help files release 2.0, 2000

[11] Hugo Elias, "Strings", "Cloths", "Gel", The good-looking textured light-sourced bouncy fun smart and stretchy page, http://freespace.virgin.net/hugo.elias/, 2000

[12] Roy Featherstone, "The Calculation of Robot Dynamics Using Articulated-Body Inertias", International Journal of Robotics Research, pp. 13-30, Spring 1983

[13] Gamasutra, http://www.gamasutra.com, Web site for the magazine Game Developer Magazine, published by Miller Freeman, 2001

[14] Andrew S. Glassner, "Graphics Gems", Edited by Andrew S. Glassner, Academic Press, 1990

[15] Granny SDK, "Granny 3D Animation", http://www.smacker.com/, 2000

[16] Havok Engine, http://www.havok.com, Web site for the Havok physics programming library, owned by Havok, 2001

[17] Chris Hecker, "Physics, The Next Frontier", Game Developer, Miller Freeman, pp. 12-20, October/November 1996

[18] Chris Hecker, "Physics, Part 2: Angular Effects", Game Developer, Miller Freeman, pp. 14-22, December 1996/January 1997

[19] Chris Hecker, "Physics, Part 3: Collision Response", Game Developer, Miller Freeman, pp. 11-18, March 1997

[20] Chris Hecker, "Physics, Part 4: The Third Dimension", Game Developer, Miller Freeman, pp. 15-25, June 1997

[21] David Herman, "A Renaissance robot", Mechanical Egineering, The American Society of Mechanical Engineers, 1998

[22] Andrés Jaramillo-Botero, "Rigid Multibody Molecular Dynamics: Strictly Parallel Computations", Internal report No. GAR1997-05, Pontificia Universidad Javeriana, Cali, Colombia, 1997

[23] Daniel Jeppsson, "Realtime Character Animation Blending Using Weighted Skeleton Hierarchies", Master Thesis in Computer Science and Engineering, Lund Institute of Technology, Lund, Sweden, 2000

[24] Prem Karla, Nadia Magnenat Thalmann, Laurent Moccozet and Gaël Sannier, "Real-time animation of realistic virtual humans", IEEE Computer Graphics and Applications, Vol.18, No.5, pp. 42-55, 1998

[25] D. H. Kochanek and R. H. Bartels, "Interpolating Splines with Local Tension, Continuity, and Bias Control," Computer Graphics, vol. 18, no. 3, pp. 33-41, July 1984

[26] Evangelos Kokkevis, Dimitri Metaxas and Norman I. Badler, "User-Controlled Physics-Based Animation for Articulated Figures", Proceedings of Computer Animation '96, 1996

[27] Jeff Lander, "Working With Motion Capture File Formats", Game Developer, Miller Freeman, pp. 30-37, January 1998,

[28] Jeff Lander, "Oh My God I Inverted Kine", Game Developer, Miller Freeman, pp. 9-12, September 1998

[29] Jeff Lander, "Making Kine More Flexible", Game Developer, Miller Freeman, pp. 15-20, November 1998

[30] Jeff Lander, "Skin Them Bones: Game Programming for the Web Generation", Game Developer, Miller Freeman, pp. 11-14, May 1998

[31] Jeff Lander, "Physics on the Back of a Cocktail Napkin", Game Developer, Miller Freeman, September 1999

[32] Jeff Lander, "Collision Response: Bouncy, Trouncy, Fun", Game Developer, Miller Freeman, March 1999

[33] Jeff Lander, "Lone Game Developer Battles Physics Simulator", Game Developer, Miller Freeman, April 1999

[34] Jeff Lander, "Trials and Tribulations of Tribology", Game Developer, Miller Freeman, August 1999

[35] Jeff Lander, "Crashing Into the New Year", Game Developer, Miller Freeman, January 1999

[36] Jehee Lee and Sung Yong Shin, "A Hierachical Approach to Interactive Motion Editing for Human-like Figures", ACM Nbr 0-201-48560-5/99/08, SIGGRAPH '99, pp. 39-48, 1999

[37] J. P. Lewis, Matt Cordner and Nickson Fong, "Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation", SIGGRAPH 2000, pp. 165-172, 2000

[38] Janzen Lo and Dimitris Metaxas, "Recursive Dynamics and Optimal Control Techniques for Human Motion Planning", IEEE Nbr 1087-4844/99, pp. 220-234, 1999

[39] MathEngine, http://www.mathengine.com, Web site for the physics programming library MathEngine, owned by MathEngine plc, 2001

[40] J. L. Meriam and L. G. Kraige, "Engineering Mechanics Volume 2: Dynamics", John Wiley and Sons Inc., New York, 1986

