

A VIRTUAL HUMAN ANIMATION TOOL USING MOTION CAPTURE DATA

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

SELİM NAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF MEDICAL INFORMATICS

JULY 2008

Approval of the Graduate School of Informatics

Prof. Dr. Nazife Baykal
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Erkan Mumcuođlu
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Yasemin Yardımcı
Supervisor

Examining Committee Members

Assoc. Prof. Dr. Erkan Mumcuođlu (METU, MIN) _____

Prof. Dr. Yasemin Yardımcı (METU, IS) _____

Assist. Prof. Dr. Tolga Can (METU, CENG) _____

Assist. Prof. Dr. Didem Gökçay (METU, MIN) _____

Assist. Prof. Dr. Alptekin Temizel (METU, WBL) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Selim Nar

Signature : _____

ABSTRACT

A VIRTUAL HUMAN ANIMATION TOOL USING MOTION CAPTURE DATA

NAR, Selim

M.S., Department of Medical Informatics

Supervisor: Prof. Dr. Yasemin YARDIMCI

July 2008, 100 pages

In this study, we developed an animation tool to animate 3D virtual characters. The tool offers facilities to integrate motion capture data with a 3D character mesh and animate the mesh by using Skeleton Subsurface Deformation and Dual Quaternion Skinning Methods. It is a compact tool, so it is possible to distribute, install and use the tool with ease.

This tool can be used to illustrate medical kinematic gait data for educational purposes. For validation, we obtained medical motion capture data from two separate sources and animated a 3D mesh model by using this data. The animations are presented to physicians for evaluation. The results show that the tool is sufficient in displaying obvious gait patterns of the patients.

The tool provides interactivity for inspecting the movements of patient from different angles and distances. We animate anonymous virtual characters which provide anonymity of the patient.

Keywords: Virtual Human Animation, Motion Capture, Medical Animation, Human Gait

ÖZ

HAREKET YAKALAMA VERİSİ KULLANARAK SANAL İNSAN CANLANDIRMA ARACI

NAR, Selim

Yüksek Lisans, Tıp Bilişimi

Tez Yöneticisi: Prof. Dr. Yasemin YARDIMCI

Temmuz 2008, 100 sayfa

Bu çalışmada, üç boyutlu sanal karakterleri canlandırmak için bir animasyon aracı geliştirdik. Araç, hareket yakalama verisini ve 3B karakter modellerini entegre edebilecek, dahası "Skeleton Subspace Deformation" ve "Dual Quaternion Skinning" metotlarını kullanarak sanal karakterleri canlandırabilecek olanakları sunmaktadır. Aracın küçük ve etkili bir araç olarak geliştirilmesi, dağıtılması, kurulması ve kullanılmasında kolaylık sağlamaktadır.

Araç, eğitimsel amaçlar için medikal kinematik yürüyüş verisinin görselleştirilmesinde kullanılabilir. Doğrulama amacıyla, iki ayrı kaynaktan medikal hareket yakalama verisi elde edip, bu veri üzerinden 3B bir sanal karakteri canlandırdık. Bu animasyonlar yürüme uzmanı olan doktorlara ölçme ve değerlendirme amacı ile gösterildi. Sonuçlar, aracın hastalara ait bariz yürüme bozukluklarını göstermede yeterli olduğunu ortaya koydu.

Geliştirilen araç, hasta hareketlerinin farklı açı ve mesafelerden etkileşimli olarak incelenmesine olanak vermektedir. Animasyonlarda, 3B anonim karakterlerin kullanımı hasta mahremiyetini sağlamaktadır.

Anahtar Kelimeler: Sanal İnsan Canlandırma (Animasyonu), Hareket Yakalama, Tıpta animasyon, İnsan Yürüyüşü

ACKNOWLEDGMENTS

I express sincere appreciation to Prof. Dr. Yasemin Yardımcı for her guidance and insight throughout the research. Thanks go to our faculty secretary Sibel Gülnar and my colleague Berna Bakır for their spiritual support. The suggestions and comments of Assoc. Prof. Dr. Güneş Yavuzer and Assoc. Prof. Dr. Haydar Gök are gratefully acknowledged.

TABLE OF CONTENTS

ABSTRACT	iv
ÖZ	v
ACKNOWLEDGMENTS	vi
TABLE OF CONTENTS	vii
LIST OF TABLES	ix
LIST OF FIGURES	x
LIST OF ABBREVIATIONS	xii
CHAPTER	
1. INTRODUCTION	1
1.1 Overview	1
1.2 Deformation Methods	2
1.2.1 Geometric Deformation Methods	3
1.2.2 Physical Deformation Methods	4
1.3 Character Animation Methods and Skeleton Animation	7
1.3.1 Geometric Methods and Skeleton Subspace Deformation	7
1.3.1.1 Skeleton Subspace Deformation	8
1.3.1.2 Other Geometric Methods	9
1.3.2 Example-Based Methods and Shape Interpolation	15
1.3.3 Physical-Based Methods	17
2. REPRESENTATIONS: QUATERNIONS AND DUAL-QUATERNIONS	23
2.1 Overview	23
2.2 Orientation Representations	23
2.3 Quaternions	25
2.4 Dual Quaternions	27
2.4.1 Properties of Dual Quaternions	28
2.4.2 Rotation with Dual Quaternions	29
2.4.3 Translation with Dual Quaternions	31
2.4.4 Dual Quaternions Representing Both Rotation and Translation	33
2.4.5 Behaviors of Dual Quaternion Transformations	35
3. SKELETON ANIMATION WITH SKELETON SUBSPACE DEFORMATION AND DUAL- QUATERNIONS SKINNING	37
3.1 Overview	37
3.2 Articulated Skeleton Structure	37
3.3 Skeleton Subspace Deformation	40
3.4 Dual-Quaternions Skinning	46
4. ANIMATING THE SKIN AND ANIMATION DATA	51
4.1 Overview	51
4.2 Key-Framing and Frame interpolation	52
4.3 Animation Data	54
4.4 Motion Capture	55
4.5 Motion Capture Systems	56
5. FEATURES OF THE ANIMATION TOOL	58
5.1 Overview	58
5.2 Interaction with Objects in our Animation Tool	58

