

# **Rotation Points From Motion Capture Data Using a Closed Form Solution**

Dissertation  
for  
Ph.D. in Computer Science  
at  
University of Colorado, Colorado Springs

by  
Jonathan Kipling Knight  
2 May 2005  
(M.A. Applied Mathematics, Cal. State Fullerton, 1995)  
(B.S. Physics, Cal Poly, San Luis Obispo, 1987)

Directed by Dr. Sudhanshu Kumar Semwal

## **Abstract**

A skeleton composed of rotation points is usually made to fit motion capture data by guessing the size and iterating until the model fits the motion. This involves a few assumptions and approximations and sometimes produces more than one answer. This dissertation presents a purely mathematical solution that has only one answer. This new closed-form solution is robust in noisy and missing motion capture data. The Minimum Variance Method produces the same accuracy as linear and non-linear Least-Squares methods of fitting spheres, cylinders, circles and planes to generic data without the initial guessing. The method significantly reduces the amount of work needed for calculating a rotation point by only requiring  $O(N)$  averaging of the data and one inversion of a  $3 \times 3$  positive-definite matrix for any data-set. The same matrix can be used to compensate for cylindrical motion inaccuracies. This method aids the realism of motion data animation by allowing the subtle nuances of human motion to be displayed. The calculations can be reused for the same actor and marker-set allowing different data sets to be blended.

Dedicated to my wife Kiki for her patience.

© Copyright by Jonathan Kipling Knight 2005, All Rights Reserved

This thesis for the degree of Philosophical Doctor by

Jonathan Kipling Knight

has been approved for the

Department of Computer Science

by

---

Sudhanshu Kumar Semwal, Chair

---

Charles M. Schub

---

C. Edward Chow

---

Bob Carlson

---

Jugal K. Kalita

---

Date

## Table of Contents

<b>1.Introduction.....</b>	<b>1</b>
1.1. Motivation.....	1
1.2. History.....	4
1.3. Realism.....	6
1.4. Real-time.....	6
1.5. Paper Outline.....	6
1.6. Terminology.....	7
<b>2.Motion Model.....</b>	<b>8</b>
2.1. Closed-Form Rotation Point Determination.....	8
<i>Theory</i> .....	9
<i>Pseudo-code</i> .....	13
<i>Examples</i> .....	14
<i>Results</i> .....	20
<i>Limitations</i> .....	22
2.2. Segment Coordinate System.....	23
2.3. Motion Capture Data.....	26
<i>Capturing Data</i> .....	26
<i>Data Format</i> .....	27
<i>Correlating Data to Segments</i> .....	29
2.4. Motion Algorithm.....	30
<i>Pseudo-code</i> .....	31

<b>3.Articulated Model.....</b>	<b>34</b>
3.1. Tetrahedron.....	35
3.2. Tetrahedral Mesh.....	37
3.3. Rigid Body.....	38
3.4. Segment.....	39
3.5. Figure.....	39
<i>Converting Triangulated Surface into Tetrahedral Solid.....</i>	<i>42</i>
<b>4.Programming Model.....</b>	<b>45</b>
4.1. Articulated Figure.....	45
<b>5.Products.....</b>	<b>47</b>
5.1. Walking Human Figure.....	47
5.2. Rolling On Floor.....	48
5.3. Break-dancing.....	49
5.4. Salsa Dance.....	50
<b>6.Conclusion.....</b>	<b>52</b>
<b>7.Bibliography.....</b>	<b>54</b>
<b>8.Appendix.....</b>	<b>60</b>
8.1. Taylor Array Size.....	60
8.2. Inertial Properties of a Tetrahedron.....	61
8.3. Inertial Properties of a Rigid Body.....	64
8.4. User's Guide for Macintosh Application.....	66

	<i>Menu</i> .....	67
	<i>Options Pane</i> .....	69
	<i>Animation Pane</i> .....	71
	<i>Graphs Pane</i> .....	72
8.5.	Specification for Articulated Tetrahedral Model Format.....	73
8.6.	Proof of Positive-Definite Matrix for Rotation Point.....	75
8.7.	Relevant C++ Implementations.....	76
	<i>Minimum Variance Method</i> .....	76
	<i>Rotation Point Calculation of Hierarchical Articulated Data</i> .....	77
	<i>Calculation of fixed axes of data</i> .....	79
	<i>Drawing Rotation Points with Constants of Motion</i> .....	80
8.8.	Kalman Filter.....	82
	<i>Theory</i> .....	82

# 1 Introduction

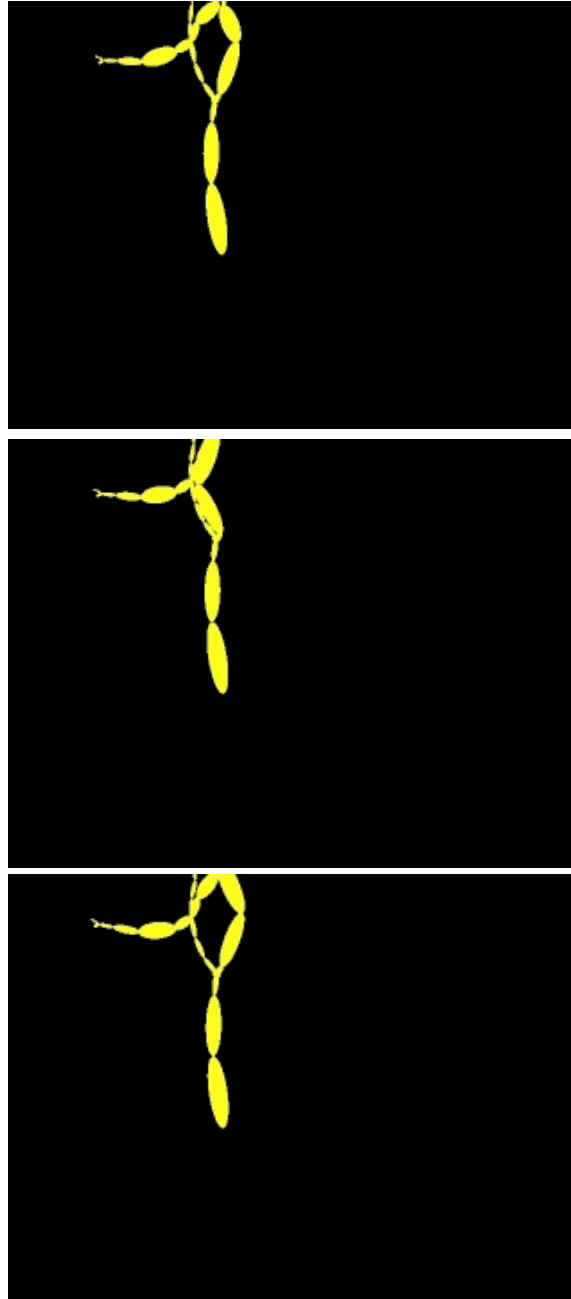
## 1.1 Motivation

Through the research into articulated motion, one fundamental problem arose which existing method produced unsatisfactory results. The problem was to find a skeleton under a data-set. Judgment is made by having the method satisfy the simple criteria:

1. Quick
2. No a-priori knowledge of skeleton
3. Generalized to any shape of articulated model

Many authors use Least-Squares techniques to fit a pre-existing skeleton into the data by “squishing” and adjustment joints until it fits the data. This is generally termed as inverse kinematics. O’Brien (2000) produced a global optimization of least-squares analysis of a skeleton. They claim real-time calculations but they are using a-priori knowledge of a human shaped model to fit into the data. Their method, and many others, involve constraints on the skeleton which include being symmetric human shapes. Other constraints on the model must be enforced for Inverse Kinematics because there are cases where the problem is severely under-constrained. When a problem is under-constrained, multiple solutions exist and artificial constraints must be introduced. A good example of this problem is when a human wants to place his hand on the table. Carnegie-Melon University (CMU) Graphics Lab produced a similar sequence (labeled 05-05) using a ballerina outstretching her leg. Once outstretched, the knee joint flopped between two possible solutions in three consecutive time frames:

## Knight 2



This is a well studied problem with various authors coming up with appropriate constraints. But what if there was a method that did not need to know the skeleton beforehand and did not have to guess at constraints and initial conditions? Various attempts were tried in order to obtain this goal. Originally, genetic algorithms combined with kinetics were tried and failed due to the slow nature of the genetic algorithm. Next, Least-Squares fitting was tried but sometimes did not converge to the right answer (the



initial guess was not close enough). The Kalman filter was tried next but had the tendency to converge to a biased answer or none at all. Finally, after stepping back in frustration, the Minimum Variance equation was discovered which produced a correct answer all of the time, excluding non-moving joints. The method does not fit into the mold of either Kinetic or Kinematic methods but is a purely closed-form mathematical solution of the rotation points of the skeleton inside. It satisfies the self-imposed criteria of quick  $O(N)$ ; has absolutely no a-priori knowledge of an articulated figure except for a simple association of a marker with a segment; can conform to the exact form of the underlying actor, whether it be deformed, animal, human or robot: as long as it has articulated non-translational joints.

Articulated motion is described as connected solid segments moving as a whole. Examples of such bodies are the human body, insects, and robots. The approach in this research is to have a local view of a segment's motion. Each segment will have control of its motion relative to its more proximal segment (e.g. the hand controls the wrist motion). This produces a hierarchy of segments starting from the root (hips) and moving outward to the end effectors. The hierarchy produces a set of constants that determine the rotation point. The constants include the position of the rotation point relative to the parent's fixed reference frame and the null vector of the joint, which is usually the one axis in a cylindrical joint. Once the constants of motion are found for the motion capture data each segment will store for determining the absolute position of its rotation point at a later time frame.

## 1.2 History

Motion capture animation has been continuously improved by many authors<sup>1</sup>. Their contributions can be divided into two methods of analysis: kinematic and kinetic<sup>2</sup>. In kinematic methods, scientists study the mechanical displacements of the limbs during motion. In kinetic methods, the energies and forces on the limbs are studied during the motion of the articulated figure leading to dynamical formulation. This new method does not fall in either of these categories.

Kinematic methods are used in animation by determining the joint angles from space-time constraints. Holt et al. (1997) estimated the 3D motion of an articulated object from a sequence of 2D perspective views. They used a decomposition approach to break down the motion of each segment. This was a good use of video motion capture to estimate the animation of a figure. Inverse Kinematics (IK) is the backwards use of positioning to determine which angles are necessary to get from one posture to the next. IK is the more popular method to determine motion. Grochow et al. (2004) have a fairly complete system building on the IK method. Their method compensates for the multiple possible poses by a probabilistic model based on previously known styles of poses.

Kinetic methods are used in animation by analyzing the changes in energy, inertia, or forces involved in the motion. These values determine the way the joint angles change in time. Kinetic methods had their start in the stick and/or block figures of Jensen, et al. in 1977<sup>3</sup>. Jensen produced a system for interactive computer modeling of the

---

<sup>1</sup> Grochow 2004

<sup>2</sup> Aggarwal 1998

<sup>3</sup> Semwal 1999

musculoskeletal system. A good example of this physics based motion analysis can be seen in Semwal et al. (1999) as well. Their method allows a cyclist to visualize the leg rotations and forces involved in the pedal movement. A straightforward physics approach is to solve for the equation of motion whether it be Newtonian mechanics or Lagrangian mechanics. This approach is not easily computed, usually by time consuming iterative methods. The method involves solving simultaneous second order partial differential equation. These complications make them not used very often. Results are mediocre in both quality of animation and controlling the figure. A more efficient attitude to solving these equations is using recursive methods<sup>4</sup>. Liu and Popovic (2002) presented a SIGGRAPH paper explaining a novel method of infusing physical reality into sparsely keyed motion data. They presented an articulated figure that realistically played hopscotch from a minimal set of predetermined positions. Their method involved a figure composed of ellipsoids. The key-frames would be set up by the animator and the method could correct the positions to follow physical reality. Liu and Popovic's method produced wonderful motions but relied on off-line calculations. The method presented in this dissertation is an "on-line" method in which there is virtually no work involved in following motion capture data.

Popovic and Witkin (1999) presented another novel idea to transform standard models of motion into a diverse assortment of similar motions. For instance, their method could take a standard run sequence and transform it into a run with a limp sequence, while retaining physical reality. This method relies on a library of standard motions that contain every motion regime that may be transformed.

---

<sup>4</sup> Znamenacek 1998

### **1.3 Realism**

This research is concerned with the realism as perceived by the viewer of the animation. If the viewer perceives realistic motion then the motion is classified as satisfactory. This is also known as the classic Turing test. Only a skeleton based on joints is used so the realism relates to the algorithm producing the motion. The generated motion has been presented to an audience who has agreed that the motion is better than the same data-set analyzed by CMU Graphics Lab using inverse kinematics. It is the author's opinion that the human's ability to identify subtleties in motion is the determining factor of realism. The Minimum Variance Method allows for the subtleties to still be displayed.

### **1.4 Real-time**

The "real-time" phrase, as used in this research, relates to the aspects of the motion algorithm. The algorithm is as fast as analyzing only ten data points for each segment and then reusing the constants in all time frames of the data. The constants that determine the rotation point of each segment can be reused so long as the data came from the same actor wearing the same non-moved markers. Not much more speed can be achieved without a-priori knowledge.

### **1.5 Paper Outline**

This paper is structured as follows. Section 1 contains the introduction to the research topic. In Section 2, the motion model is derived. The articulated figure model is presented in Section 3. The programming model follows in Section 4. Section 5 contains

the products that have come out of the research. The conclusion is in Section 6. The bibliography is in Section 7. The appendix contains various proofs and mathematical details that have been derived as well as a user's guide to the application that was developed for the research.

## 1.6 Terminology

Some terms have been borrowed from biology. Distal is a description of a part of a body that is closer to the peripherals (e.g. finger is distal to the elbow). Proximal is a part that is closer to the central body. In this paper, these terms are used to describe parts of the body that are placed in a tree structure representing the articulated figure. There is always a root to the tree, which has nothing proximal. A parent segment is proximal to its children. As explained in the next chapter, a segment's proximity to the central body is not strictly the same as the position in segment tree hierarchy. Throughout the paper, vector notation is used and a caret (^) symbolizes that the vector with a caret has been normalized to have a length of unity. Also, during matrix algebra, a vector is considered to be a column vector and a row vector is a column vector explicitly transposed ( $T$ ).

## 2 Motion Model

This chapter sets forth the method used to turn an uncorrelated set of data points into a moving human figure. The first question that must be answer is where to put the skeleton. The skeleton is a set of rotation points with lines drawn in between. So where are the rotation points relative to the surface data points in the data? The original motion capture data does not come with any explicit information about the size and shape and rotations of the actor doing the motion. The method presented next solves the local minimization solution for a single rotation point in the articulated figure. The follow-up question to answer is how to get a fixed coordinate frame relative to each segment in articulated figure under the data points. Putting the two answers to these questions together is what allows this motion model to animate a skeleton through direct calculations from the raw data.

### 2.1 Closed-Form Rotation Point Determination

A more successful solution than previous method was discovered during course of this research. It was found that a closed-form solution exists when the variance of the square of the distance from the measurement to the point of rotation is minimized. Joint rotations cannot be determined unless the point of rotation is determined. Most data does not come with this information. Data comes in the form of  $(t,x,y,z)$  for a particular point on a figure during its motion in absolute coordinates. Previous solutions<sup>5,6,7</sup> to determine

---

<sup>5</sup> O'Brien 2000

<sup>6</sup> Bodenheimer 1997

<sup>7</sup> Herda 2000

this point involve iterations on a least-squares equation or M-estimators starting from an initial guess. This approach involves either linear or non-linear fitting of the data and has a chance of not converging to a solution. The initial guess must be close enough to the truth or the iterations may diverge away from the point. This chapter presents a new closed-form solution to the center of rotation that involves no guessing and just the inversion of a 3x3 matrix. The Minimum Variance Method is robust with noise and also works with cylindrical motion, with a little extra work. The method is a suitable replacement for linear and non-linear least-squares fitting of a sphere, cylinder, circle, and a plane. The requirements in order to determine the point of rotation are as follows:

- 1) Fixed axes relative to the point of rotation.
- 2) At least 4 points far enough apart.

The problem amounts to finding the best-fit sphere for a 2 or 3 DOF joint or the best-fit cylinder for a 1 DOF joint. According to the NIST,<sup>8</sup> the best approach to this problem is non-linear Least-Squares fitting. The Minimum Variance Method eliminates the guesswork involved in least-squares fitting and produces an immediate answer.