[41] Tomas Möller and Eric Haines, "Real-Time Rendering", A K Peters Ltd., 1999

[42] Brian Vincent Mirtich, "Impulse-based Dynamics Simulation of Rigid Body Systems", PhD dissertation at the University of California at Berkeley, 1996

[43] Yoshihiko Nakamura and Katsu Yamane, "Dynamics Computation of Structure-Varying Kinematic Chains and Its Application to Human Figures", IEEE Transactions on Robotics and Automation, Vol. 16, No. 2, April 2000

[44] Dinesh K. Pai, Uri M. Ascher and Paul G. Kry, "Forward Dynamics Algorithms for Multibody Chains and Contact", Department of Computer Science, University of British Columbia, Vancouver, Canada, 2000

[45] Zoran Popović, "Controlling Physics in Realistic Character Animation", Communications of the ACM, Vol. 43, No. 7, pp. 50-58, July 2000

[46] Luciana Porcher Nedel and Daniel Thalmann, "Real Time Muscle Deformations Using Mass-Spring Systems", Computer Graphics Lab, Swiss Federal Institute of Technology, EPFL, 2000

[47] Jerry E. Pratt and Gill A. Pratt, "Exploiting Natural Dynamics in the Control of a 3D Bipedal Walking Simulation", International Conference on Climbing and Walking Robots (CLAWAR99), Portsmouth, UK, September 1999

[48] Paul R. P., "Robot Manipulators: Mathematics, Programming and Control", MIT Press, Cambridge, MA, 1981

[49] G. Rodriguez, A. Jain and K. Kreutz-Delgado, "A Spatial Operator Algebra for Manipulator Modeling and Control", The International Journal of Robotics Research, pp. 371-381, 1991

[50] D. F. Rogers and J. A. Adams, "Mathematical Elements for Computer Graphics", Macgraw-Hill, New York, 1976

[51] Gaël Sannier, Selim Balcisoy, Nadia Magnenat Thalmann and Daniel Thalmann, "VHD: a system for directing real-time virtual actors", The visual Computer, Springer, Vol.15, No 7/8, pp. 320-329, 1999

[52] I. Sharf and G. M. T. D'Eleuterio, "Computer Simulation of Elastic Chains Using a Recursive Formulation", IEEE nbr CH2555-1/88/0000/1539, pp. 1539-1545, 1988

[53] Ken Shoemake, "Animating Rotation with Quaternion Curves", SIGGRAPH '85, pp. 245-254, 1985

[54] Robert W. Soutas-Little, "Motion Analysis and Biomechanics", Michigan State University in East Lansing, Michigan, 1999

[55] Nadia Magnenat Thalmann and Daniel Thalmann, "Computer Animation in Future Technologies", Computer Graphics Lab, EPFL, MIRALab, 1998

[56] Nadia Magnenat Thalmann and Daniel Thalmann, "State-of-TheArt in Computer Animation", Computer Graphics Lab, EPFL, MIRALab, 1998

[57] Russel Turner, "Interactive Construction and Animation of Layered Elastic Characters", PhD dissertation in Computer Science at École Polytechnique Fédérale de Lausanne, 1993

[58] M. W. Walker and D. E. Orin, "Efficient Dynamics Computer Simulation of Robotic Mechanisms", ASME J. Dynamics Systems, Measurement and Control, pp. 205-211, 1982

[59] Allen Watt, "3D Computer Graphics", Addison-Wesley, 2000

[60] Chris Welman, "Inverse Kinematics and Geometric Constraints for articulated figure manipulation", Master Thesis in Computer Science at the Simon Frasier University, September 1993

[61] Andrew Witkin and David Baraff, "Physically Based Modeling: Principles and Practice, Differential Equation Basics", SIGGRAPH '97 Course notes, 1997

[62] Andrew Witkin, "Physically Based Modeling: Principles and Practice, Particle System Dynamics", SIGGRAPH '97 Course notes, 1997

[63] Andrew Witkin, "Physically Based Modeling: Principles and Practice, Constrained Dynamics", SIGGRAPH '97 Course notes, 1997

[64] W. A Wolovich and H. Elliot, "A Computational Technique for Inverse Kinematics", in Proceedings of 23rd Conference on Decision and Control, pp. 1359-1362, 1984

[65] Mason Woo, Jackie Neider, Tom Davis and Dave Shreiner, "OpenGL Programming Guide", The Official Guide To Learning OpenGL, Version 1.2, Third Edition, Addison Wesley, 1999

[66] Hugh D. Young, "University Physics", Addison-Wesley, Eight edition, 1992