5.2.1 Mouse Selection in the 3D Scene	60
5.2.2 Mouse Drag Action.....	60
5.3 Animation Infrastructure of the Tool.....	62
5.3.1 Animation Using Input Files.....	63
5.3.2 Deformation by User Input	63
5.4 Moving Over the Scene	64
5.5 Calculating Bone Weights	65
5.5.1 Calculating Vertex Bone Distances	66
5.5.2 Bone Weight Calculation Algorithm	67
5.6 Reading and Animating Raw Mocap File	67
5.6.1 Calculating Bone Transformations	68
5.6.2 Outcomes of our Bone Transformation Calculation	69
5.6.3 Alternative Solution for Bone Transformation Calculation.....	70
5.7 Calculating Skin Offset Transformations	71
6. RESULTS, CONCLUSIONS AND FUTURE DIRECTIONS	73
6.1 Outcomes of the Animation Tool	73
6.1.1 Motion Capture Process.....	74
6.1.2 Acquiring Bone Transformation in MotionBuilder	74
6.1.3 Final Outcomes.....	75
6.2 Validation of the Tool	77
6.2.1 Brief Information about the Human Gait.....	77
6.2.2 Validation Methodology	79
6.3 Discussions	88
6.4 Conclusion.....	88
6.5 Future Directions	89
6.5.1 Improvements for Motion Capture Data.....	89
6.5.2 Improving Existing Algorithms and Integrating New Ones	90
6.5.3 Improving User Interface.....	90
6.5.4 Platform Independency	91
6.5.5 Performance Issues	91
6.5.6 Web-Based Animation Tool	91
REFERENCES.....	92
APPENDICES	
A: DEFINITIONS OF GAIT DISORDERS.....	96
A.1 Multiple Sclerosis (MS).....	96
A.2 Polio.....	96
A.3 Cerebral palsy	96
A.4 Osteoarthritis.....	97
B: PROGRAM CODES	98
B.1 OpenGL Mouse Selection Code with C++	98
B.2 Mouse Drag Code with C++	99
B.3 Calculation of Line Plane Intersection with C++.....	99
B.4 Calculation of Rotation Quaternion between Two Vectors with Matlab Script.....	100

LIST OF TABLES

Table 1 Quaternion component multiplication.....	26
Table 2 Animation key frames of bone defined by quaternion rotation, scaling and translation.....	53
Table 3 Descriptions of Joint Angles	79
Table 4 The answers of PMs for each video	80
Table 5 Ratings of all joints on sagittal and coronal plane for video A	81
Table 6 Ratings of all joints on sagittal and coronal plane for video B.....	81
Table 7 Ratings of all joints on sagittal and coronal plane for video C.....	81

LIST OF FIGURES

Figure 1 A mapping for twist along y axis	3
Figure 2 FFD applied to an 3D object.....	4
Figure 3 A cubic spline curve which is defined by CP P1, P2, P3, and P4.....	4
Figure 4 A mass-spring model. Each mass node put force on its neighbor nodes and all of the system	5
Figure 5 A plane which is deformed by wires method.....	10
Figure 6 Bending and twisting	11
Figure 7 Spline Coordinate System - Frenet-frames	11
Figure 8 A bone curve derived from two standard bones.....	12
Figure 9 SSD (left) and log-matrix (right) comparison	13
Figure 10 Linearized Hill-type model. (a) Force-length relation, (b) Force-velocity relation	18
Figure 11 (a) 1D Mass-spring model applied on human skeleton, (b) Spring-mass, (c) Spring-mass-damper (Courtesy of (Aubel & Thalmann, 2000))	19
Figure 12 (a) Smooth skinning, (b) Anatomic deformation and (c) Underlying musculature.....	20
Figure 13 Behavior of the various muscle models with flexion at the elbow joint and resulting skinning (Courtesy of (Ferdin, et al., 1997))	21
Figure 14 (a) Skeleton and surface, (b) Skeleton and muscles, (c) Skin	21
Figure 15 Ellipsoid musculature model with different poses	22
Figure 16 Halfway interpolation of (a) and (c) results in (b)	24
Figure 17 Euler's angle	24
Figure 18 A point rotated around an axis by angle α	25
Figure 19 Rotation of a triangular prism around an axis in 3D space	30
Figure 20 In (a) there is only translation and in (b) rotation around the axis is added to same translation.....	32
Figure 21 (a) shows rotation only and (b) shows both translation and rotation	35
Figure 22 (a) and (b) shows the same screw movement from different angles	35
Figure 23 (a) and (b) shows the effects of both translation, rotation and axis change	36
Figure 24 A sample bone hierarchy	38
Figure 25 Step by step evaluation of the recursive skeleton formation.....	39
Figure 26 Skeletal configuration in the third recursion step with new bone lengths.....	40
Figure 27 Use of SOT	41
Figure 28 Weighted Blending	43
Figure 29 Collapsing Defect	44
Figure 30 Collapsing defect when upper arm twisted 180 degrees	45
Figure 31 A collapse is observed as leg twists 180 degrees	45
Figure 32 QLERP vs. SLERP	48
Figure 33 Note the volume loss and collapses in SSD (a) and compared to DQS (b).....	50
Figure 34 Structure of an Animation Production	52
Figure 35 Optical mocap data used in games such as Devil May Cry (a) and Heavenly Swords (b) ..	56
Figure 36 Summary of steps from motion capture to validation	59
Figure 37 Viewing frustum	61
Figure 38 A drag event along x axis on elbow joint from (a) to (b).....	62
Figure 39 (a) is a walking animation sequence and (b) is a jogging animation sequence	62
Figure 40 By using direct user input, model is deformed from its left upper arm and and left forearm. Initial model is displayed in (a) and manually deformed one is in (b)	64