### ***2.1.1 Theory***

Presented here is a closed form solution that has time complexity of  $O(N)$  and involves the inversion of a 3x3 positive-definite matrix. Stated “simply”, the closed form solution involves solving for the absolute minimum of the variance of the square of the lengths from the point of rotation to each point in the data. A point ( $p_i$ ) around a rotation point ( $r$ ) can be represented as

---

<sup>8</sup> Shakarji 1998

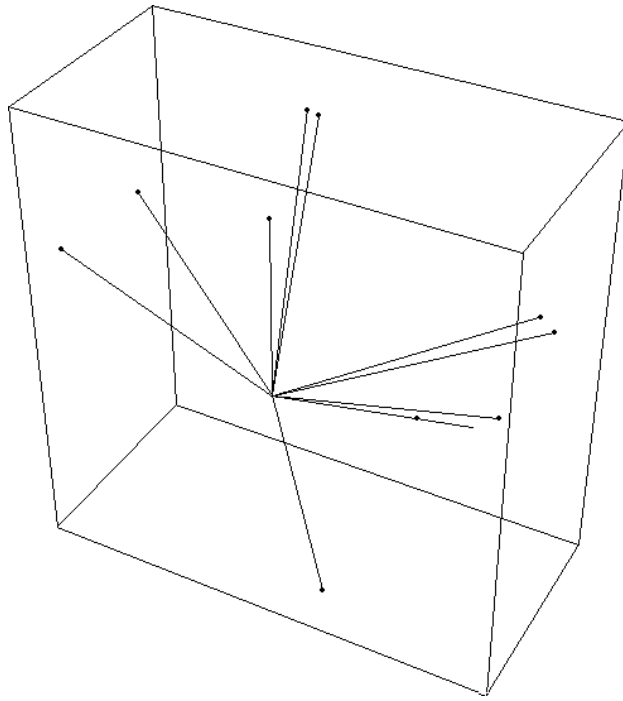
Knight 10

$$p_i = r + L_i \rho_i$$

where

$$\|\rho_i\| = 1$$

and  $L_i$  is the distance to  $p_i$  from  $r$



The standard definition of the sampled variance of the square of the distance is

$$s^2 \equiv \text{Var}(L^2) = \frac{N}{N-1} (\overline{L^4} - \overline{L^2}^2)$$

where

$$L_i = \|p_i - r\|$$

$$\overline{L^k} = \frac{1}{N} \sum \|p_i - r\|^k$$

The formula to solve after collecting the data is when the gradient of the variance is set to zero.

**Equation 2.1.1**  $\nabla s^2 = 0$  where



$$\nabla \equiv \left( \frac{\partial}{\partial x} \frac{\partial}{\partial y} \frac{\partial}{\partial z} \right)^T$$

Taking the gradient derivative of the variance produces

$$\nabla s^2 = \frac{N}{N-1} \left( \nabla \bar{L}^4 - 2\bar{L}^2 \nabla \bar{L}^2 \right)$$

The gradient of the mean powers of  $L$  are

$$\nabla \bar{L}^2 = 2(r - \bar{p}) \text{ and } \nabla \bar{L}^4 = \frac{4}{N} \sum L_i^2 (r - p_i)$$

and

$$\bar{L}^2 = \bar{p}^2 - 2\bar{p}^T r + r^2$$

Substituting these in produces

$$\nabla s^2 = \frac{N}{N-1} \left( 4\bar{L}^2 r - \frac{4}{N} \sum L_i^2 p_i - 4\bar{L}^2 (r - \bar{p}) \right)$$

Further simplification produces this

$$\nabla s^2 = \frac{4}{N-1} \left( 2 \sum p_i (p_i^T - \bar{p}^T) r - \sum p_i (p_i^2 - \bar{p}^2) \right)$$

$$\nabla s^2 = 8(Ar - b)$$

Setting this to zero ends up being a simple 3x3 linear equation to solve

**Equation 2.1.2**  $Ar = b$

where

**Equation 2.1.3**  $A = \frac{1}{N-1} \sum (p_i - \bar{p}) p_i^T$

**Equation 2.1.4**  $b = \frac{1}{2(N-1)} \sum (p_i - \bar{p}) p_i^T p_i$

Knight 12

The notation used here is such that

$$pp^T = \begin{pmatrix} x \\ y \\ z \end{pmatrix} (x \ y \ z) = \begin{pmatrix} xx & xy & xz \\ yx & yy & yz \\ zx & zy & zz \end{pmatrix}$$

$$p^T p = (x \ y \ z) \begin{pmatrix} x \\ y \\ z \end{pmatrix} = x^2 + y^2 + z^2$$

and the mean of the data points is

$$\bar{p} = \frac{1}{N} \sum p_i$$

The Minimum Variance Matrix  $A$  is positive-definite (cf. Appendix) so Cholesky decomposition can be used for a more efficient solution to the equation. There are two exceptions to this statement. The trivial case is if all points coincide,  $A=0$ . The non-trivial case is during planar motion. If there exists a vector  $n$  such that

$$p_i^T n = \text{constant}$$

then  $A=0$ . All is not lost though if  $A$  is near singular. Some mathematical concepts must be explained in order to continue solving the rotation point of planar data.

The Null Space of a matrix is a set of vectors that solve the equation  $An=0$ . This set of vectors is inherently extracted during the Singular Value Decomposition<sup>9</sup> (SVD) of any matrix based on some threshold. The condition number of a matrix is a measure that increases to infinity as a matrix becomes closer to singular. It too is extracted during SVD. If the threshold is equal to the inverse of the condition number then a single vector exists in the Null Space of  $A$  and that vector happens to be the normal to the plane of

---

<sup>9</sup> Press 1992

motion and is the best-fit plane for the data. This is proven by solving for the minimum of the variance of the distance from the plane.

$$p_i^T n = z_i$$

The variance of the distance  $z_i$  is

$$\text{Var}(z) = \frac{N}{N-1} \left( \overline{z^2} - \bar{z}^2 \right)$$

and the gradient of the variance is

**Equation 2.1.5**  $\nabla \text{Var}(z) = 2An$

Setting the gradient to zero will produce the minimum variance of  $z$ . The only solution to this equation is the Null Space of the Minimum Variance Matrix  $A$ .

Another interesting property for this equation comes from the fact that the double derivative matrix (Hessian) is

**Equation 2.1.6**  $\text{Hessian}(s^2) \equiv \frac{1}{2} \nabla \nabla^T s^2 = 4A$

The Hessian is positive-definite because  $A$  is positive-definite (cf. Appendix). If the Hessian of the function is positive-definite then the solution found is an absolute minimum. This says that the solution found for the rotation point is the absolute best to minimize the variance of the square of the lengths.

## 2.1.2 Pseudo-code

The procedure to determine point of rotation is as follows:

### Procedure 2.1.1

Choose one set of points on segment.

Make points relative to parent.

$M$  = column matrix of parent coordinate axes

Knight 14

$c$  = center of parent's coordinate frame

$$p_i = M^T(p_i - c)$$

Calculate variance of points.

$$\bar{p} = \frac{1}{N} \sum p_i$$

$$\overline{p^2} = \frac{1}{N} \sum p_i^T p_i$$

$$\text{Var}(p) = \frac{N}{N-1} (\overline{p^2} - \bar{p}^2)$$

if variance of points is large then  
use spherical formula.

$$A = \frac{1}{N-1} \sum (p_i - \bar{p}) p_i^T$$

$$b = \frac{1}{2(N-1)} \sum (p_i - \bar{p}) p_i^T p_i$$

solve  $Ar_0 = b$

if condition number of  $A$  is large then  
use cylindrical formula.

solve  $An = 0$  ( $n$  is cylinder axis or Null vector)

return  $r_1 = r_0 + nn^T(\bar{p} - r_0)$

else

return  $r_0$

end

### 2.1.3 Examples

#### Example #1 - Spherical Joint

Results from a-priori data are comparable to linear or non-linear least-squares fitting. Spherical motion data is produced below with added noise. The points are in the following table. The center of the sphere is at  $(0.6 \ -0.2 \ 0.9)^T$  with a radius of 1.2. Standard deviation of the noise of each point is 0.01.

<b>x</b>	<b>y</b>	<b>z</b>
1.71641	0.0532489	0.534942
-0.257165	-0.984895	0.59321
-0.0154738	0.0295651	1.90681

1.19134	-0.00156902	1.9146
1.31147	0.607414	0.374474
1.2274	-0.0905109	1.91278
1.62875	-0.631279	0.507247
0.367905	0.908098	0.472217
1.74362	-0.526356	0.621037
0.510973	-1.29795	0.495898

The points in the table produce the Minimum Variance Matrix:

$$A = \begin{pmatrix} 0.540464 & 0.0639822 & -0.0671881 \\ 0.0639822 & 0.462403 & 0.0550300 \\ -0.0671881 & 0.055030 & 0.459960 \end{pmatrix}$$

$$b = \begin{pmatrix} 0.251491 \\ 0.00024245 \\ 0.362261 \end{pmatrix}$$

Calculated results are as follows:

<b>Method</b>	<b>x</b>	<b>y</b>	<b>z</b>	<b>L</b>	<b>error</b>
Original	0.6	-0.2	0.9	1.2	0
Min Var	0.599337	-0.189249	0.897781	1.19811	0.0109972
Least Squares	0.599084	-0.189018	0.89785	1.19818	0.0112276

The results from above tell how close the answer is for Levenberg-Marquardt non-linear Least-Squares fitting and the Minimum Variance Method. Using the Singular Value Decomposition of A, the condition number for A is determined as 1.63852. A spherical solution is assumed better than cylindrical since the condition number is not

Knight 16

large. As the results show, a spherical solution from Minimum Variance or from Non-linear Least-Squares give approximately the same answer.

### Example #2 - Cylindrical Joint

An example for cylindrical data shows similar good results. The data is produced from a circle at the same rotation point and radius and normal vector of  $(0.1 \ 0.2 \ 0.974679)^T$ . The same amount of noise is introduced (0.01).

<b>x</b>	<b>y</b>	<b>z</b>
0.965464	0.896476	0.641555
1.68157	0.248753	0.703445
1.61990	-0.871485	0.930594
-0.592497	-0.0747179	1.01050
-0.402654	-0.818748	1.12976
-0.0912121	-1.14638	1.16056
0.117659	-1.26534	1.16835
-0.569521	-0.403927	1.06119
1.69910	-0.702110	0.889177
1.67226	-0.724973	0.891406

The Minimum Variance equation for this data becomes

$$A = \begin{pmatrix} 1.02288 & 0.110289 & -0.129224 \\ 0.110289 & 0.448177 & -0.100884 \\ -0.129224 & -0.100884 & 0.0336710 \end{pmatrix}$$

$$b = \begin{pmatrix} 0.475661 \\ -0.119367 \\ -0.0260505 \end{pmatrix}$$

Calculated results are as follows:

Method	x	y	z	L	error
Original r	0.6	-0.2	0.9	1.2	0
Min Var	0.700154	-0.0243642	1.8404	1.534	0.961886
Least-Squares	0.60523	-0.204356	0.936388	1.19704	0.0370193
Original Null	0.1	0.2	0.974679	N/A	0
Min Var Null	-0.102284	-0.194435	-0.975568	N/A	0.0060804
Least-Squares Null	-0.102284	-0.194435	-0.975568	N/A	0.0060804
Min Var new r	0.60204	-0.210872	0.904606	1.19708	0.0119822

The results of this example show, once again, almost the same answer as non-linear least-squares fitting. The condition number of this example's matrix is 67184.8 which tells us the matrix is nearly singular. Knowing this, additional work must be done in the Minimum Variance method. Using the cylindrical formula (projecting onto the mean plane), a better answer than the least-squares answer is found.

### Example #3 - Case study of CMU Data 60-08

The CMU Graphics Lab produced a one minute long motion capture data-set of a salsa dance in 60-08. The data file contains 3421 time slices for 41 markers on two figures. This case study will concentrate on analyzing the performance of the Minimum Variance Method in determining the rotation points in the female subject. Four passes on the data will collect rotation point calculations, each pass randomly removing from 0 to 99% of the time frames in increments of 1%. 400 calculated rotation points were

Knight 18

collected for each segment modeled. The calculated constants are the relative rotation points as referenced in each segment's parent's coordinate system. The 400 calculations were averaged and the standard deviations were calculated as well. These values are presented in the tables below.

<b>Rotation Point</b>	<b>Mean x (m)</b>	<b>Mean y (m)</b>	<b>Mean z (m)</b>
Waist	0.13324824	-0.063898286	0.130439024
Neck	-0.077279042	-0.003811468	-0.010261576
Left Ankle	0.437510805	-0.032639212	0.040255145
Left Wrist	0.110981565	-0.075779216	0.014702334
Left Elbow	0.283393628	-0.060979142	0.012893845
Left Knee	-0.244076283	-0.07952933	0.009066338
Right Elbow	0.253621623	-0.146993545	-0.001828167
Right Knee	0.188950453	-0.080176833	-0.003163181
Right Ankle	0.241527156	-0.065156728	0.006255086
Right Wrist	0.211446183	-0.069707086	-0.019905695
Left Shoulder	0.01607591	-0.069950812	-0.119403856
Left Hip	0.003814737	-0.019484536	-0.198284078
Right Shoulder	-0.00219989	-0.069832041	0.132929455
Right Hip	0.263017638	-0.022520684	-0.202218743

**Table of Means of Rotation Points**

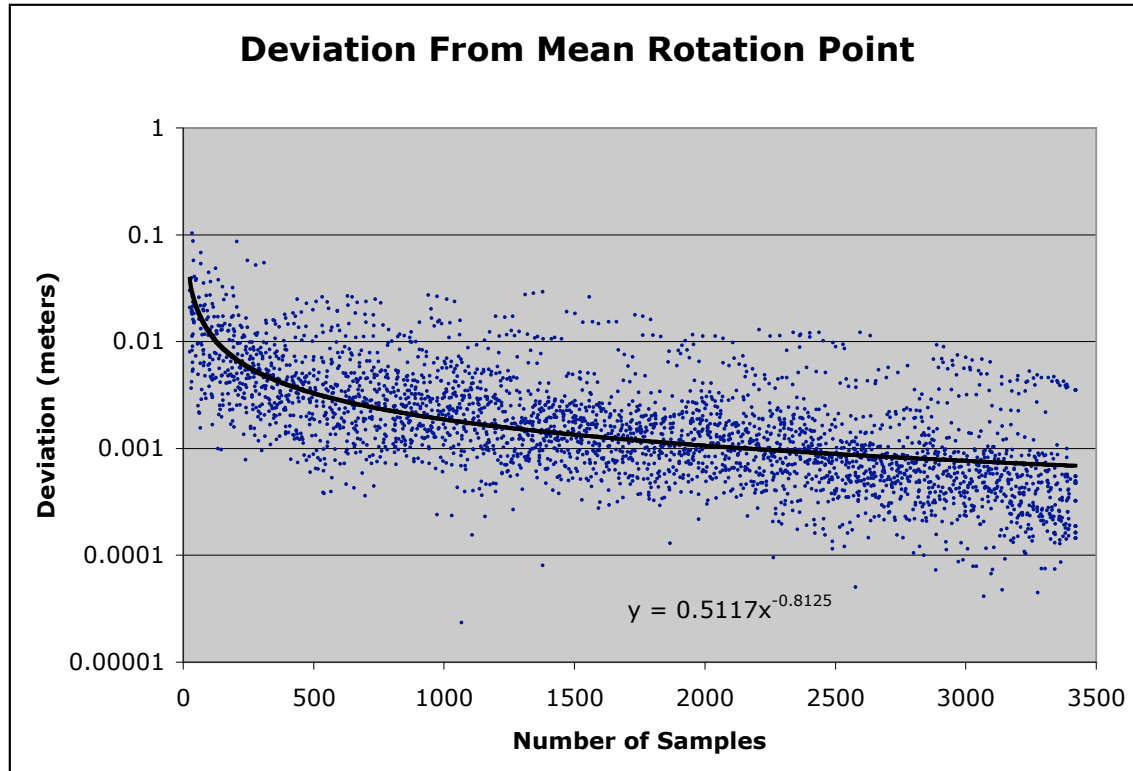
<b>Rotation Point</b>	<b><math>\sigma_x</math></b>	<b><math>\sigma_y</math></b>	<b><math>\sigma_z</math></b>
Waist	0.003896258	0.004605284	0.014676666
Neck	0.001768695	0.001355136	0.001206322
Left Ankle	0.01981794	0.011053845	0.011233588



Left Wrist	0.002071725	0.002539415	0.000947775
Left Elbow	0.015518996	0.004050952	0.00398332
Left Knee	0.011752322	0.002762332	0.003553895
Right Elbow	0.017223326	0.00815392	0.004313482
Right Knee	0.016813465	0.003744518	0.003104466
Right Ankle	0.129159355	0.034143233	0.020772927
Right Wrist	0.00163169	0.000935295	0.000834657
Left Shoulder	0.001960198	0.000852649	0.00314934
Left Hip	0.001577703	0.001635193	0.004697627
Right Shoulder	0.000891381	0.00158762	0.003755364
Right Hip	0.001447724	0.00128305	0.004635324

**Table of Standard Deviations of Rotation Points.**

Most of the standard deviations are less than one centimeter, but there are some significant outliers. Further analysis of the calculated points for the ankles and elbows show that the four runs produced two answers due to different orientations of the parent's reference frame. Therefore the standard deviation presented above for the ankles and elbows are erroneously calculating the deviation from the average of two distinct means. It is more appropriate to calculate the standard deviation from a single mean. When these outliers are removed from the calculation of the deviation, a very informative graph can be produced below. Every calculation for every segment is presented below as a deviation from the single mean rotation point versus the number of sample.