Figure 41 Same scene is displayed from different camera views (a) and (b).....	65
Figure 42 Skeleton is aligned in T-Pose to the transparent model in T-Pose.....	65
Figure 43 Same scene is displayed from different camera views (a) and (b).....	66
Figure 44 from given joint positions in (a) each bone is transformed back to its local frame, joint111 is transformed to its local coordinate in (e).....	69
Figure 45 Rotation is applied to (a) spine joint translated the skeleton to pose (b).....	70
Figure 46 Outcomes of our bone transformation calculation algorithm.....	70
Figure 47 The affect of SOT	71
Figure 48 A screenshot from the marker tracking tool developed in Koç University.	74
Figure 49 An actor is shaped and oriented to be aligned with the markers as much as possible. Displayed from different views (MotionBuilder).....	75
Figure 50 When rotated from (a) to (b), upper arm causes defects at the side of the chest because of the our weight assignment algorithm	76
Figure 51 Outcome of the mug Alzheimer animation is shown for different moments	76
Figure 52 Outcome of the multiple sclerosis (MS) animation is shown for different moments.....	77
Figure 53 Anatomical planes of the human body.....	78
Figure 54 (a) Lower body joints and (b) hip rotation axes	78
Figure 55 Human Gait Cycle	79
Figure 56 MotionBuilder skeleton scaling transformation by mouse is shown in (a) and rotation shown in (b).....	90

LIST OF ABBREVIATIONS

CGI	: Computer Generated Image.
CAD	: Computer Aided Design
CAGD	: Computer Aided Geometric Design
FFD	: Free Form Deformation
NURBS	: Non-Uniform Rational B-Splines
CP	: Control Point
2D	: Two Dimensional
3D	: Three Dimensional
FEM	: Finite Element Methods
Mocap	: Motion Capture
Mocap Lab	: Motion Capture Laboratory
SSD	: Skeleton Subspace Deformation
PSD	: Pose space deformation
LOD	: Level of Detail
RBF	: Radial Basis Function
SIM	: Shape interpolation methods
SVD	: Singular Value Decomposition
PCA	: Principal Component Analysis
MWE	: Multi-weight Enveloping
SWE	: Single-weight Enveloping
GDM	: Geometric Deformation Methods
LBS	: Linear Blend Skinning
MPS	: Matrix-palette Skinning
B-spline	: Bezier Spline Curve
SBS	: Spherical Blend Skinning
SLERP	: Spherical Linear Interpolation
QLERP	: Quaternions Linear Interpolation
DQS	: Dual Quaternion Skinning
DQ	: Dual Quaternions
ScLERP	: Screw Linear Interpolation
DLB	: Dual Quaternion Linear Blending
DIB	: Dual quaternion Iterative Blending

API : Application Programming Interface
SOT : Skin Offset Transformation

CHAPTER 1

INTRODUCTION

1.1 Overview

Human body animation has always played a major role in Computer Graphics. It is one of the musts to generate a virtual human. Achieving a fully realistic human body animation is an open research area since body is an extremely complex design which is hard to model and implement. Unfortunately, in computer graphics realism is not the only issue. Some applications of computer graphics require interactive speed. Because of this, researchers must also put computational complexity into consideration while developing a new algorithm. This also applies to human body animation.

Despite all of these problems, virtual humans are more common in our daily lives. We see them in movies, commercials, documentary films, etc. We play as one of them in computer games, experiment with them in simulation softwares. Medicine is another field where virtual humans are used. Since, it is easier to comprehend some concepts with illustrations and simulations, use of virtual humans became a necessity for training or sharing knowledge in medicine. Nowadays, a surgeon is able to experiment and train within a virtual simulation environment with a virtual human using force back devices. There are several highly detailed pre-rendered videos of human animation for illustrative purposes. Every body has a chance to witness a baby moving in the uterus on television, thanks to Computer Generated Image (CGI). Human movement is a subject of biomechanics. Orthopedists, neurologists, and even psychologists who we will refer as gait analysts in our thesis make use of visual inspection on human movement for diagnosing a disease or disorder. Such data can be acquired by using a motion capture system. Software companies and researchers work on developing software packages and expert systems which can capture and convert the motion capture data into more interpretable forms, charts, summaries, reports etc. These packages are indeed very beneficial for analysis of human movement for diagnosis. Still, physicians, not software packages play the vital role in diagnosis. And it must be emphasized that most physicians still prefer analyzing human movement via visual inspection then analyzing via use of charts and reports. This shows that visual movement data gathered from patients still can be more valuable in its raw than its processed

form. Besides, visual human movement data is also being used in training and teaching. Medical Students still learn about diseases and disorders by watching videos and looking at still pictures. A physician must be well aware of the movement pattern that is caused by a disorder to make a successful diagnosis. Indeed, some of the software packages we mentioned can also provide illustrations by animating a skeletal structure retrieved from motion capture data, but just animating a skeletal structure seems unrealistic and unfamiliar to human eye.

For all of these reasons, virtual humans can be used to illustrate visual human movements in addition to a skeletal structure. Videos and pictures captured from patients and pre-rendered videos of human animations are useful for the same reasons. But they lack interactivity and also suffer from large storage space and transmission bandwidth they require. By using real-time animations, it is possible to inspect the movements of patient from different angles and distances. Hence, it would be wise to provide physicians with a compact and handy tool using a virtual human to illustrate movement data in real-time. That tool would use motion capture data with 3D human models. It will also be beneficial considering disk space because such files are generally small compared to video and image files. Besides, anonymous virtual characters are advantageous for protecting the privacy of the patient.

In the light of these facts, we propose a tool to visualize medical human motion data by using virtual human. This tool will demonstrate virtual humans in interactive rates. Physicians and medical students will be able to inspect movements of patient from any angle and distance they prefer. The storage to represent this kind of medical data will also be reduced significantly. To animate virtual humans, we examined many deformation methods. Most of these methods are developed and used for especially 3D human models. We have also briefly examined other well known deformation methods to see whether they can be of any help in our case. Most of the deformation methods used for virtual human animation is classified as skeleton (bone) animation. First, we briefly review the deformation methods other than skeleton animation. Then, we will talk about character and skeleton animation methods.