As can be readily seen from the above graph, a statistically significant amount of calculations are within one centimeter of accuracy when analyzing more than about 200 samples. The accuracy gets better on average with a power law of  $N^{-0.8125}$ .

### **2.1.4 Results**

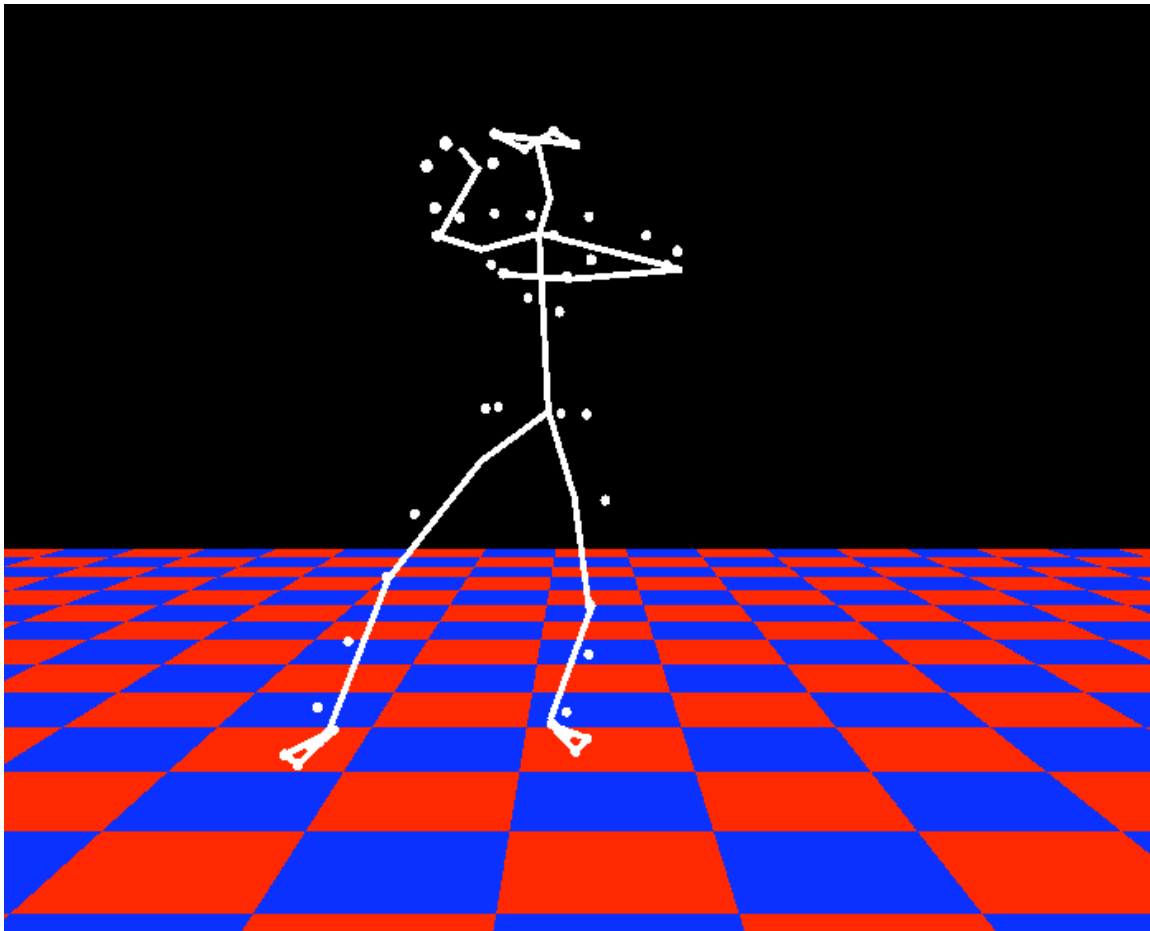
As has been shown by examples above, the Minimum Variance Method produces similar answers to the Least-Squares Method but without the initial guessing. This new method can produce a better answer in the cylindrical case with only a little extra work.

Statistical analysis of real motion capture data reveal an error of about one centimeter when 200 samples are analyzed. The error goes down to one millimeter when 2000 samples are analyzed. Lessons learned during implementation of the equations 2.1.3 and 2.1.4 have shown the the difference from the mean must be calculated during

each step of the summation. If the presented equation is algebraically re-arranged, propagation of finite math errors (as present in all computers) produce extremely significant errors in the results. This is not uncommon with these kinds of averaging and shows up in many “deviation” statistics. For a significant amount of motion capture data analyzed, the determinant of the matrix is small but since the condition number does not get very big, the matrix is not considered near singular. None-the-less, the Singular-Value Decomposition (SVD) is recommended instead of the Cholesky Decomposition for the fact that SVD significantly reduces the error propagation for small matrices. The matrix can become singular though in two different cases. The matrix is singular when the points are identical or when the points are planar. When the points are identical, the rotation point can be any point in space. This occurs if the joint doesn’t move. When the joint doesn’t move, then no rotation point can be calculated may just as well be modeled as being permanently attached. When the points are planar, the rotation point can be any point on the rotation axis. This is why the projection of the spherical answer along the axis onto the plane compensates for the spherical assumption error. This does not occur too often in real 3D motion capture data due to measurement errors. During any physically determined measurements, there is always some degree of noise. Measurement noise alone is enough to keep the matrix away from being singular. Just like an average, the Minimum Variance equation will smooth out the noise when enough data is sampled. This produces a very robust, deterministic answer for the rotation points. Through the many data-files that have been analyzed, this equation has produced visibly incorrect rotation points only if the joint didn’t move much. An example of connecting the rotation points is in the figure below. Each rotation point is well

Knight 22

determined from the 41 markers except the waist. This motion data entitled “ericcamper.c3d” is a figure doing some standing martial arts moves. He does not bend at the waist during the motion. As a result, the rotation point ends up high in his chest. There is no impact to the animation though since the joint never moved.



**Figure A - Rotation Points of Data**

### ***2.1.5 Limitations***

The limitations of the Minimum Variance Method for calculation are due to not following the previously stated requirements - i.e. the joint must move significantly

around a rotation point that is fixed relative to a stable reference frame. If this require fails, so does the algorithm.

## ***2.2 Segment Coordinate System***

Now to answer the question of where to get the fixed coordinate system on a segment from the data. Most information in motion capture data comes in the form of absolute Cartesian coordinates of markers placed on the segments of an articulated figure. This is considered the rawest form of the data. Usually, no information is available to determine the rotation points of the underlying skeleton. To determine these, the frames of data must be analyzed en-mass using a hierarchical model and the Minimum Variance Method. This method will only work if the following conditions are met for a joint:

- 1) No translational freedom.
- 2) The orientation of the parent segment can be determined.
- 3) The joint moves.

These restrictions are not that unreasonable since previous methods come with more. The orientation of a segment can be easily determined if there are at least three non-linear data points fixed to that segment, i.e. markers in a time frame. It still is possible to determine the orientation if there is only one or two points but is less accurate. There is a hierarchical dependency for determining the center of rotation and the orientation. The orientation can be determined if there:

- 1) is one data point, rotation point, and a rotation axis (non-linear).
- 2) are two data points and one rotation point (non-linear).
- 3) are three or more data points (non-linear).

Requirements 1 and 2 rely on previously calculated constants as can be determined by the Minimum Variance Method. The root of the segment tree is a special case and must follow requirement 3. All subsequent segments of the tree can follow any of the orientation requirements. So, for a human, the minimum number of data points is 17 for 15 segments. This  $N+2$  absolute minimum is not a recommendation. Requirements 1 and 2 rely on every joint below in the tree to calculate its rotation points properly. The ripple down effect can escalate to an undesirable level if these minimal requirements are followed. With that in mind a better number of markers to follow is  $3N$ , i.e. 3 for every segment.

The information can be retrieved if calculated hierarchically from root to leaf. First, define the tree. Then, assign the data points to their appropriate segments. The root must have three points. No center of rotation for the root can be determined. The children of the root can determine their center of rotation relative to their parent by Minimum Variance Method. If there are three or more points, each time frame can contribute to the fitting. The point of rotation in absolute coordinates for a segment is

$$r = p_c + A_c r'$$

$$\hat{n} = A_c \hat{n}'$$

Where  $r'$  is the constant relative rotation point,  $\hat{n}'$  is the constant relative rotation axis,  $p_c$  is the center of the coordinate system, and  $A_c$  is the 3x3 matrix of column vectors that represent the axes of the coordinate system.  $A_c$  is determined differently for whichever requirement is followed.

Requirement 1:

$$\hat{x} = \hat{n}$$

$$\hat{z} = \frac{p_0 \times \hat{x}}{\|p_0 \times \hat{x}\|}$$

$$\hat{y} = \hat{z} \times \hat{x}$$

$$p_c = r$$

Requirement 2:

$$\hat{x} = \frac{p_0 - r}{\|p_0 - r\|}$$

$$\hat{z} = \frac{p_1 \times \hat{x}}{\|p_1 \times \hat{x}\|}$$

$$\hat{y} = \hat{z} \times \hat{x}$$

$$p_c = r$$

Requirement 3:

$$\hat{x} = \frac{p_1 - p_0}{\|p_1 - p_0\|}$$

$$\hat{z} = \frac{p_2 \times \hat{x}}{\|p_2 \times \hat{x}\|}$$

$$\hat{y} = \hat{z} \times \hat{x}$$

$$p_c = p_0$$

Now that the three coordinate axes have been created, the rotation matrix can be assembled from the column vectors.

$$A_c = (\hat{x} \hat{y} \hat{z})$$

There is hierarchical dependency here in that requirements 1 and 2 make a segment depend on the parent segment's coordinate system to determine his own coordinate system. The C++ implementation of this recursive dependency is presented in the appendix. The dependencies work quite well with each other as long as the rotation points are accurate. This method was implement before the Minimum Variance Method

Knight 26

was discovered. During the many failures of other methods for determining the rotation points, this dependency was nasty. With the success rate of the Minimum Variance Method, the coordinate system calculations have become much cleaner.

## **2.3 Motion Capture Data**

Many disciplines need in-depth analysis of motion. Ergonomic studies need optimal reach information. Olympic runners require efficiency information for running better/farther/faster. In order to study the particular motion, data must be acquired for the particular motion regime. The motion capture data used in this thesis comes from the very large database (>2GB) of motions captured by Carnegie Mellon University (CMU) Graphics Lab. The data is freely downloadable at <http://mocap.cs.cmu.edu/>. The database was created with funding from National Science Foundation grant # EIA-0196217. There are 1576 trials in 6 categories and 23 subcategories and growing continuously.

### ***2.3.1 Capturing Data***

Data can either be artificially generated or actual measurements from actors. There are many different techniques that have been around for 40 years. This thesis is not involved in the process of capturing data but relies on previously captured data. Artificially generated data points are similar to animation key frames where an animator would create scenes that perform an act. These points may not be realistic but are compensated for realism with various simulation techniques. Real data is captured by attaching sensors to various places on each moving segment of the body. If the data is sparse, it is possible to manually insert critical data points like footsteps amongst the real



data. Motion capture data is usually acquired on forty or more points on the body depending on the motion being studied.

### ***2.3.2 Data Format***

The CMU data comes in a few file formats. The raw absolute Cartesian coordinates of the data in time are stored in the C3D file format. C3D is one of the oldest formats for storing this kind of data. Each time frame is stored, with each frame consisting of X,Y,Z for each marker on the body. If any data is missing from a frame, that point is zeroed and marked. The captured motion also comes in the form of ASF and AMC file formats. These are created after the VICON program has done analysis of the data. The ASF format stores the skeleton and joint information to create an articulated figure on the screen. The AMC format contains every time frame's translation and rotation for each bone. This work focuses on the use of raw XYZ data and therefore does not consider the post-analysis data in the AMC or ASF files. A new file format was produced to address the special use of tetrahedral meshes to build up the articulated figure. This format, called Articulated Tetrahedral Model (ATM), replaces the ASF format. A simple example is printed here of a 5R1P manipulation arm:

```
FIGURE 5R1P
SEGMENTS 3

MESH 0 cube.mesh
MASS 1.0
NAME Base
TRANSLATE_Z 0.5
JOINT Root
3 0.0 0.0 0.5
CHILDREN 1
1 // extendArm
ENDMESH
```

Knight 28

```
MESH 1 cube.mesh
MASS 1.0
NAME extendArm
SCALE_XYZ 0.2 0.2 1.0
TRANSLATE_Z 1.5
PARENT 0 // Base
JOINT 2R
3 0.0 0.0 1.0
DEGREES_OF_FREEDOM 2 0
VALUE 0.0
3 0.0 1.0 0.0
FROM -90.0 TO 90.0
VALUE 0.0
3 0.0 0.0 1.0
FROM -180.0 TO 180.0
CHILDREN 1
2 // Hand
ENDMESH
```

```
MESH 2 cube.mesh
MASS 1.0
NAME Hand
SCALE_XYZ 0.8 0.8 0.4
TRANSLATE_Z 2.5
PARENT 1 // extendedArm
JOINT 3R1P
3 0.0 0.0 2.0
DEGREES_OF_FREEDOM 3 1
VALUE 0.0
3 1.0 0.0 0.0
FROM -90.0 TO 90.0
VALUE 0.0
3 0.0 1.0 0.0
FROM -90.0 TO 90.0
VALUE 0.0
3 0.0 0.0 1.0
FROM -180.0 TO 180.0
VALUE 0.0
3 0.0 0.0 1.0
FROM 0.0 TO 0.4
ENDMESH
```

```
ENDFIGURE
```

The specification for the ATM format is explained in the Appendix.

### ***2.3.3 Correlating Data to Segments***

A motion capture file (e.g. C3D file) contains the data, as well as a name associated with each set. For example, a marker is placed on the right thigh and is called “JOE::RTHI”. Usually, the ATM file doesn’t have the same designation so a cross-correlation must be achieved to identify which segment this data marker belongs. This thesis has set up a two file process to cross-correlate a C3D data-set with the segments on the ATM model. Firstly, a correlation file specifies which standard Marker Set is to be used. It then lists the marker names in the C3D file alongside the marker names in a standard Marker Set. An example is as follows:

```
42 WANDS
MARKERSET vicon512.txt

JOE::LBWT = LBWT Left back waist
JOE::RBWT = RBWT Right back waist
JOE::LFWT = LFWT Left front waist
JOE::LTHI = LTHI Left thigh
JOE::RFRM = RARM Right forearm
...
```

The second file is the Marker Set that specifies the markers and their locations on the articulated model. An example of this file is as follows:

```
51 WANDS

WAND RFHD
SEGMENT Head
LENGTH 0.0
anterior right top

WAND LFHD
SEGMENT Head
```

Knight 30

```
LENGTH 0.0  
anterior left top  
  
WAND RSHO  
SEGMENT Chest  
LENGTH 0.0  
right mid top  
...
```

As the reader may notice, the format allows the placement of markers in a relative fashion onto the specified segment of the figure. This dual level correlation allows for the use of a single set of markers on a figure for many data-sets. This may or may not be an advantage depending on how varied the motion capture systems are. In CMU's case, about 80% of the motion capture has the same marker sets.

## 2.4 Motion Algorithm

The motion equations are expressed in homogeneous vectors and matrices. Homogeneous vector math is a convenience so that both translation and rotation of 3D vectors can be combined together into a single matrix. This technique of vector manipulation is fairly common and appears in the OpenGL standard for drawing 3D graphics. A 3D position vector is extended to four components where the fourth is usually set to one. A 3D direction vector is similarly extended except the fourth component is set to zero, in essence saying it is a position located at infinity. A homogenous matrix is a 4x4 matrix that usually contains (0 0 0 1) in the bottom row. These definitions simplify the sequential concatenation of translations and rotations as applied to a vector. A translation is expressed as a 4x4 matrix  $T$  multiplied to a 4x1 column vector thus:

$$T(r)a = r + a$$

where

$$T(r) = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The cross-product can similarly be turned into a 4x4 matrix operation

$$S(r)a = r \times a$$

where

$$S(r) = \begin{pmatrix} 0 & -z & y & 0 \\ z & 0 & -x & 0 \\ -y & x & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

A rotation around a unit vector  $\hat{r}$  by a counter clockwise angle  $\theta$  is expressed

similarly:

**Equation 2.3.1**  $R(\theta, \hat{r}) = I + S(\hat{r})[\sin\theta + S(\hat{r})(1 - \cos\theta)]$

where the angle is to be rotated around the unit vector  $\hat{r}$  using the Right-Hand Rule for direction of the angle. The following pseudo-code represents the entire motion algorithm for a generic tree-structured articulated figure.

### 2.4.1 Pseudo-code

#### Procedure 2.3.1

```

Procedure MoveArticulatedFigure( time )
Begin
    motion = IdentityMatrix
    MoveSegment( rootOfFigure, time, motion )
End

```

Knight 32

### Procedure 2.3.2

```
Procedure MoveSegment( segment, time, motion )
```

```
Begin
```

```
    motion = motion * MovementMatrix(time)
```

```
    pointOfRotation = motion * pointOfRotation
```

```
    rotationAxes = motion * rotationAxes
```

```
    vertices = motion * vertices
```

```
    for each child of segment
```

```
        Begin
```

```
            motionCopy = motion
```

```
            MoveSegment( child, time, motionCopy )
```

```
        End
```

```
End
```

```
Function MovementMatrix( time )
```

```
Begin
```

```
    displacement = Displacement(time)
```

```
    angles = Angles(time)
```

```
    m = T( pointOfRotation+displacement-  
previousDisplacement )
```

```
    for each axis of rotationAxes
```

```
        m = m * R( angles[i]-previousAngles[i], axis )
```

```
    m = m * T(-pointOfRotation)
```

```
    previousDisplacement = displacement
```

```
    previousAngles = angles
```

```
    return m
```

```
End
```

```
Function Displacement( time )
```

```
Begin
```

```
    return position of point of rotation  
        relative to parent segment
```

```
End
```

```
Function Angles( time )
```

```
Begin
```

```
    return array of angles for the axes  
        relative to parent segment
```

```
End
```

The two user-supplied functions Displacement and Angle will provide the amount of change for each degree of freedom (DOF) for the segment's joint. This research has

used Taylor series expansions to calculate these values for each DOF as well as followed the data directly. These procedures are compatible with the Denavit-Hartenberg notation for generic joint-link coordinate frames. These procedures are not used when the Minimum Variance Method is calculating the rotation points. They are used subsequently once the points are determined.

### 3 Articulated Model

Any three-dimensional simulation requires three-dimensional information. Most of the models followed by previous authors use either stick figures; shapes defined by 2D surfaces; or easy 3D shapes for each segment. Two-dimensional surfaces are very time consuming to calculate volumetric information. One must perform a surface integral approximation. The added complexity is inadequate for real-time physical simulations making volumes and moments of inertia unnecessarily difficult to calculate. Simple three-dimensional shapes such as blocks, ellipsoids and cylinders have been used in the past but reduce the ability for diverse shapes. This research had the intention of using physical calculations in its motion model. As it turns out, none of the physics was needed by the Minimum Variance Method. The new method uses the hierarchical organization of the segments in order to traverse the tree for calculations. None-the-less, this articulated model is still an efficient aid to modeling and is presented here anyway. A three-dimensional equivalent to the triangulated mesh is used. A rigid body (e.g. each segment) is subdivided into face-connected tetrahedrons to make up a tetrahedral mesh. Each tetrahedron will have four vertices, four triangular faces, six edges and density. The advantage of dividing space into tetrahedrons is to have the ability to vary the mass distribution and to ease inertial calculations. A tetrahedral mesh can approximate any volumetric shape. The approximation of the original shape gets better when more tetrahedrons are used within a shape.

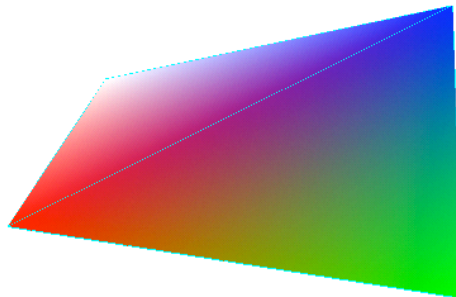


To ease drawing, each face of the tetrahedra are labeled if they are on the surface of the shape. The draw algorithm will traverse the list of tetrahedrons and draw (using OpenGL) only those triangle faces that are labeled as on the surface.

A hierarchical approach to building the articulated figure is used in the thesis. The figure is made of segments; the segments are made of a rigid body and a joint; the rigid body is made of a tetrahedral mesh; the tetrahedral mesh is made of tetrahedrons; the tetrahedra are made of triangles and vertices.

### 3.1 Tetrahedron

The tetrahedron is the simplest 3D shape that can fill a volume completely when placed together face to face.



**Figure B - Tetrahedron**

Any arbitrary shape can be achieved by placing tetrahedrons next to each other. Granted the resulting surface is not smooth but today's graphics engines are specifically designed to draw triangulated surfaces.

The tetrahedron is made of four triangle faces and four vertices. Given the vertices  $\vec{r}_i$ , the volume is easy to determine by the following formula:<sup>10</sup>.

---

<sup>10</sup> CRC Handbook 1985

$$V_{Tet} = \frac{1}{6}(r_3 - r_0) \cdot (r_2 - r_0) \times (r_1 - r_0)$$

Assuming the density is constant over the tetrahedron, the center of mass is the centroid. The centroid is easily determined by the average of the four vertices.

$$C_{Vol} = \frac{1}{4} \sum_{i=0}^3 r_i$$

There is a fairly simple algorithm if the density is assumed to be linearly changing between the four vertices. It can be derived by doing an integration over the tetrahedron volume of the differential mass. If each vertex has a density value associated, the center of mass can be calculated with

$$C_{mass} = \frac{4}{5} C_{Vol} + \frac{1}{20\bar{\rho}} \sum_{i=0}^3 \rho_i r_i$$

$$m_{Tet} = V_{Tet} \bar{\rho}$$

Other quantities are more difficult but can be exactly determined. The moment of inertia and angular momentum are very desirable traits to follow during any dynamic simulation. They both involve integrating over the tetrahedral volume. The equations have been derived in the Appendix. The result is a matrix equation involving the geometric tensor defined by

$$Q(r) = r^T r - r r^T = \begin{pmatrix} r^2 - x^2 & -xy & -xz \\ -yx & r^2 - y^2 & -yz \\ -zx & -zy & r^2 - z^2 \end{pmatrix}$$

where  $r = (x \ y \ z)^T$  and in general

$$pp^T = \begin{pmatrix} x \\ y \\ z \end{pmatrix} (x \ y \ z) = \begin{pmatrix} xx & xy & xz \\ yx & yy & yz \\ zx & zy & zz \end{pmatrix}$$

$$p^T p = (x \ y \ z) \begin{pmatrix} x \\ y \\ z \end{pmatrix} = x^2 + y^2 + z^2$$

and the properties of moment of inertia and angular momentum about the center of rotation ( $C_{rot}$ ) become

$$I_{Tet} = m_{Tet} \hat{w}^T Q_{Tet} \hat{w}$$

$$L_{Tet} = m_{Tet} Q_{Tet} w$$

where

$$Q_{Tet} = Q(C_{mass} - C_{rot}) + \frac{1}{20} \sum_{i=0}^3 Q(r_i - C_{mass})$$

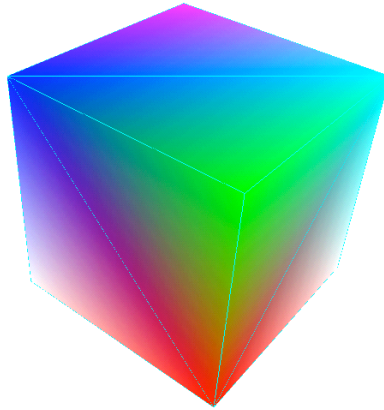
These calculations pave the way for the macro properties and the energy calculations, which are more interesting for studies in dynamics.

### 3.2 Tetrahedral Mesh

The tetrahedral mesh is a packed group of tetrahedrons (cf. Figure C), face-to-face, producing a solid shape. This mesh has the properties of mass, volume, moment of inertia, angular momentum, all of which can be simply added together from its constituent tetrahedrons. The geometric tensor  $Q$ , centroid, and the center of mass are not additive but are weighted averages of the constituent tetrahedrons.

$$C_{Total} = \frac{\sum C_i m_i}{\sum m_i}$$

The time complexity of all quantities at this level is  $O(n)$  where  $n$  is the number of constituent tetrahedrons. A simple cube is made of five tetrahedrons as in the following figure.



**Figure C - Tetrahedral Mesh**

### 3.3 Rigid Body

A tetrahedral mesh can be made a rigid body once it has been manipulated into its final shape. This new object has the property that it cannot be molded or stretched. It still can be translated and rotated as a whole. This has advantages because the rigid body has a fixed volume, mass, principal moments, principal axes and relative center of mass thereby reducing the calculations. The fixed values can be calculated at time of creation of the body so it will not be necessary to calculate during motion. The volume and mass are simple additions. The center of mass and geometric tensor are simple weighted averages. The principal moments and principal axes are much more complicated but only need to be calculated once. The principal moments and axes have been derived in the Appendix. The simplified moment of inertia and angular momentum is composed of the principal moments and axes, and the center of mass inertia tensor as derived in the Appendix.

$$I = m\hat{w}^T Q(C_{mass} - C_{rot})\hat{w} + \hat{w}^T \left( \sum_{i=0}^2 \lambda_i \hat{p}_i \hat{p}_i^T \right) \hat{w}$$

$$L = mQ(C_{mass} - C_{rot})w + \left( \sum_{i=0}^2 \lambda_i \hat{p}_i \hat{p}_i^T \right) w$$

where  $\lambda_i$  are the principal moments

$p_i$  are the principal axes

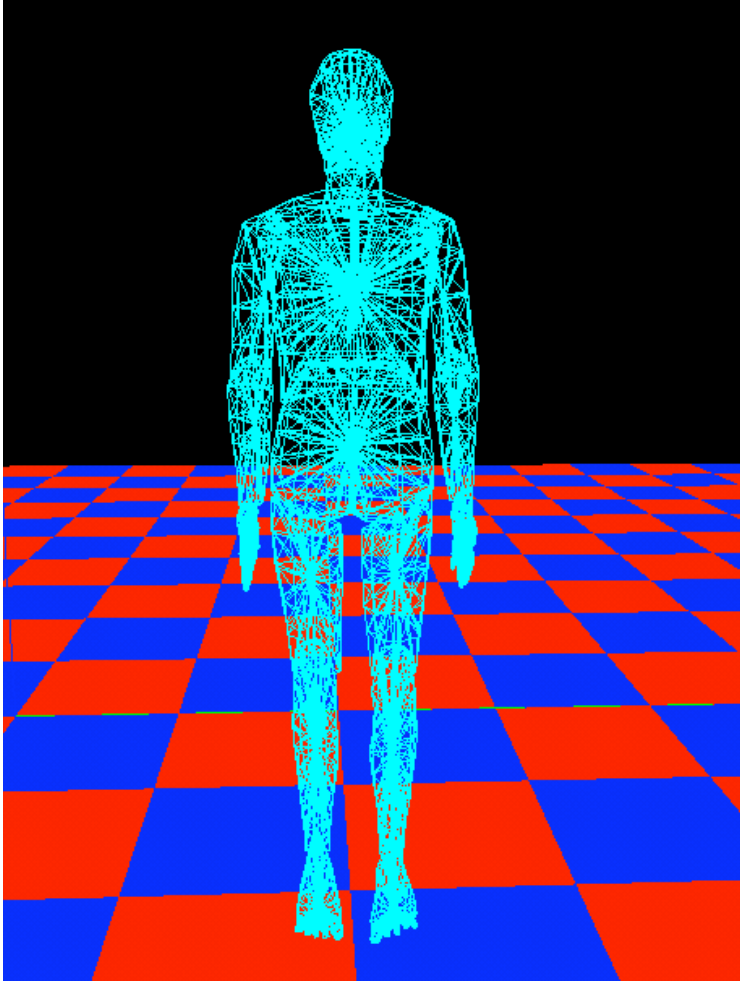
These equations are much simpler than summing the calculated values for the constituent tetrahedrons. As the rigid body is translated and rotated, so must the principal axes and center of mass. If those are maintained throughout the figure motion, the inertial properties will not be complex to calculate.

### 3.4 Segment

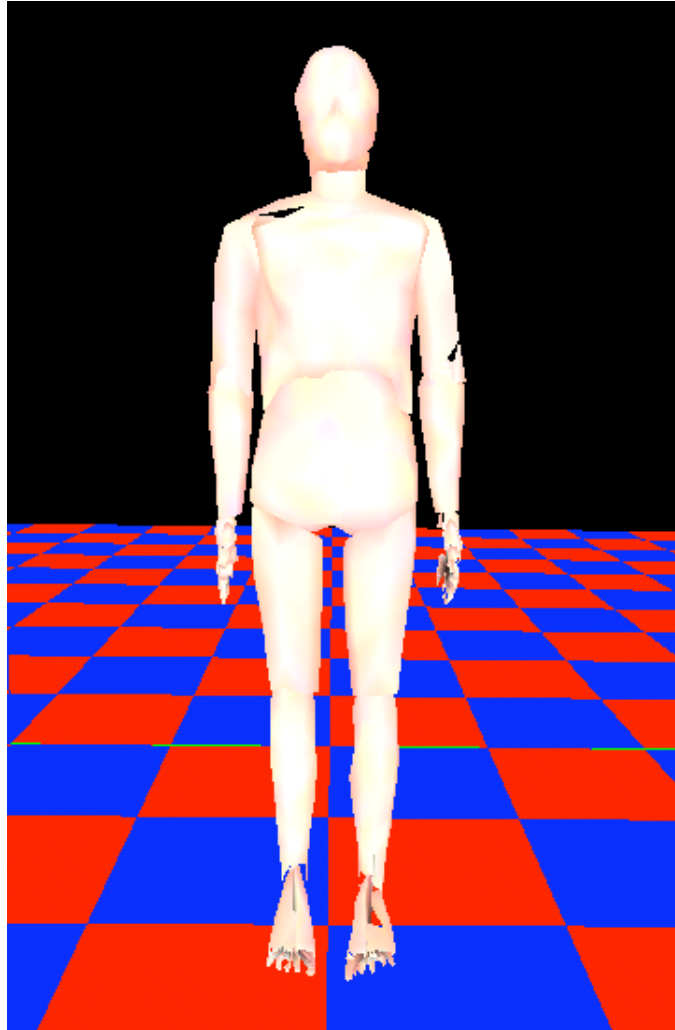
The segment is composed of a rigid body and a joint. The segment also has knowledge of its parent and its children in the tree of segments that make up the whole figure. The segment is the level at which the calculations are made for the motion model. The segment's motion variables are the constants defining the relative position of the rotation point.

### 3.5 Figure

The articulated figure is a set of segments attached into a tree structure. There are some segments without children and one without a parent. The one without a parent is the root segment. The ones without children are the end-effectors.



**Figure D - Articulated Tetrahedral Mesh**



**Figure E - Articulated Figure**

For any articulated figure, there is always one root segment. It does not matter which segment is the base since there is no physical significance (the hand serves just as well as the chest). The hips were chosen in Figure E because it is nearest to the center of mass of the human figure and the fact that there is usually four markers on the hips. The root of the segment tree is the starting point for message sending. Bolt (2000) used 17 degrees of freedom (DOF) in his model of the human figure. Ko and Cremer (1996) used 34 DOF for their a-priori system. The Minimum Variance Method is independent of the number degrees so the DOF model is a self-imposed limit for other kinds of simulations.

### ***3.5.1 Converting Triangulated Surface into Tetrahedral Solid***

Most figures are available as a set of triangles that completely cover the surface. While this is easy to draw, it is difficult to calculate physical properties such as volume and inertia tensor. Converting a triangulated surface into a tetrahedral solid would make it easier to handle for physical simulations. The conversion is very time consuming and should be done offline to the simulation. There are a few techniques available to produce a tetrahedral solid. Delaunay meshes are a wonderful and well-studied technique that can produce a tetrahedral mesh of any set of points provided it is a convex figure.