1.2 Deformation Methods

Areas such as computer drawing and computer aided design, make use of deformation methods to create and modify surfaces, solids, and curves. Some deformation methods are used to simulate fabric draping and folding (Gibson & Mirtich, 1997). Other deformation methods segment images and fit curved surfaces to noisy images (Gibson & Mirtich, 1997). Computer graphics and animation make great use of deformation methods, particularly for facial expressions, character animation, and cloth animation. Even, simulation of training and surgical systems requires physically based, non-linear, and complex deformation of tissues which is able to execute in interactive speeds (Gibson & Mirtich, 1997).


We can categorize deformation methods by the techniques they use, two main groups: These are Geometric (Non-physical) and Physical deformation methods. We will give brief information about some well known Geometric and Physical deformation methods. Geometric deformation methods

(GDM) are generally computationally efficient whereas physical methods aren't. GDMs do not depend on explicit physical rules; their success mostly depends on the skill of designer. Since, physical methods apply deformations based on rules and mathematical models; they can generally be modified via available parameters. Thus, physical approaches limit the flexibility of designer, though they simplify the task.

1.2.1 Geometric Deformation Methods

In this section, we look at some well known GDMs such as Free Form Deformation (FFD) and splines which are used for general purposes. Most of skeleton deformation methods for character animation are also GDMs. We will give the details about these methods in skeleton animation section.

In one of the earliest works, (Burtnyk & Wein, 1976) incorporated bilinear interpolation to deform 2D objects. Later on, a work related to FFD was done by Barr (1984). Barr studied the effect of mappings between 3D spaces and how hierarchical arrangement of these mappings affects surfaces. For instance, an object twists along x axis when the mapping below is applied where $f : R^3 \longrightarrow R^3$:

$$f(p) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos v_x & -\sin v_x & 0 \\ 0 & \sin v_x & \cos v_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix}$$


**Figure 1 A mapping for twist along y axis
(Courtesy of (Gjerde))**

Similar deformations can be achieved by applying other mappings or their combination them. Even though, Barr's method was a powerful design tool, it was not intuitive. It has limited deformation types and it was too hard to make regional deformations.

FFD credited to Sederberg (1986) was a generalized approach to Barr's method and it became really popular. Initially an object is placed in a lattice of grid points. This grid can be any standard geometry such as a rectangular prism or sphere. As the grid that encloses the object is distorted by modifying its grid nodes, object inside distorts in the same way. When a grid node is moved, interpolation is applied to find vertex locations of the object inside the distorted grid. FFD extends Burtnyk and Wein's method by using higher order interpolation such as cubic interpolation in 3D space. Since, it is easier and intuitive to manipulate grid points than manipulating the objects vertices, this methods proves to be very powerful and useful. In Figure 2, FFD is applied to a 3D object.

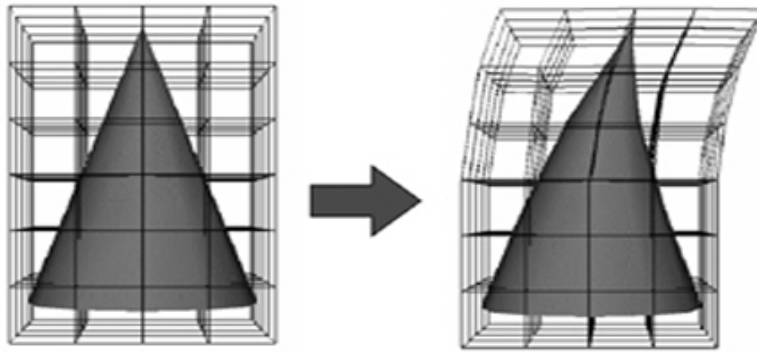


Figure 2 FFD applied to a 3D object
 (Courtesy of (Digital Human Research Center))

Another well known geometric deformation method is splines which are widely used in CAGD. Bezier curves are developed by Pierre Bézier in the 1970's because of the need for numerically specifying curves and surfaces. Designers needed intuitive ways while modifying deformable objects in Computer Aided Geometric Design (CAGD). After the introduction of Bezier curves, many other set of curves are developed such as double-quadratic curves, Bezier splines (B-spline), rational B-splines, NURBS, etc. In all of these representations, a curve or surface is defined by a set of points which are called control points (CP). A designer can easily modify a curve by moving its CPs, adding new ones, deleting existing ones or changing their weights.



Figure 3 A cubic spline curve which is defined by CP P1, P2, P3, and P4

Since, these methods are computationally efficient, they allow object editing in real time. The designer is also granted a great control over the shape with many CPs. On the other side, to make a precise modification can be quite troublesome due to the number of CPs. Parent (1977) and Allen, Wyvil and Witten (1989) proposed methods which moves neighbor CPs of a modified CP along itself based on a proximity criteria. Bartels and Beatty (1989) came up with a way to manipulate the shape of the curve by directly moving the points on a B-spline curve itself instead of editing CPs.

1.2.2 Physical Deformation Methods

In this section, we look at some well known physical methods such as mass-spring methods, continuum and finite element methods and approximate continuum models, low and degree freedom methods. "A Survey of Deformable Modeling in Computer Graphics" by Gibson and Mirtich (1997) is a valuable resource about physical deformation models. There are also other physical deformation methods for character animation such as multi-layered human body model. We will give the details about these methods in skeleton animation section.

Mass-spring method is one of the important physical deformation models. Object is modeled as a lattice structure whose nodes behave as point masses that are connected to each other by springs. Most of the time linear spring forces are used, but tissues such as human skin which exhibit inelastic behavior use non-linear springs.