#### **3.5.1.1 Triangulated Surface with no normals**

Which way is outside? A surface without normals can only answer this by using the 3-D equivalent of the Jordan curve Theorem. Unfortunately the theorem is only proven for the 2D case. The 3D equivalent has known counterexamples that must be explained in order to be useful.

#### **Jordan Curve Theorem**

A simply closed curve divides the region into two distinct areas; an inside and an outside.

A simple test to see if a point exists inside or outside the curve involves shooting a ray in an arbitrary direction and counting the times it crosses the curve. If the count is odd, the point is inside; if even it is outside. This can be used in 3D keeping in mind that there are special extreme surfaces that don't work. The known figures that don't work are



the Klein Bottle (which is not simply closed); a surface with a hole (not closed); Alexander's Horned Sphere (has infinitely recursive horns wrapping around itself). A real-world limitation can exclude these known problems.

**Theorem 1**

A simply closed tessellated surface with finite size and finite number of elements divides space into two distinct regions; an inside and an outside.

**Lemma 1**

A neighboring element of a simply closed tessellated surface has the outside on the same side.

**Lemma 2**

An arbitrary ray from an outside point will cross a simply closed tessellated surface an even number of times. An inside point will have an odd number of crossings.

**Proof by Contradiction**

A neighboring element has the opposite side be the outside. Look at the intervening edge. Place a point epsilon outside of both elements such that no other elements have been crossed and near the intervening edge. Now draw a ray from one point through the neighboring point. The number of crossings of one point would be exactly one more than the other number of crossings. This would specify one point is on the inside and one point is on the outside that contradicts the starting conditions.

**3.5.1.2 Central Convexity Point insertion.**

Extra points carefully placed inside an arbitrary figure can produce a simple tetrahedral mesh. The simple technique involves inserting points inside of the figure such that the point can "see" the inside of the surface triangles in its immediate vicinity. The

Knight 44

necessary condition for this point placement is that it can be used as a vertex of a tetrahedron as long as the surface triangle normal is pointing away from it (i.e. convex). Once the point can no longer see contiguous surface triangles, a new point must be inserted. The insertion continues until there are no more surface triangles left to use in a tetrahedron.

## 4 Programming Model

### 4.1 Articulated Figure

The articulated figure designed for this research is for general purpose articulated modeling. The original attempt was to use it for physical based calculations. With the advent of the Minimum Variance Method, the model has been resigned to a simple organizational tool for the hierarchical tree traversals. The entire implementation is explained here for future use. The C++ object model is set up in a hierarchical manner for the articulated figure. An articulated figure (ArtFigure.cpp) is made of segments (Segment.cpp). A segment is a rigid body (RigidBody.cpp) that is linked to another by a joint. The rigid body is a shape that can be translated and rotated, but not molded. The rigid body is made of a mesh of tetrahedrons (TetrahedralMesh.cpp). The mesh is a face-connected list of tetrahedrons. Only faces on the surface have no connected tetrahedrons and are available for drawing. A tetrahedron (Tetrahedron.cpp) is a four sided figure with four vertices assigned. Each side of the tetrahedron is a triangle (Triangle.cpp). Since adjacent tetrahedrons share vertices, the tetrahedrons will contain only references to its points. All of the points for the tetrahedral mesh will be stored in a linear array inside the TetrahedralMesh instance. This allows easy access to the points for quick translations and rotations without duplication of effort. The following figure is a UML Diagram describing the relationships between objects.

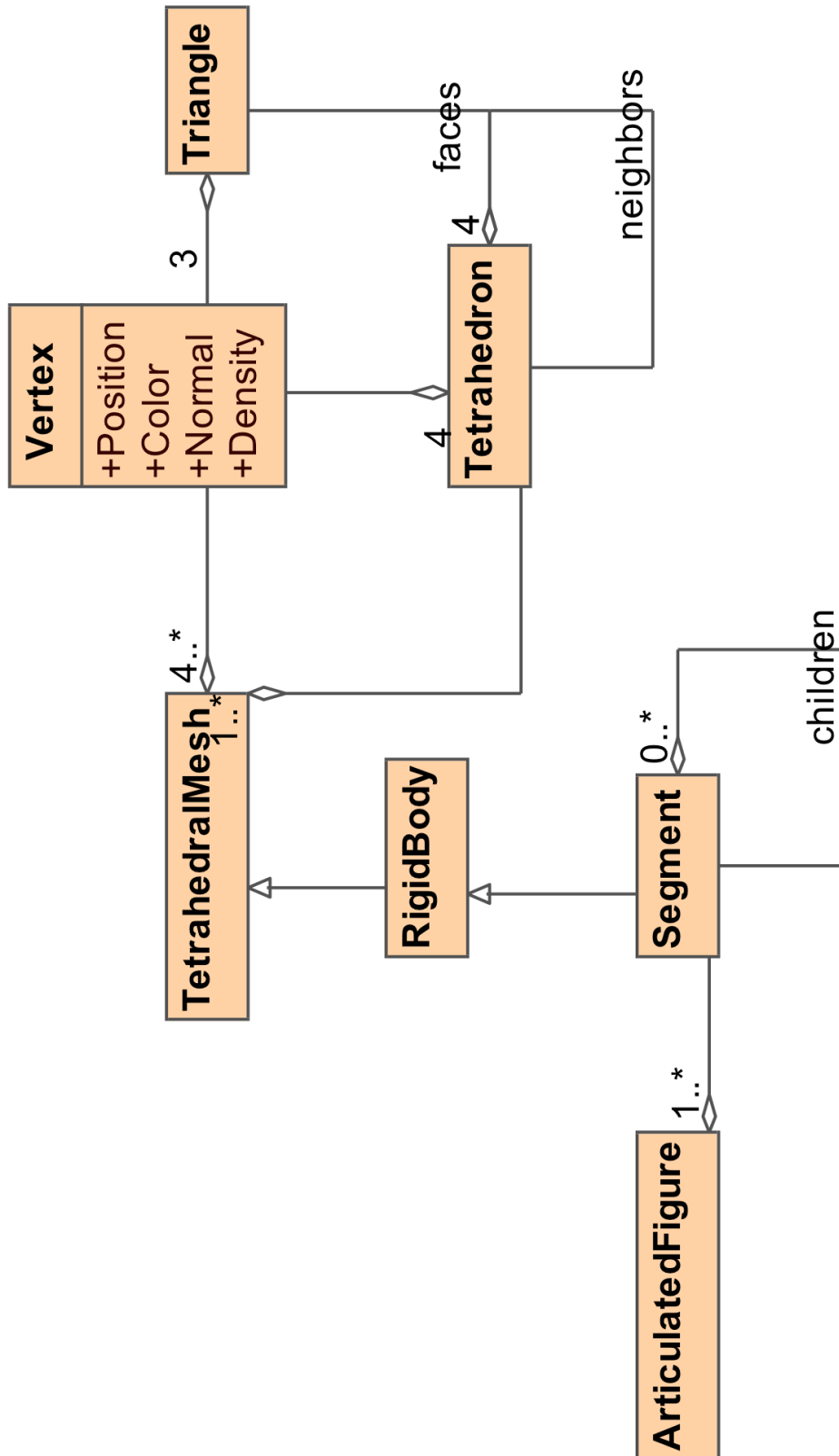


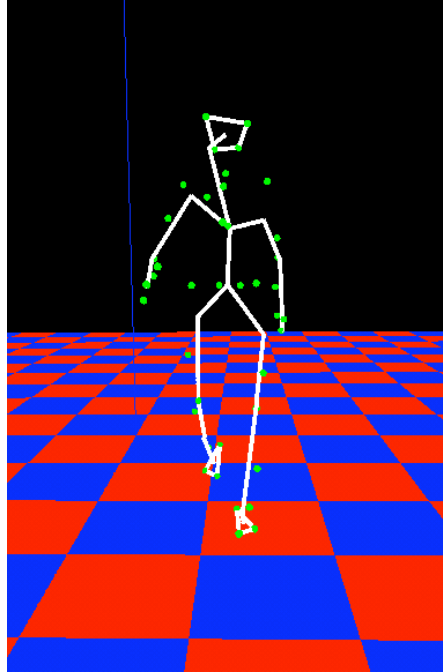
Figure F - UML Diagram

## 5 Products

An extremely large (~4GB) set of motion capture data has been downloaded free off the Internet. Most of the sets come from Carnegie Mellon University Graphics Lab. A large variety of motion regimes were analyzed by the Minimum Variance method. Each of the data-sets had their different quirks like missing time frames or negligible motion for certain joints. The Minimum Variance Method is actually independent of missing data and time frames. As counter-intuitive as it seems, the more complicated the motion, the easier it is to calculate the rotation points using the Minimum Variance method. The original list of simple motions have been changed to include more complicated ones such as break-dancing and rolling on the floor. To be fair though, an example of a weak product is given.

### 5.1 Walking Human Figure

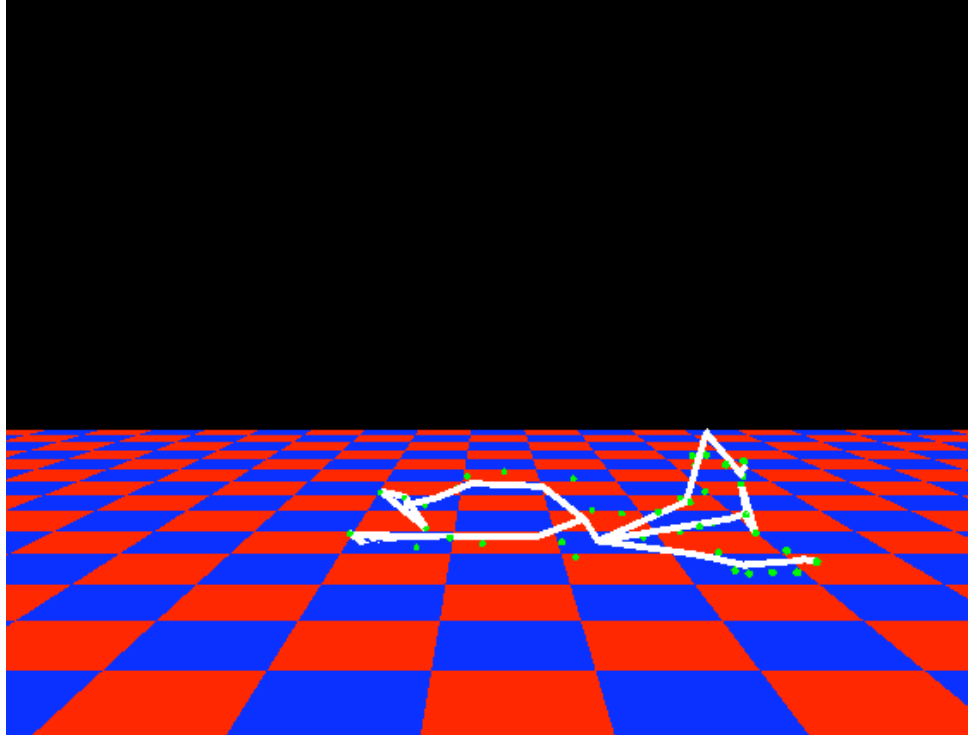
This motion regime has some issues with the Minimum Variance Method. During normal walking, almost all of the joints exhibit either cylindrical motion or no motion at all. As you can see, this motion regime does not fair well because of the small amounts of motion. The left arm and the neck have produced visibly off positions.



CMU/02\_01

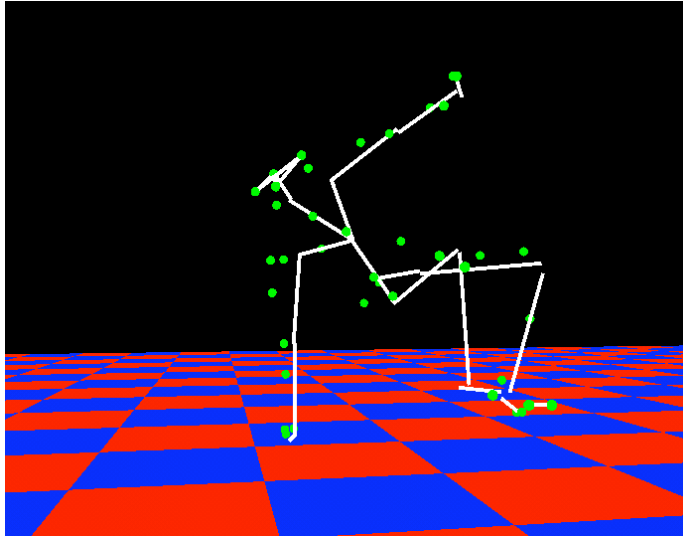
## 5.2 Rolling On Floor

Rolling on the floor was a data-set that was avoided for a good part of the earlier research because it was assumed the motion regime was too complex to handle for most methods. It was the first successful data-set for the Minimum Variance Method.



### 5.3 Break-dancing

The break-dancing sequence in the CMU data-set 85-14 is a very successful match for the requirements of the Minimum Variance Method. Nearly all of the joints are exercised during the motion capture and therefore a strong collection of constants for the motion. The drawing of the figure at everyday time frame looks realistic and some reviewers have agreed that the motion animated by Minimum Variance has more realism than that rendered by inverse kinematics from CMU.

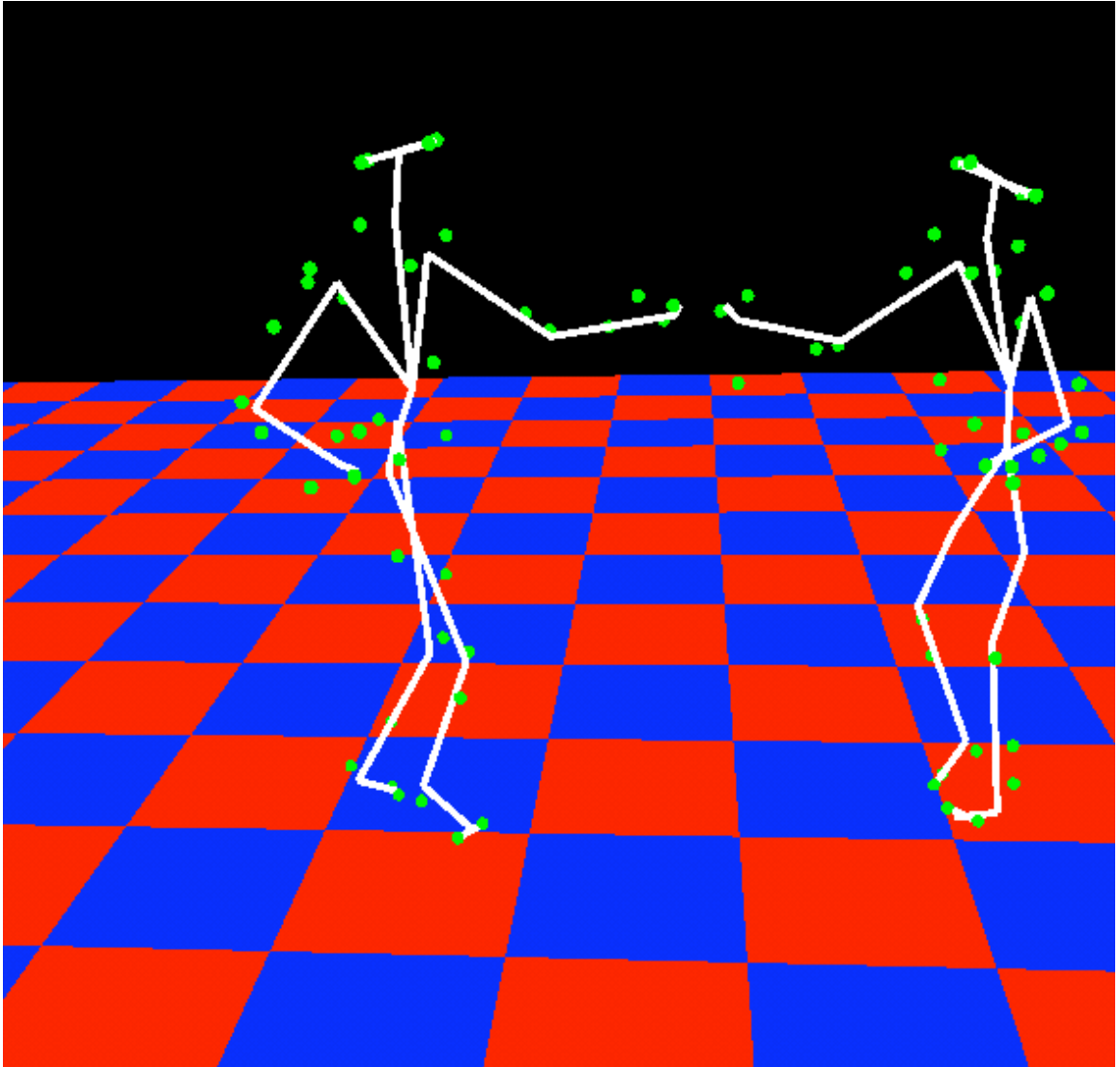


CMU/85-14

## 5.4 Salsa Dance

The salsa dance sequence is also a difficult data-set to model. There are two actors in the one c3d file. Inverse kinematics methods usually will consider the figures one at a time to analyze the motion. The Minimum Variance Method can analyze both figures and start drawing them in real-time. Separate segment trees must be maintained and two sets of constants are calculated for each figure. The method successfully and realistically handled the dual figure salsa dancing as can be seen in the picture below and in movies that were generated in the research material.