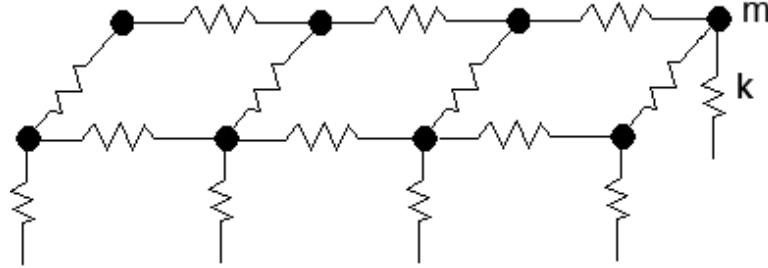


Figure 4 A mass-spring model. Each mass node put force on its neighbor nodes and all of the system

As seen in Figure 4, each mass node in the lattice behaves as a force source for other mass nodes. Their behavior of motion conforms to Newton's Second Law

$$m_i \ddot{x}_i = -\gamma_i \dot{x}_i + \sum_j g_{ij} + f_i$$

where m_i is the mass of node i , $x_i \in \mathfrak{R}^3$ is the position of mass node i , and terms on the right-hand side are forces affecting the mass node. The first term $-\gamma_i \dot{x}_i$ is the damping force depending on the velocity. $\sum_j g_{ij}$ is the sum of force applied by spring on mass i where each g_{ij} is parameterized by k_{ij} and m_j . f_i is the combination of external forces on the mass i such as gravity, collision force applied by another object or user applied force. Motion of entire system is basically decided by the motion of all mass nodes forming the lattice. Since m_i is known, it is possible to calculate acceleration of i th mass \ddot{x}_i at time t . In Mass-spring models, ordinary differential equations are integrated through time. By using a step by step procedure, nonlinear ordinary differential equations are converted into a sequence of linear algebraic systems.

Among the first implementations of mass-spring methods was about cloth animation which belong to Terzopoulos, Platt, Barr and Fleischer (1987). Another main application of mass-spring method is facial animation. Terzopoulos and Waters (1990) applied a dynamic spring model. They modeled a mesh grid composed of three distinct layers of facial tissue. These layers are the dermis, subcutaneous fatty tissue and the muscle layer. For each layer, different spring constants were used. This face model used 6500 springs, and still it can be animated in real-time. Provot (1995) also made use of spring models for cloth animation. Provot take into account the non-elastic behaviors of woven fabrics took.

Dynamics of mass-spring models are relatively simple compared to other physical deformation methods. They can be used at interactive times. They can be easily modeled to parallel computing. But, it has also some disadvantages. It is not an easy task to assign proper material properties such as spring constants to achieve a realistic mass-spring model. As, certain constraints are not naturally implemented in the model; one has to find a means to integrate them. For example, there is no direct way of volume preservation; it can be implemented by applying some nonintuitive external forces on the mass nodes. And some examples of mass-spring models exhibit stiffness problems due to large spring constants which are used to model nearly rigid objects. Stiffness problem can be observed as poor stability and requires the numerical integrator to take small time steps, since large spring constants causes the right hand side of the equation become small.

Continuum deformation method is another widely used method. A deformable object is based on equilibrium of the object's material properties and external forces. A deformation can be realized by a function of external forces and material properties of the object. An object is considered to be at equilibrium, when it reaches the minimum potential energy. The total potential energy of the deformable object is defined by the difference of total strain energy of the object and the work done by external loads on the object. Hence:

$$\Pi = \Lambda - W$$

where Π is the minimum potential energy. We can define strain energy as energy stored in the body by deformations. The work done can be due to three sources: forces applied to discrete points, forces applied over all the objects such as gravity and forces over only the surface of the object such as pressure.

By expressing both Λ and W in terms of object deformation, the shape of the object in equilibrium can be achieved. When Λ and W are represented by a function of material displacement over the object, the system minimum energy state which is the equilibrium can be found by determining the value where derivative of Π with respect to material displacement function is equal to zero. If a closed-form analytical solution can not be found, Finite Element Model (FEM) can be used to find a approximate solution of partial differential equations.

In FEM, object is divided into a set of elements and approximate solution of continuous equilibrium equation of each element is calculated. To achieve continuity between elements, these approximations must conform to constraints at the node points and the element boundaries. Further information about this method can be found in (Gibson & Mirtich, 1997).

FEM exhibits more realistic deformations relative to mass-spring models, but it requires high computational power which makes it difficult to apply in real-time systems. Traditional FEM is generally applied to metals or similar materials which has very limited deformations. It will fail with materials such as human skin which can deform more compared to metals (Gibson & Mirtich, 1997).

1.3 Character Animation Methods and Skeleton Animation

Even though most of the deformation methods can be applied to character animation, there are particular methods developed for this specific purpose. Character animation methods are various and application dependent. There is a trade off between realism, speed and compactness. For instance, priority for computer games is speed where models need to be rendered in real-time. Collision detection and artificial intelligence must be also handled at the same time. Realism is of secondary importance in the case of computer games. This also applies for war simulations. In most of simulations, realism is important, but this more to do with environment physics than the quality of animation. On the other hand, animators who prepare animations for films need to achieve visual realism. Since, movies do not require any interactivity, film animation are all prerendered. However, short render times will boost the speed of film production. There are also web-based animations. Since data will be distributed through internet, data compactness is an issue in these applications. Web-based animations take advantage of LOD algorithms and progressive transmission. We can give avatars as an instance for web-based applications of virtual human animation.

Quality of animation is not simply determined by the quality of deformation method. If an animator tries to make realistic human walking animation, virtual human must move in the same way as the real one. In films or movies, to animate a virtual human, motion data of a real human is captured in a motion capture laboratory (mocap lab), and then this mocap data is applied to virtual human animation. Quality of animation is determined by the positions of markers and cameras, the number of markers and the quality of software and system used. This goes same for computer games, but in computer games generally lower resolution mocap data is used where there are only a few numbers of markers relative to the ones used for film animations. Motion can also be prepared by an expert animator via use of animation tools such as Maya, 3D Studio Max, etc. We will give more detail about mocap data in Section 4.4 and 4.5.

We mentioned that there are various kinds of deformation methods for virtual character animation. Merry, Marais & Gain (2006); Collins & Hilton (2001) and Kavan, Collins, Zára & O'Sullivan (2007) make categorizations about these deformation methods. We will categorize these methods according to the techniques they use. We separate them as geometric, example-based and physical-based deformation methods. Most of these methods are driven by an underlying skeletal structure. We will explain each category. But, we will put more emphasis on geometric skeleton animation methods because the two deformation methods we implemented are both geometric deformation methods. We did not implement any example-based methods, but they are a great portion of character animation methods. Thus, we will also give detailed information about them.

1.3.1 Geometric Methods and Skeleton Subspace Deformation

Interactive applications generally make use of geometric skeleton deformation methods for their simplicity and efficiency. In this section we will give information about well known GDMs which are specifically developed for virtual character animation. Common points of all GDMs are their speed and simplicity. The most well known and oldest geometric skeleton animation method is Skeleton

Subspace Deformation (SSD). Due to its high speed and simplicity, it is the most preferred GDM method even today. But, SSD suffers from some well known defects such as collapsing elbow, twisting elbow and candy wrapper. Lately, new GDMs which are free of these defects are proposed. All new methods attack the problems of SSD and take it as a reference in their benchmark tests. Hence, it is wise to give basics about SSD before discussing other methods.

1.3.1.1 Skeleton Subspace Deformation

SSD is an unpublished method but, it is subsumed in other published articles (Magenat-Thalmann, Laperrri, re, & Thalmann, 1988). SSD is given multiple names such as Single-Weight Enveloping (SWE), Smooth Skinning, Matrix-palette Skinning (MPS) and Linear Blend Skinning (LBS). In SSD, deformation is driven by an articulated structure. In analogy to real world, the hierarchical articulated structure is called the skeleton and the surface model is called the skin. The skeleton is actually a virtual one and does not exist in the real animation. The skeleton consists of links, called bones, which are connected by joints. The bones are in general considered to be rigid and have fixed length. Every bone is represented in a local coordinate frame called the bone coordinate system.

A character is modeled in a rest pose. Rest pose of humanoid characters are generally T-shaped where whole body is straight and arms are opened in both sides perpendicular to body. Hence, the pose is also called T-pose. Some may call it dress pose or binding pose as well. The skeletal structure is placed inside the character model and bones are aligned along body limbs in its rest pose. The bone's root joint is the origin of the bone local coordinate system and bone itself lies along an axis of local coordinate system.

Each vertex v is then assigned a set of influencing joints together with a weight factor corresponding to each influencing joint. When a bone moves, the movement is applied to the assigned vertices. Deforming the character into a different pose involves transforming each bone along with its influenced vertices. A vertex can be influenced by multiple bones. The deformed vertex \bar{v} is a weighted sum of the movements determined from each bone through by the blending function:

$$\bar{v} = \sum_{k=0}^{n-1} w_k M_k M_{Rest,k}^{-1} v$$

where influence of each bone k on a vertex v is rated by a weight w_k whose value lies in the interval $[0, 1]$. Total sum of all weights per vertex must be normalized to 1.0. M_k is the local transformation of bone k for new skeleton pose. $M_{Rest,k}^{-1}$ transforms vertex v to the local coordinate of bone k in the rest pose. A new pose of a model is created by defining a new skeleton pose.

Most interactive applications still prefer SSD because it still outperforms other GDMs in simplicity and speed. Besides, SSD only exposes the defects in extreme cases. Hence, we have

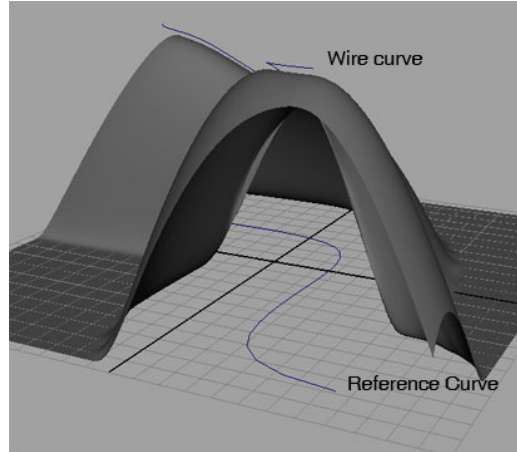
implemented SSD for our virtual human animation. More insight into the SSD algorithm will be given in Section 3.3.

1.3.1.2 Other Geometric Methods

There have been many alternatives or enhancements to SSD. These methods try to prevent most known defect of SSD while trying to achieve speed and simplicity. Some of them also contribute to deformations such as muscle effects. When a man flexes his arm muscles, a growth in the mass of fore arm can be observed. So contributing muscle bulge effect to the deformation increases the reality and quality of animation. An important portion of these methods GDMs which are capable of performing muscle bulge and similar effects are curve bone models. Curve bone methods emphasize the fact that SSD defects are side affects of linearity. So, non-linear nature of curve bones can prevent SSD defects (Forstmann & Ohya, 2006). Other important set of GDMs are modified versions of SSD. These methods especially aim to operate with input data similar to SSD to keep modifications minimal for existing applications using SSD as much as possible.

Singh and Fiume (1998) introduced means to deform models via curves. They called it Wires. Though Wires was a generic model deformation method, it became a basis for studies on curve based bone deformations. Wires method make use of two curves which are reference curve R and wire curve W and three scalar values s , r , f . s is the scale factor. r is the max influence distance. And f is the scalar valued influence function. Any curve C can be parameterized by variable $u \in [0,1]$ where $C(0)$ and $C(1)$ are the starting and ending point of C respectively and any point on curve is denoted by $C(u)$. At the beginning of the deformation, each point P which are to deform are permanently mapped to a point on R such that the set of points mapped on R is defined as $P_R = \left\{ R(p_R) \in \mathfrak{R}^3 \mid p_R = \min_{u \in [0,1]} (\|R(u) - P\|), p_R \in [0,1] \right\}$. In this definition p_R is a variable just like u which defines the one dimensional position on the reference curve R .