CMU/60-8

## 6 Conclusion

The Minimum Variance Method is a very robust method in the face of complicated motions. For a given actor and marker-set, a single set of constants can be calculated for each joint if the actor first does a “convolution” run which involves rotating every modeled joint to a significant degree. These constants can be calculated in the face of sparse data and uncertain measurements. These calculations can be as accurate as one millimeter depending on the amount of data analyzed and the quality of the data. Once these constants are stored, they can be reused for any new motion capture as long as the actor and markers do not change. In addition to this reusability convenience, the algorithm is quick,  $O(N)$ , depending on how many points are analyzed for each joint. Existing methods, i.e. least-squares fitting involve more amount of work because of the simple fact there are iterations involve in finding an optimized skeleton. Another advantage over existing methods is the fact that there can be only one solution calculated from the data. It has been shown at the beginning an example of this break-down. Optimizing in placing markers on the body can be done. It is standard practice to place markers on the joints but this does not aid this new algorithm. Ideally there should be three non-linear markers on each segment and they should be at least 5 cm from the rotation point unless small joints like the fingers make it scale down. This research has shown that a stable, realistic, real-time skeleton can be retrieved directly from the data without any interpolations or guessing. The new method has also been proven to be a suitable closed-form replacement for linear or non-linear least-squares fitting of data to a sphere, cylinder, circle or plane. It has further been proven that it is a significant

improvement on existing methods though it is not a complete replacement of Inverse Kinematics (IK). IK is still useful in producing joint angles where there are no motion capture data available. The Minimum Variance Method specifically solves the problem of finding an underlying skeleton in motion capture data with significant improvement in speed and work involved. Future research can involve building a real-looking figure on top of the skeleton to produce either game quality figures or movie quality figures that move to the realism that the Minimum Variance Method can achieve.

## 7 Bibliography

Aggarwal, J. K., Cai, Q., and Liao, W. “Nonrigid motion analysis: articulated and elastic motion” *Computer Vision and Image Understanding* 70, no. 2 (May 1998) : 142-56.

Aggarwal, J. K., Cai, Q., Liao, W., Sabata, B. “Articulated and Elastic Non-Rigid Motion: A Review” *Proceedings from Motion of non-rigid and articulated objects Workshop -- Los Alamitos, CA: Austin, TX: IEEE Computer Society Press, Nov 1994: 2-15*

Arikan, O., Forsyth, D. A., “Interactive Motion Generation from Examples” *Proceedings of SIGGRAPH 2002: 483-490.*

Bodenheimer, B., Rose, C. Rosenthal, S. Pella, J. “The Process of Motion Capture: Dealing with the Data” *Eurographics CAS (1997)*

Bolt, D.A. “Two Stage Control for High Degree of Freedom Articulated Figures” Ph.D. Thesis for Computer Science, University of Colorado in Colorado Springs, (2000).

Buades, J. M., Mas, R., and Perales, F. J. “Matching a Human Walking Sequence with a VRML Synthetic Model” *Articulated motion and deformable objects; Proceedings from the International Workshop – Sep 2000: Spain: 145-158.*

Cohen, M. “Interactive Spacetime Control for Animation” *Computer Graphics (SIGGRAPH 1992 Proceedings): vol. 26: 293-302.*

*CRC Standard Mathematical Tables 27<sup>th</sup> ed.*, William H. Beyer editor, CRC Press, Inc., Boca Raton, Florida 1985.

- Dettman, J.W. *Mathematical Methods in Physics and Engineering* Dover Publications, Inc. Mineola, N.Y, 1988.
- Duff, P. and Muller, H. "Autocalibration Algorithm for Ultrasonic Location Systems" 0-7695-2034-0/03 IEEE (2003): 62-68.
- Featherstone, Roy. "A Divide-and-Conquer Articulated-Body Algorithm for Parallel  $O(\log(n))$  Calculation of Rigid-Body Dynamics. Part 1: Basic Algorithm", *International Journal of Robotics Research* 18, no. 9 (1999): 867-875.
- Fowles, G.R. *Analytical Mechanics* 3<sup>rd</sup> ed. Holt, Rinehart, Winston: New York 1977.
- Grochow, K., Martin, S., Hertzmann, A., Popovic, Z. "Style-Based Inverse Kinematics" *ACM Transactions on Graphics: Proceedings of ACM SIGGRAPH 2004*: p 522-531.
- Herda, L., Fua, P., Plänklers, R., Boulic, R., Thalmann, D. "Skeleton-Based Motion Capture for Robust Reconstruction of Human Motion" *Computer Graphics Lab (LIG) EPFL: Lausanne, Switzerland* (2000).
- Holt, R. J., Huang, T. S., and Netravali, A. N., Qian, R. J. "Determining articulated motion from perspective views: a decomposition approach" *Pattern Recognition*: 30, no. 9, (1997): 1435.
- Jacobs, D. W., Chennubhotla, C. "Segmenting Independently Moving, Noisy Points" *Proceedings from Motion of non-rigid and articulated objects Workshop -- Los Alamitos, CA: IEEE Computer Society Press: Nov 1994: Austin, TX: 96-103.*
- Ko, H., Cremer, J. "VRLOCO: Real-Time Human Locomotion from Positional Input Stream" *Presence* 5(4) (1996): 367-380.
- Kovar, L., Gleicher, M. "Automated Extraction and Parameterization of Motions in Large

Knight 56

Data Sets” ACM Transactions on Graphics: *Proceedings of ACM SIGGRAPH 2004*: p 559-568.

Kovar, L., Gleicher, M., Pighin, F. “Motion Graphs” ACM Transactions on Graphics: *Proceedings of ACM SIGGRAPH 2002*: 473-482.

Latecki, L.J. “3D Well-Composed Pictures” *Graphical Models and Image Processing*: 59, No. 3 (May 1997): 164-172.

Lee, J., Shin, S.Y. “A Hierarchical Approach to Interactive Motion Editing for Human-like Figures” *Proceedings of ACM SIGGRAPH 1999*.

Lee, J., Chai, J, Reitsma, P. S. A. “Interactive Control of Avatars Animated with Human Motion Data” *Proceedings of SIGGRAPH 2002*: 491-500.

Li, Y, Wang, T., Shum, H-Y. “Motion Texture: A Two-Level Statistical Model for Character Motion Synthesis” *Proceedings of SIGGRAPH 2002*: 465-472.

Lopez, L F, Coutinho, F A B “Erratum: "Motion of articulated bodies: An application of gauge invariance in classical Lagrangian mechanics" (Am. J. Phys. 65 (6),528 -536 (1997))” *American Journal of Physics*: 66, no. 3, (1998): 252.

Lopez, L F, Coutinho, F A B “Motion of articulated bodies: An application of gauge invariance in classical Lagrangian mechanics” *American Journal of Physics*: 65, no. 6, (1997): 528-536.

Liu, C. K., Popovic, Z. “Synthesis of Complex Dynamic Character Motion from Simple Animation” *Proceedings of SIGGRAPH 2002*: 408-416.

Lui, Z, Gortler, S., Cohen, M. “Hierarchical Spacetime Control” Computer Graphics (*SIGGRAPH 1994 Proceedings*).

Motion Lab Systems, Inc. “C3D Format User Guide”, PDF document at <http://>

[www.motion-labs.com](http://www.motion-labs.com), Motion Lab Systems, Baton Rouge, LA, (2003) (98 pages)

O'Brien, J., Bodenheimer, Jr., R., Brostow, G.J., Hodgins, J.K. "Automatic Joint Parameter Estimation from Magnetic Motion Capture Data" *Graphics Interface* (2000)

Pollard, N. S., Reitsma, P. S. A. "Animation of Humanlike Characters: Dynamic Motion Filtering with a Physically Plausible Contact Model" *Yale Workshop on Adaptive and Learning Systems* (2001).

Popovic, Z., Witkin, A., "Physically Based Motion Transformation" *Computer Graphics Proceedings, Annual Conference Series*, 1999.

Press, W.H., Teukolsky, S.A., Vetterling W.T., Flannery, B.P. *Numerical Recipes in C: The Art of Scientific Computing* 2<sup>nd</sup> ed., Cambridge University Press: 1992.

Rao, Kashi "Describing and Segmenting Scenes from Imperfect and Incomplete Data" *Computer Vision Image Processing: Image Understanding*: 57, no. 1: Jan 1993: 1-23.

Rose, C., Guenter, B., Bodenheimer, B., Cohen, M. "Efficient Generation of Motion Transitions using Constraints" *Computer Graphics (SIGGRAPH 1996 Proceedings)*.

Safonova, A., Hodgins, J., Pollard, N. "Synthesizing Physically Realistic Human Motion in Low-Dimensional, Behavior-Specific Spaces" *ACM Transactions on Graphics: Proceedings of ACM SIGGRAPH 2004*: p 514-521.

Semwal, S., Parker, M. "An Animation System for Biomechanical Analysis of Leg Motion and Predicting Injuries during Cycling" *Real-Time Imaging*: 5(1999): 109-123.

Semwal, S.K., Armstrong, J.K., Dow, D.E., Maehara, F.E. "Multimouth surfaces for

- synthetic actor animation” *The Visual Computer*: 10 (1994): 388-406.
- Semwal, S.K., Hightower, R., Stansfield, S. “Closed Form and Geometric Algorithms for Real-Time Control of an Avatar” *IEEE Proceedings of VRAIS 1996*.
- Shakarji, C.M. “Least-Squares Fitting Algorithms of the NIST Algorithm Testing System” *J. of Research of the National Institute of Standards and Technology*: 103, No. 6 (1998): 633.
- Stone, M., DeCarlo, D., Oh, I., Rodriguez, C., Stere, A., Lees, A., Bregler, C. “Speaking with Hands: Creating Animated Conversational Characters from Recordings of Human Performance” *ACM Transactions on Graphics: Proceedings of ACM SIGGRAPH 2004*: p 506-513.
- Tabb, K., Davey, N., Adams, R., George, S. “Analysis of Human Motion Using Snakes and Neural Networks” *Articulated motion and deformable objects; Proceedings from the International Workshop*: Sep 2000: Spain: 48-57.
- Theobalt, C., Albrecht, I., Haber, J., Magnor, M., Seidel, HP. “Pitching a Baseball - Tracking High-Speed Motion with Multi-Exposure Images” *ACM Transactions on Graphics: Proceedings of ACM SIGGRAPH 2004*: p 540-547.
- Tolani, D., Goswami, A, Badler, N.I. “Real-time inverse kinematics techniques for anthropomorphic limbs” *Graphical Models*: 62 (2000): 353-388.
- Tresadern, P. and Reid, I. “Uncalibrated and Unsynchronized Human Motion Capture: A Stereo Factorization Approach” University of Oxford, England
- Tsai, H., “Symbolic Modeling and Dynamic Analysis of an Articulated N-Link Multibody System with Joint Compliance in a Gravitational Field”, Ph. D. Dissertation at Auburn University, Alabama, (1995).



- van de Panne, M., Fiume, E., Vranesic, Z. “Reusable Motion Synthesis using State-Space Controllers” *ACM Computer Graphics*: 24, no. 4 (Aug 1990): 225-234.
- van de Panne, “From Footprints to Animation” *Computer Graphics forum*: 16, no. 4 (1997): 211-223.
- Wee, L-B, Walker, M. W., McClamroch, N. H. “An Articulated-Body Model for a Free-Flying Robot and Its Use for Adaptive Motion Control”, *IEEE Transactions on Robotics and Automation*: 13, no. 2 (1997): 264-277.
- Wee, L., Walker, M, McClamroch, N. H. “An Articulated-Body Model for a Free-Flying Robot and Its Use for Adaptive Motion Control”
- Welch, G., Bishop, G. “SCAAT: Incremental Tracking with Incomplete Information” U. of North Carolina at Chapel Hill: <http://www.cs.unc.edu/~welch>.
- Wiley, D. J., Hahn, J. K. “Interpolation synthesis of articulated figure motion” *IEEE Computer Graphics and Applications*: 17 (Nov./Dec. '97): 39-45
- Witkin, A, Kass, M. “Spacetime Constraints” *Computer Graphics (SIGGRAPH 1988 Proceedings)*: 22: 335-342.
- Yamane, K., Kuffner, J., Hodgins, J. “Synthesizing Animations of Human Manipulation Tasks” *ACM Transactions on Graphics: Proceedings of ACM SIGGRAPH 2004*: p 532-539.
- Znamenáček, J., Valáček, M. “An Efficient Implementation of the Recursive Approach to Flexible Multibody Dynamics” *Multibody System Dynamics*: 2, no. 3 (1998): 227-251.

## 8 Appendix

### 8.1 Taylor Array Size

The size of the array of an m-dimensional Taylor expansion is derived here since the formula is not readily available. The Taylor expansion can be represented compactly as

**Equation 8.1.1**  $f(r) = e^{(r-a)^T \nabla} f(a)$

where the variables are

$$\text{Constant vector } a = (a_0 \ a_1 \ a_2 \ \cdots \ a_{m-1})^T$$

$$\text{Input vector } r = (r_0 \ r_1 \ r_2 \ \cdots \ r_{m-1})^T$$

$$\text{Differential operator vector } \nabla = \left( \frac{\partial}{\partial r_0} \ \frac{\partial}{\partial r_1} \ \frac{\partial}{\partial r_2} \ \cdots \ \frac{\partial}{\partial r_{m-1}} \right)^T$$

The exponential operator is expanded to

**Equation 8.1.2**  $e^y = \sum_{i=0}^{\omega} \frac{y^i}{i!}$

Where  $\omega$  is the terminating limit of approximation and

$$y = (r - a)^T \nabla = \sum_{i=0}^{m-1} (r_i - a_i) \frac{\partial}{\partial r_i}$$

Each term in the expansion of Equation 8.1.2 is a multinomial expansion. There is a recursive relationship for the number of exponential expansion terms

$$\text{Equation 8.1.3 } N_{\omega+1}(m) = \sum_{i=1}^m N_{\omega}(i)$$

Since  $N_1(m) = m+1$  and the fact that

$$\text{Equation 8.1.4 } \sum_{i=0}^m \binom{n+i}{n} = \binom{n+m+1}{n+1} \text{ and } \binom{n+i}{n} = \binom{n+i}{i}$$

leads to the conclusion that

$$\text{Equation 8.1.5 } N_{\omega}(m) = \binom{\omega+m}{m}$$

## 8.2 Inertial Properties of a Tetrahedron

Inertial properties can be quite difficult to calculate but can be exactly determined.

The moment of inertia and angular momentum is taken from standard analytical mechanics books<sup>11</sup> as

$$\text{Equation 8.2.1 } I = \int \|\hat{w} \times r\|^2 dm$$

$$\text{Equation 8.2.2 } L = \int r \times (w \times r) dm$$

Where  $dm = \rho dV$ , the density times the differential volume.