The corresponding point of P on curve R is $R(p_R)$ which is closest to P . Similarly, a point $W(p_R)$ on wire curve W is mapped to P meaning that defined algorithms to how to map model points to the points on the reference curve R . They keep the reference curve R constant where deformation is driven by the modification of wire curve W . f is a monotonically decreasing density function to determine the influence of the curve on each model point depending on the influence distance $\|R(p_r) - P\|$ and the max influence distance r .



**Figure 5 A plane which is deformed by wires method
(Courtesy of (McKinley))**

If a point P is out of influence distance, then it is not affected by the wire curve. Before deformation steps are stated, Figure 5 should be examined. Total deformation on the model point is done by scaling, rotation and translation in order. We must state that all of these three transformations are directly affected by the influence function f . First, point P is scaled along the direction vector $(P - R(p_r))$. Scaling is directly proportional to scaling factor s . From its scaled position point is rotated about the point $R(p_r)$ and around axis $W'(p_r) \times R'(p_r)$ where $W'(p_r)$ and $R'(p_r)$ are tangent of curve W and R at point $W(p_r)$ and $R(p_r)$ respectively. Finally, the translation vector $(W(p_r) - R(p_r)) * f$ is added to previous rotated point. Further analysis and the case of multiple wires are available in the article (Singh & Fiume, 1998).

Forstmann and Ohya (2006) proposed a work similar to Singh and Fiume (1998) to come up with an arc-spline based bone deformation method. For faster calculation, they exploit the spline's Frenet-frames. Frenet-frames Spline based bone methods deliver high quality deformation relative to SSD (Matrix Skinning). The real problem with SSD is that weighted summation of local transformation matrixes can result in a combined singular matrix. As we will mention in next section, it is possible to cope with the problems born out of singular matrices by incorporating extra bones. Since extra bones require extra calculation, sampling rate reduces. Spline-based bones act like infinitely many bones, so that extra-bone usage becomes unnecessary. In Figure 6, a bending and twisting comparison is given between SSD and spline-deform.



Figure 6 Bending and twisting
 (Courtesy of (Forstmann & Ohya))

In their study, splines were defined by three 3D points. For spline deformation, a complete coordinate system around the spline, the Frenet-frame, must be created. For each point on the curve, one can define a coordinate system which has three orthogonal basis vectors. The origin is the point on the curve itself. In Figure 7, coordinate systems (Frenet-frame) for some sample point on the spline are shown. Spline is defined by three points which can form a plane. In Figure 7, you can see a triangular shape which is formed by the three spline control points.

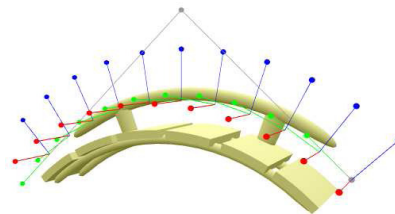


Figure 7 Spline Coordinate System - Frenet-frames
 (Courtesy of (Forstmann & Ohya))

Triangle is the plane we mentioned. First axis of Frenet-frame for each point on curve is that plane normal. Second axis is the derivative of spline function at that point. And the third axis is the cross product of the first and second axis, which points upwards in Figure 7. Each point of the character model is mapped to a point on the curve with a recursive space subdivision algorithm. Hence, each point of the model is related with a Frenet-frame of the curve. Subsequently, a combined (rotation, scaling, translation, etc.) transformation between the Frenet-frame of initial curve and the Frenet-frame of deformed curve is calculated. Weights for multiple splines are assigned by a painting tool which can be found in various 3D authoring tool. Finally, the combined transformation is applied to the model point related to the Frenet-frame. The transformation is applied in proportion with the spline weight. In their approach, a maximum of three spline-curves can influence a vertex of the

animated character. And this is sufficient for skinned animation. Their optimized implementation with OpenGL-shading language demonstrates very promising results in real-time.

Forstmann, Ohya, Krohn-Grimberghe and McDougall (2007) introduced enhanced and optimized version of Forstmann and Ohya's work. They incorporated reusable deformation-styles into B-spline based bone animation. Deformation-styles can be used to create complex material behaviors of metal, cloth or muscles. A pre-designed style can be applied to an arbitrary number of joints simultaneously. Once, the Frenet-frame of a model point is created, the model point is mapped to the coordinate system of the related Frenet-frame. Every deformation such as spline and style deformation is done in Frenet-frame. There are Style 1 and Style 2 deformations. Style 1 deformation is done by a radial scale function. The function deforms a point in the Frenet-frame due to the pose, intensity values of textures. Basically, an intensity value from the 2D texture is found for a model point. According to this intensity value and given pose, model point in the Frenet-frame is scaled. Style 1 can deal with cloth like effects. Style 2 uses values of 2D curves instead of intensity textures. Style 2 deformation is driven by two scaling functions. These are frontal and lateral scaling functions. Frontal and lateral scale functions scales points along blue and red axes which are shown Figure 7 respectively. Style 2 deformation deals with bulging effects. Style 1, Style 2 and spline deformations are applied in order. And then, points in the Frenet-frame are mapped back to model coordinate space. Forstmann et al. achieved a very efficient GPU based implementation of the spline-based skinned skeletal animation system, which outperforms the GPU based implementation described in (Forstmann & Ohya, 2006) by factor three for the basic spline skinning without deformation-styles.