The moment of inertia and angular momentum are very desirable traits to follow during any dynamic simulation. They both involve integrating over the tetrahedral

---

<sup>11</sup> Fowles 1977

Knight 62

volume. I have derived them below using parametrized coordinates. A point inside the tetrahedron can be uniquely determined by

$$\mathbf{Equation 8.2.3} \quad r = a_0 + (a_1 + (a_2 + a_3 t_0) t_1) t_2$$

where

$$a_0 = v_2$$

$$a_1 = v_3 - v_2$$

$$a_2 = v_0 - v_3$$

$$a_3 = v_1 - v_0$$

and  $v_i$  are one of the tetrahedron vertices

The differential volume is determined by the parameter space change formula

$$dV = \frac{\partial r}{\partial t_2} \cdot \left( \frac{\partial r}{\partial t_1} \times \frac{\partial r}{\partial t_0} \right) dt_0 dt_1 dt_2$$
$$\frac{\partial r}{\partial t_2} \cdot \left( \frac{\partial r}{\partial t_1} \times \frac{\partial r}{\partial t_0} \right) 6 t_1 t_2^2 V_{Tet}$$

Each parameter  $t_i$  varies from zero to one for inside the tetrahedron so the entire integral results in

$$\mathbf{Equation 8.2.4} \quad I = 6m \iiint_0^1 \|\hat{w} \times r\|^2 t_1 t_2^2 dt_0 dt_1 dt_2$$

where  $\hat{w}$  is the angular velocity unit vector (i.e. the spin axis) and  $m$  is the mass of the entire tetrahedron. The density of the tetrahedron is kept constant and thus the mass is brought out of the integrand as the density times the volume. Performing the triple integral produces a double sum. The integral turns into the double sum

**Equation 8.2.5**  $I = 6m \sum_{i=0}^3 \sum_{j=0}^3 (\hat{w} \times \vec{a}_i) \cdot (\hat{w} \times \vec{a}_j) \iiint_{0..1} t_0^{n_{0ij}} t_1^{n_{1ij}} t_2^{n_{2ij}} dt_0 dt_1 dt_2$  where

$$n_{kij} = \left\lfloor \frac{i+k}{3} \right\rfloor + \left\lfloor \frac{j+k}{3} \right\rfloor + k$$

Integrating the double sum and expanding the  $a_i$  terms into  $v_i$  terms produces the equation

**Equation 8.2.6**  $I = m \sum_{i=0}^3 \sum_{j=0}^3 q_{ij} a_{ij}$

$$a_{ij} = (\hat{w} \times v_i) \cdot (\hat{w} \times v_j)$$

$$q_{ij} = \begin{cases} \frac{1}{20} & j \neq i \\ \frac{1}{10} & j = i \end{cases}$$

Since both  $q_{ij}$  and  $a_{ij}$  are symmetric then Equation 8.2.6 reduces to ten terms.

The angular momentum is very similar with the integral

**Equation 8.2.7**  $\vec{L} = 6m \iiint_{0..1} \vec{r} \times (\vec{w} \times \vec{r}) t_1 t_2^2 dt_0 dt_1 dt_2$

**Equation 8.2.8**  $L = m \sum_{i=0}^3 \sum_{j=0}^3 q_{ij} b_{ij}$

$$b_{ij} = v_i \times (w \times v_j)$$

In this case,  $b_{ij}$  is not symmetric so all sixteen terms must be calculated. A much simpler form of these equations occurs when a body has a coordinate system that is

Knight 64

centered on its center of mass and is aligned to its principal axes. This simplification will be approached in the next section. The following matrix equations separate out the purely geometric quantity  $Q$  from the spinning quantity  $\hat{w}$ .

**Equation 8.2.9**  $I_{Tet} = m_{Tet} \hat{w}^T Q_{Tet} \hat{w}$

**Equation 8.2.10**  $L_{Tet} = m_{Tet} Q_{Tet} w$

$$Q_{Tet}(r) = Q(C_{mass} - r) + \frac{1}{20} \sum_{i=0}^3 Q(v_i - C_{mass})$$

$$Q(r) = r^T r - r r^T = \begin{pmatrix} r^2 - x^2 & -xy & -xz \\ -yx & r^2 - y^2 & -yz \\ -zx & -zy & r^2 - z^2 \end{pmatrix}$$

The time complexity of  $Q_{Tet}$  is 30 multiplications + 54 additions. For each of the above equations, the vectors are relative to the center of rotation. To generalize, one must subtract the center of rotation from the four vertex vectors.

### 8.3 Inertial Properties of a Rigid Body

A rigid body has the property that it cannot be molded, stretched or deformed. It still can be translated and rotated as a whole. This has advantages because it has a fixed volume, mass, principal moments, principal axes and relative center of mass thereby reducing the calculations for the inertial properties. The fixed values can be calculated at time of creation of the body so it will not be necessary to recalculate during motion. The center of mass and geometric tensor are simple weighted averages. The principal moments and principal axes are much more complicated but only need to be calculated once. The principal moments and axes are calculated by solving the eigensystem  $Ax = \lambda x$  where  $A$  is a symmetric matrix,  $\lambda$  are the eigenvalues (principal moments) and  $x$  are their

corresponding eigenvectors (principal axes). The system is usually solved by iterative techniques like the Jacobi method (Press et al. 1992). In the case of the inertial body, the symmetric matrix  $A$  is the Inertia Tensor, which is the mass times the geometric tensor, in the center-of-mass reference frame. The center-of-mass geometric tensor can be isolated by noting that, for a rigid body, the angular momentum is

$$L_{RB} = \sum L_{Tet_i}$$

where the sought after tensor for the tetrahedral mesh is

**Equation 8.3.1** 
$$Q_{RB}(r) = \frac{1}{m_{RB}} \sum m_i Q_{Tet_i} - Q(C_{mass_{RB}} - r)$$

Each tetrahedron's contribution must be calculated relative to the rigid body's center of mass. The eigensystem equation to solve then becomes

**Equation 8.3.2** 
$$mQ\hat{p} = \lambda\hat{p}$$

which has three real solutions for  $\lambda$  and their corresponding  $p$ . The eigenvectors  $p$  are orthogonal<sup>12</sup> and therefore any vector can be represented with components in the three directions. The angular velocity vector is thus

$$w = \sum_{i=0}^2 w_i \hat{p}_i$$

where  $w_i = w \cdot \hat{p}_i$

Using this solution for a rigid body greatly simplifies the calculation efforts for the dynamically changing moment of inertia and angular momentum. The formulae become

---

<sup>12</sup> Dettman, 1988

$$\text{Equation 8.3.3 } I = \sum_{i=0}^2 \hat{w}_i^2 \lambda_i + m \hat{w}^T Q (\bar{C}_{mass} - \bar{r}) \hat{w}$$

$$\text{Equation 8.3.4 } \bar{L} = \sum_{i=0}^2 w_i \lambda_i \hat{p}_i + m Q (\bar{C}_{mass} - \bar{r}) \bar{w}$$

The complexity of these equations no longer depend on the number of tetrahedrons that make up the rigid body. As long as the center-of-mass  $C_{mass}$  and  $\hat{p}$  are continually updated as the body moves then this formula can be used directly.

## 8.4 User's Guide for Macintosh Application


The following sections describe the use of the application that was built for the research. The application was designed using the Xcode 1.5 integrated development environment available for free from Apple and comes with the MacOSX 10.3 operating system. There are 41558 lines of code with a total McCabe's Cyclomatic Complexity Number of 7503. The application follows the recommended Macintosh human interface and is compatible with MacOSX 10.1, 10.2, and 10.3 though only tested on 10.3. The engine that drives the calculations are written entirely in standard portable C++ and the graphics panes are written in OpenGL. The user interface is written in ObjectiveC that compliment the NIB files for Interface Builder. Everything except the user interface is portable. The code was compiled for the G4 processor and was tested on a PowerBook G4.




### **8.4.1 Menu**

The File Menu provides the basic file saving and retrieving capabilities some of which are available in the Options Pane as well.

#### **Add Figure**

The Add Figure menu item (optionally use -1) allows the user to add a figure to the graphics. A figure can be an Articulated Tetrahedral Mesh (ATM) file (as explained in this Appendix); a MESH formatted file (both tetrahedral and triangulated); and a PLY formatted file. Only the ATM file can be used for the hierarchical analysis of the motion capture data. When this menu item is picked, a standard Open dialog is presented for choosing a file. When a file is chosen, the file is parsed and presented in the Animation Pane. The Options Pane text for the figure is filled out with the path of the file and the height of the figure is filled out.

#### **Get Data**

This menu item (optionally use -2) will present a standard Open dialog to allow the user to choose a C3D motion capture data file. The selected file will associate the data with the currently active figure (see the Options Pane). Once chosen, the file is parsed and connected lists of data frames are filled out.


#### **Save Temp Correlation**

This menu item allows the user to save a temporary correlation text file associated with the C3D data file that was previously chosen. The text file can then be edited to allow a hierarchical correlation of the data markers with segments on the figure. The number of markers are stated on the first line (e.g. 41 WANDS). The marker set file name is stated on the second line (e.g. MARKERSET vicon512.txt). Then comes a list of

Knight 68

marker correlations (e.g. Jim:RTHI = RTHI Right thigh). The left side of the equals sign is the single word that the marker is labeled as in the data (e.g. Jim:RTHI). The right side has a single word that corresponds to a marker in the above mentioned marker set file followed by a free-form description.

### **Get Correlation**

This menu item (optionally use -3) presents the user with an Open dialog to retrieve a correlation file. The file is parsed and the left side is looked up in the C3D data and the right side is looked up in the marker set file. The marker set file determines which segment to attach the marker and where on the segment. If the marker is not to be used in the correlation, just put some uncorrelated name (e.g. Unknown) on the right hand side.

### **Get Motion**

This menu item retrieves motion files that have been previous saved or manually created. Motion files allow the user to create motion of an articulated figure based on a Taylor expansion of any or all degrees of freedom.

### **Save Motion**

This menu item will save the Taylor expansions that were created when the “Analyze” button is pressed in the Graphs Pane.

### **Save Data**

This menu item will save the joint angles that were created when the “Analyze” button is pressed in the Graphs Pane.

### **Animate**

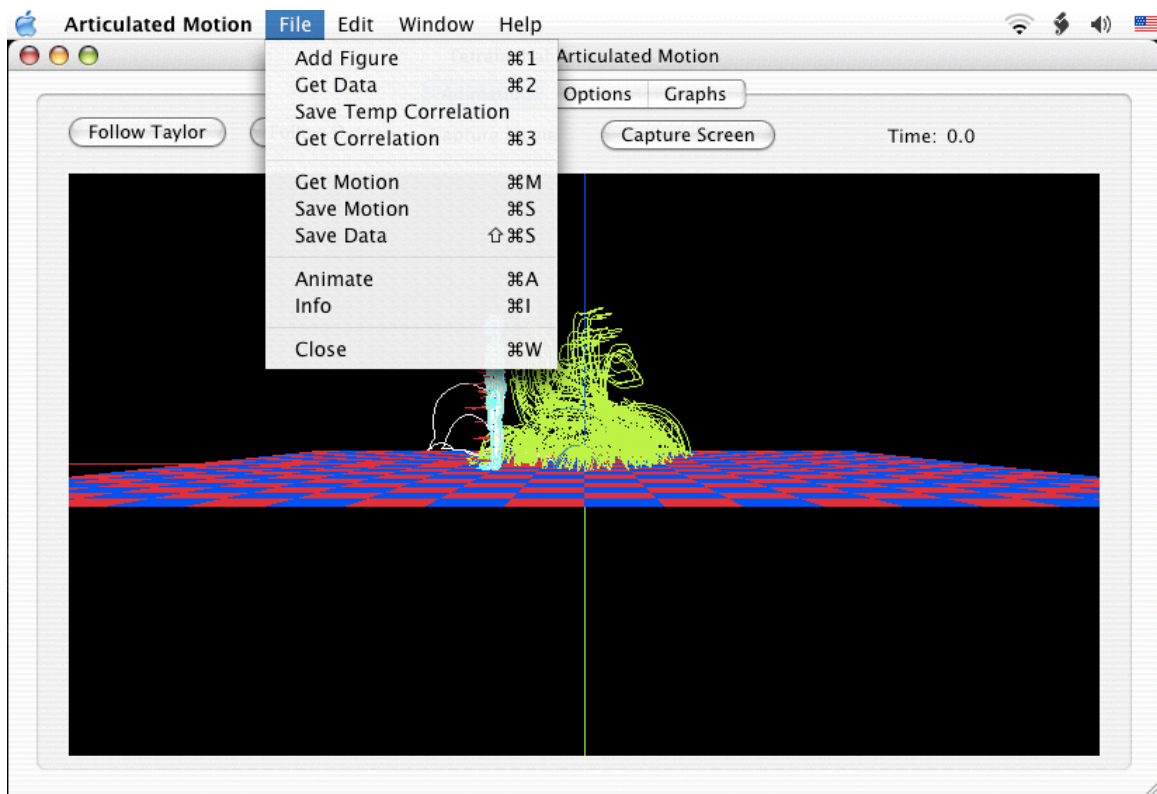
This menu item will start the animation.

### Info

This menu item will enable some graphics hardware information to be displayed on the Animation Pane.

### Close

This menu item will close the window.



### 8.4.2 Options Pane

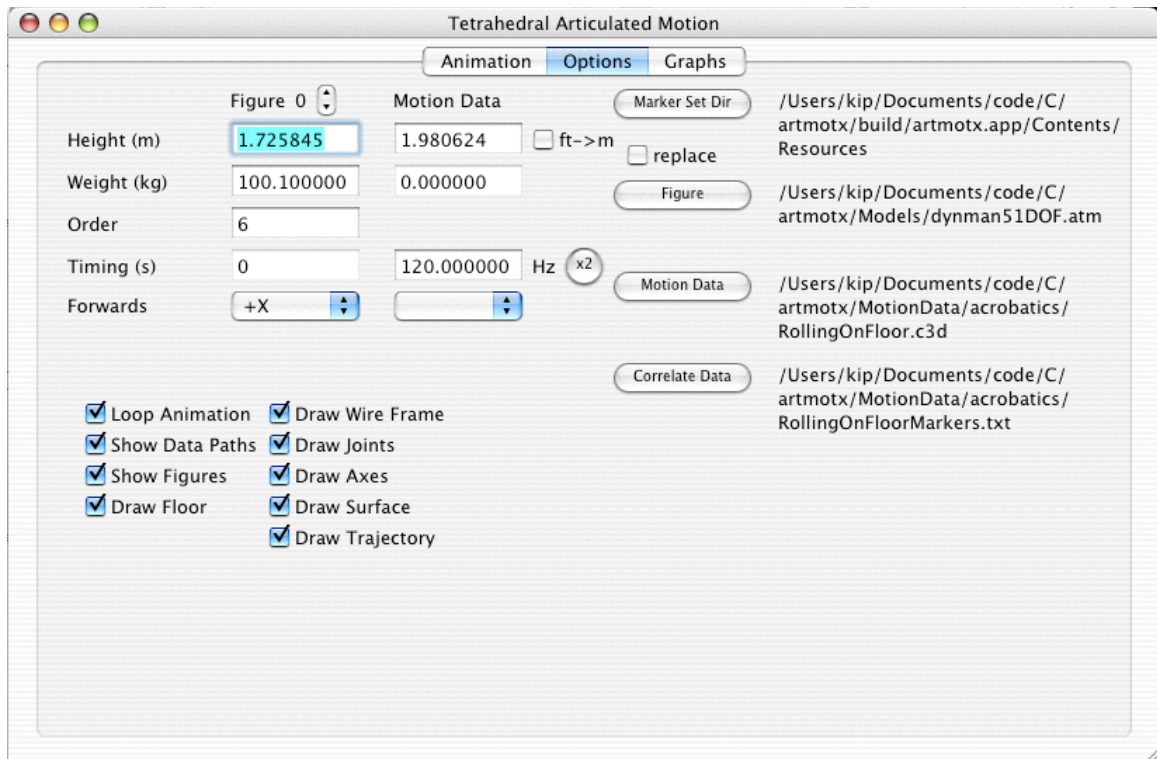
The Options pane contains all of the figure, data, and animation options available. On the right are all of the files that are used in the current analysis. The

Knight 70

“Marker Set Dir” is the directory that all marker set files are located for the correlation. The “Figure” file is the currently active articulated model. The “Motion Data” file is the current C3D being analyzed. The “Correlate Data” file is the correlation file that binds the data with the hierarchical figure. The “replace” checkbox will replace the current figure with the opened figure if it is on. Otherwise the opened file will be added to the scene. The “ft->m” checkbox will convert the motion data from feet to meters if on. The “x2” button will multiply the data sampling rate by two. The last two buttons are there for convenience since the data does not always specify the correct units or sampling rate. Once the data is read in, the height and sampling rate text can be edited to whatever size and the internal data will be updated. The same goes for the Figure information. The height of the figure can be edited.

### **Drawing Options**

The many checkboxes allow the selection of various things to be drawn in the Animation Pane. “Loop Animation” will make the animation repeat when the maximum time of the data is reached. “Show Data Paths” will display the paths taken for each marker in the data. The paths will be colored lighter if the marker has been correlated to the figure. “Show Figures” will display all figures that have been read in. “Draw Floor” will ... draw the floor?! The other checkboxes draw extra little goodies on the figure.

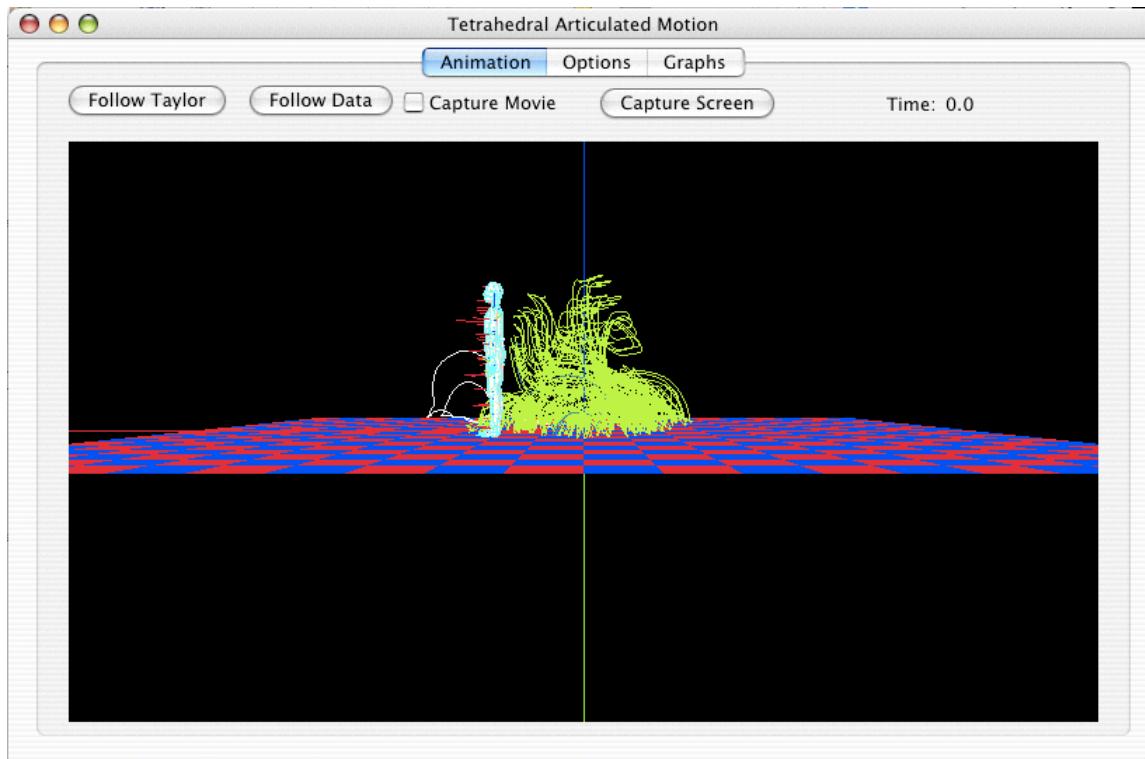


### 8.4.3 Animation Pane

This pane is where all of the action happens. Select this pane once all of the files and options are set in the Options Pane. The time of the animation is displayed in the upper right in seconds. The Capture Screen button will take an exact snapshot of the OpenGL window and allow the user to save it as a TIFF file. The TIFF file is a 32-bit ABGR8888 color file. The “Capture Movie” checkbox is similar except that it saves many TIFF files at 0.1 second intervals for the entire loop. The “Follow Data” starts the animation process which involves draw the moving figure and the skeleton in the data. There are options for panning the scene around. If the Command key (⌘) is held down while the mouse button is held down and the screen is dragged by the mouse, then the scene is dragged up/down or left/right. If the Option key is held down with the mouse

Knight 72

button, the user can zoom in or out of the scene with the mouse. With just the mouse button and the mouse dragging, the user can rotate the scene around the (0,0,0) point.



#### ***8.4.4 Graphs Pane***

This pane was added for analysis of joint angles was just a visualization aid for research. The analyze button currently were try to determine angles needed to rotate in order to follow the current data-set. Once the analysis is done, a joint can be picked from the pop-up. When the user slides the horizontal time bars left and right, a new Taylor expansion will be generated for the time window.



## 8.5 Specification for Articulated Tetrahedral Model Format

The Articulated Tetrahedral Model (ATM) file format was created in order to bring together the necessary information to produce a jointed figure. It is a text format that references other the individual segments' mesh files. The text file has keywords followed by values and sub-fields. All words are separated by spaces. The following are the list of available keywords:

**FIGURE** myFigure

This keyword defines the name of the entire figure.

**SEGMENTS** 16

The number of segments of the figure follows:

Knight 74

```
MESH 0 chest.mesh
```

MESH identifies the mesh file that follows the index value. The file is identified as a tetrahedral or triangular mesh in the \*.mesh format defined by the freeware MEDIT tool for editing meshes. The filename is for a file that is located in the same folder as the ATM file.

```
NAME Chest
```

The name of the segment follows this keyword.

```
MASS 30.0 // kg
```

The mass of the segment follows this keyword.

```
SCALE_XYZ 0.2 0.2 1.0  
TRANSLATE_X 1.5  
TRANSLATE_Y 1.5  
TRANSLATE_Z 1.5  
TRANSLATE_XYZ 1.5 2.5 3.5  
ROTATE_X 35.2  
PARENT 4 // hips
```

The index of the parent segment is defined here.

```
JOINT Waist  
3 0.994765 1.002953 1.084350  
DEGREES_OF_FREEDOM 1 0  
VALUE 0.0  
3 -0.257042 0.966392 -0.003915  
FROM -93.199994 TO 29.000000
```

The name of the joint that is proximal to this segment is defined here followed by joint parameters. The center of rotation is defined immediately following the JOINT keyword. First a count of how many dimensions for the vector is specified (usually 3).



Then, each coordinate, e.g. x y z. After the joint position, its freedoms are defined. DEGREES\_OF\_FREEDOM is followed by two integers. The first is the count of rotational freedoms, and the second is the count of translational freedoms. Then comes the list of freedom parameters for which there are three for each freedom. The current value is defined, then the freedom axis, then the freedom limits. Rotational limits and values are in degrees and translational ones are in meters.

```
CHILDREN 3
  1 // Neck
  2 // Upper Left Arm
  3 // Upper Right Arm
```

The list of child indices are defined here

```
ENDMESH
```

Mandatory ending of the segment information

```
ENDFIGURE
```

Mandatory ending of the figure information

## 8.6 Proof of Positive-Definite Matrix for Rotation Point

Positive-definite is defined as  $h^T A h > 0$  for an arbitrary vector  $h \neq 0$ . The matrix of interest is

$$A = \frac{1}{N} \sum p_i (p_i^T - \bar{p}^T)$$

This can be re-arranged as a double sum to

$$A = \frac{1}{N^2} \sum_i \sum_j p_i (p_i^T - p_j^T)$$

Then applying an arbitrary vector  $h$

Knight 76

$$h^T Ah = \frac{1}{N^2} \sum_i \sum_j h^T p_i (p_i^T h - p_j^T h)$$

Now defining  $q_i = h^T p_i$  simplifies the double sum

$$h^T Ah = \frac{1}{N^2} \sum_i \sum_j q_i (q_i - q_j)$$

This further simplifies to

$$h^T Ah = \overline{q^2} - \bar{q}^2$$

which is always positive for an arbitrary set of values  $q_i$  except for one non-trivial case.

The exception is when the  $p_i$  are coplanar. In the coplanar case, all  $q_i$  are the same constant when  $h$  is the normal to the plane and therefore  $h^T Ah = 0$ .

## 8.7 Relevant C++ Implementations

### 8.7.1 Minimum Variance Method

```
void Segment::GetDataRotationPoint()
{
    Event * e = NULL;
    unsigned int j;
    Vector p(3),p3(3);
    Vector b,r,v,p_i;
    Matrix A(3,3),P2(3,3);
    double p2 = 0.0,cn;

    if( CollectRelativeDataPoints() )
    {
        e = pathToFollow.beginning;
        while( e )
        {
            p_i = e->position;
            p += p_i;
            P2 += p_i.MultTranspose(p_i);
            p2 = p_i*p_i;
            p3 += p2*p_i;
        }
    }
}
```

```

    e = e->next;
}
if( pathToFollow.count )
{
    j = pathToFollow.count;
    P2 /= j;
    p /= j;
    p3 /= j;
    p2 = P2.Trace();

    b = 0.5*(p3 - p2*p);
    A = P2 - p.MultTranspose(p);

    // find answer based on the idea that the matrix A
    // is positive-definite
    r = A.CholeskyDecomposition().CholeskyBacksubstitution(b);

    // get Null vector based on a threshold
    // that is a little above the inverse of
    // the condition number
    cn = A.ConditionNumber();

    v = A.NullSpace(1.000001/cn).Column(0);
    relativeDataParentOneDOFaxis = DirectionVector(v);
    // project the answer onto plane if
    // the condition number is large "enough"
    if( cn > 100.0 )
    {
        r = r + v*((p-r)*v);
    }
}
}
relativeDataParentRotationPoint = PositionVector(r);
}

```

### 8.7.2 Rotation Point Calculation of Hierarchical Articulated Data

```

void Segment::SetDataRelativeStuff() // call from root
{
    Matrix dataAxes = Matrix::Identity(3);
    PositionVector dataCenter,rot,c,v;
    unsigned int i;
    Units::Time::second t = 0.0;
    Event * e[numCorrelated];

```

```

// calculate the data centroid
unsigned int n = 0, n1=numCorrelated;
if( correlatedFrame.NextFrame(n1,e,t) )
{
    c = 0.0;
    for( i=0; i<numCorrelated; i++ )
    {
        if( e[i] )
        {
            c += e[i]->position;
            n++;
        }
    }
    if( n )
        c /= n;
}
if( parent )
{
    GetDataRotationPoint();
    if( parent->numCorrelated == 0 ) // set from parent's
        parent->relativeDataParentRotationPoint =
relativeDataParentRotationPoint;
    t = 0.0;
    if( parent->GetDataAxes(t,dataAxes,dataCenter) )
    {
        rot = dataCenter +
dataAxes*relativeDataParentRotationPoint;
        v = dataAxes*relativeDataParentOneDOFaxis;
    }
    else
    {
        cerr << "No parent data axes for " << name << endl;
        rot = PositionVector(0.0,0.0,0.0);
        v = PositionVector(1.0,0.0,0.0);
    }
    t = 0.0;
    if( GetDataAxes(t,dataAxes,dataCenter) )
    {
        relativeDataRotationPoint = (rot - dataCenter)*dataAxes;
        relativeDataCentroid = (c - dataCenter)*dataAxes;
        relativeDataOneDOFaxis = dataAxes.Transpose()*v;
    }
    else
    {
        cerr << "No self data axes for " << name << endl;

```

```

    }
}
else // root
{
    t = 0.0;
    if( !GetDataAxes(t,dataAxes,dataCenter) )
        cerr << "Damn, root really needs >= 3 correlated markers"
            << endl;
    relativeDataCentroid = (c - dataCenter)*dataAxes;
    relativeDataRotationPoint = relativeDataCentroid;
    relativeDataOneDOFaxis = DirectionVector(1.0,0.0,0.0);
    relativeDataParentOneDOFaxis = DirectionVector(1.0,0.0,0.0);
    relativeDataParentRotationPoint = PositionVector(0.0,0.0,0.0);
}
for( i=0; i<numChildren; i++ )
    children[i]->SetDataRelativeStuff(); // recursive
}

```

### 8.7.3 Calculation of fixed axes of data

```

bool Segment::GetDataAxes( Units::Time::second& t,
                           Matrix& dataAxes,
                           PositionVector& dataCenter ) const
{
    Event * e[3] = {NULL,NULL,NULL};

    // a segment without axes data, use parent's
    if( parent && axesFrame.numElements == 0 )
    {
        if( !parent->GetDataAxes( t, dataAxes, dataCenter ) ) //
recursive
            return false;
    }
    // for root and all segments with many points gets in here
    else if( axesFrame.numElements == 3 )
    {
        if( !axesFrame.NextAbsoluteFrame(3,e,t) )
        {
            if( t < axesFrame.maxTime )
                cerr << "Missing events for " << jointName << " t " <<
t << endl;
            return false;
        }
        dataCenter = e[0]->position;
    }
}

```

Knight 80

```
        dataAxes = Matrix::Axes(dataCenter,e[1]->position,e[2]-
>position);
    }
    // segment must have parent beyond this point
    else if( !parent )
    {
        return false;
    }
    else if( axesFrame.numElements == 2 )
    {
        Matrix pDataAxes;
        PositionVector pDataCenter;
        if( !axesFrame.NextAbsoluteFrame(2,e,t) )
            return false;
        if( !parent->GetSameDataAxes( t, pDataAxes, pDataCenter ) ) //
recursive
            return false;
        dataCenter = pDataCenter + pDataAxes *
relativeDataParentRotationPoint;
        dataAxes = Matrix::Axes(dataCenter,e[0]->position,e[1]-
>position);
    }
    else if( axesFrame.numElements == 1 )
    {
        Matrix pDataAxes;
        PositionVector pDataCenter,r2;

        if( !axesFrame.NextAbsoluteFrame(1,e,t) )
            return false;
        if( !parent->GetSameDataAxes( t, pDataAxes, pDataCenter ) ) //
recursive
            return false;
        dataCenter = pDataCenter + pDataAxes *
relativeDataParentRotationPoint;
        r2 = dataCenter + pDataAxes*relativeDataParentOneDOFaxis;
        dataAxes = Matrix::Axes(dataCenter,r2,e[0]->position);
    }
    else
        return false;
    return true;
}
```

#### 8.7.4 Drawing Rotation Points with Constants of Motion

```
void Segment::DrawNearestEvents( double radius, Units::Time::second t )
```

```

{
    unsigned int i;
    Matrix dataAxes,dataAxes0,dataAxes1;
    PositionVector dataCenter,dataCenter0,dataCenter1,rot,pos,ep;

    Color::White.glColor();
    correlatedFrame.Draw(radius*0.5,t);
    t -= 0.00001;
    if( parent && parent->GetDataAxes( t, dataAxes0, dataCenter0 ) )
    {
        rot = dataCenter0 + dataAxes0*relativeDataParentRotationPoint;
    }
    else if( parent != NULL )
    {
        return;
    }
    if( GetDataAxes( t, dataAxes, dataCenter ) )
    {
        if( parent == NULL )
            rot = dataCenter + dataAxes*relativeDataCentroid;
    }
    else
    {
        return;
    }
    glBegin(GL_LINES);
    for( i=0; i<numChildren; i++ )
    {
        rot.glVertex();
        (dataCenter + dataAxes*children[i]-
>relativeDataParentRotationPoint).glVertex();
    }
    if( numChildren == 0 )
    {
        rot.glVertex();
        (dataCenter + dataAxes*relativeDataCentroid).glVertex();
    }
    glEnd();
    if( numChildren == 0 )
        correlatedFrame.DrawLines(t);
}

```

## 8.8 Kalman Filter

The Kalman filter was developed in the 1960's and has been proven successful in describing complex motion. It has been used in a wide variety of applications including tracking missiles from radar measurements to sensor calibration for motion capture systems. The complexity of defining the process and measurement errors before-hand makes this system unwieldy when solving a problem with no a-priori knowledge. The implementation here attempts to solve the problem of finding the point of rotation of a joint. Upon implementing this, it was found that the Kalman filter was not robust and found the solution in only a small percentage. It tended to converge to a solution that had some bias or outright diverge to infinity. Presented in this section are the equations used to attempt the Kalman Filter solution to the rotation point.

### 8.8.1 Theory

Each measurement  $p$  is assumed to be of the form

$$p_i = r + L_i \rho_i \text{ where}$$

$$\rho_i = \begin{pmatrix} \cos \theta_i \cos \phi_i \\ \sin \theta_i \cos \phi_i \\ \sin \phi_i \end{pmatrix}$$

The Kalman Filter attempts to correct a solution by compensating for a-priori knowledge of the noise and a new measurement. The measurement vector  $z$  is used to recalculate the current state  $x$  which is composed of both constants of motion as well as changing variables. Each measurement has the same rotation point and different lengths





Knight 84

A new guess of the process variables are calculated from iterating over the following equations:

$$x_k = Ax'_{k-1}$$

$$C_k = AC'_{k-1}A^T$$

$$z'_k = Hx_k$$

$$G = (HC_kH^T)^{-1}C_kH^T$$

$$x'_k = x_k + G(z_k - z'_k)$$

$$C'_k = (1 - GH)C_k$$

This is the simple linear model for Kalman filtering and relies on the initial guess of the state variables and the error covariance matrix  $C$ . Each iteration will produce a refinement of the process variables in  $x$  and the covariance matrix will tend towards zero. Once it converges, the rotation point is realized inside the vector  $x$ . There were too many cases of divergence during the investigation so the method was not continued.