Another work about curve bone skinning is due to (Xiaosong, Arun, & Jian, 2006). They used B-spline curve bones along with the rigid joint-based skeleton structure for the skin shape deformation. Deformation follows the tangent of the curve bone and curve points, so it results in higher sampling rates. Hence, collapsing skin and other undesirable skin deformation problems are avoided. The curve bone retains the advantages of SSD. It is easy to use and allows full control over the animation process. It also possible to achieve realistic muscle and fat bulge effects. Similar to earlier studies, three CPs are used to define curve bones. Curve bones are created according to the existing rigid skeleton structure. In the case where a joint is shared by only two bones, a CP is placed on each bone according to the angle between the bones and the joint itself is used as the third CP as shown in Figure 8.

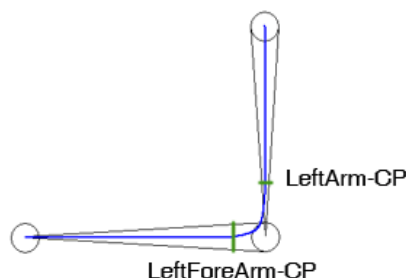


Figure 8 A bone curve derived from two standard bones

Cases such as a joint is shared by three or four bones are handled with conformable approaches. Deformation is similar to Forstmann and Ohya (2006). Each point on the curve has a local frame similar to Frenet-frame. Since B-splines with three CPs are used, local frames can be interpolated from three CPs. To prevent skin moving out from underneath skeleton, origins of these local frames must be projected onto the bone. Twisting rigid curve can not directly affect curve bone. Hence, two extra attributes, twist angle and twist distribution are defined to cope with this problem. Deformation occurs similar to earlier spline based method. Curves which bulge skin like a muscle are also defined and related to rigid bones. So, muscle bulge effects can be implicitly manipulated through rigid bones.

In their work Magnenat-Thalmann, Cordier, Seo and Papagianakis (2004) describe techniques by which dressed human characters are modeled and simulated in real-time. The method is also capable of operating with the SSD input. A matrix logarithm method which was proposed by Alexa (2002) was used to deform human models. It is shortly called as log-matrix. Alexa proposed a new operator such that $\alpha \otimes T = T^\alpha$ where α is a scalar value and T is a matrix. On that mathematical basis, he showed that the transformation matrices with no negative real eigenvalues can be defined by scalar multiples and a commutative addition of transformations. These operations allow the linear combination of transformations. Hence, one can create weighted combination of transformations, interpolate between transformations and construct or use arbitrary transformations in a structure similar to a basis of a vector space. By combining the matrix operator \otimes he proposed with basic matrix logarithm knowledge, Alexa came up with the equation which can be used for linear combination of an arbitrary number of transformations T_i with weights w_i :

$$\bigoplus_i w_i \otimes T_i = e^{\sum_i w_i \log(T_i)}$$

This technique has been originally demonstrated for key frame interpolation in animation. But, Thalmann et al. took advantage of it by replacing it with standard SSD blending equation. Alexa's equation requires the evaluation of the logarithm and exponential of matrices. Hence, its computation takes longer time than the classical SSD. Classical matrix blending leads to degenerated matrix as the relative rotation of two blender matrices approaches to π . Thus, volume loss and collapse effects occur on the joint adjacent parts of the limbs. Alexa's blending technique cope with these kinds of defects. But, it comes with its own defects because it picks a longer trajectory than necessary when interpolating rotations. A comparison of the log-matrix and SSD is given in Figure 9. Note the collapse at the hip.



**Figure 9 SSD (left) and log-matrix (right) comparison
(Courtesy of (Thalmann et al))**

Another method which can operate with SSD input is proposed by Kavan and Ladislav and Zara (2005). They called it Spherical blend skinning (SBS). In their work, Ladislav and Zara emphasized that skeleton deformations indeed have a spherical nature not linear. In SBS, transformations are considered to consist of a translation and rotation. To change the interpolation domain, SBS divides transformation into translation and rotation matrices and then convert rotation matrixes into quaternion representation. Quaternion representation is a four dimensional representation for rotations. We will give detailed information about quaternions in Section 3.3. The key point is to change the interpolation domain. The transition to non-linear interpolation domain is composed of two steps: determination of the center of rotation, and interpolation of multiple quaternions. The choice of the center of rotation has a considerable influence on the result of interpolation. Hence, singular value decomposition (SVD) algorithm is used to find the optimal rotation center. To make real-time execution possible, SVD is computed only for clusters of vertices instead of complete model. However, this introduces discontinuities in the deformed skin between individual clusters. In the case of two quaternions, second step is simple. But, for more than two quaternions, it gets quite difficult to interpolate. The linear interpolation of translation matrices is straightforward. But, quaternion interpolation must be handled delicately. An already established interpolation of two rotations is known as spherical linear interpolation (SLERP). SLERP is substantially slower than the simple linear interpolation used in SSD. To decrease time complexity, Ladislav and Zara proposed an approximate but faster linear interpolation of two quaternions and called it quaternions linear interpolation (QLERP). They come up with a blending equation to calculate deformed vertex v'

$$Q(v - r_c) + \sum_{i=1}^n w_i C_{ji}^{tr} r_c$$

where $Q()$ is QLERP function, r_c is rotation center, v is rest pose vertex and C_{ji}^{tr} is translation matrix. First term is quaternion interpolation and second term is just for translation matrix blending just as SSD. We already mentioned that SVD is computed only for clusters of vertices. Though the trick enables real-time performance, time complexity is still significantly high. Besides, using cluster centers may cause discontinuities of the deformed skin. But, SBS still proved to be useful to prevent well known SSD defects.

Dual quaternion skinning (DQS) is the latest and probably the most powerful method which operates with SSD inputs. Main goal of DQS is also to overcome SSD defects. Ladislav, Steven, Zara, and Carol (2007) extends SBS further more so that it makes use of dual quaternions (DQ) instead of quaternions. As we will explain in more detail later on, a DQ is an eight dimensional vector and is capable of representing both rotations and translations. Similar to SBS, bone transformations are assumed to be rigid (no scale or shear). DQs are also capable of representing translations. Hence, it is possible to define a rotation around a point with DQs and avoid rotation center problem of the SBS. For this method to work, all rotation and translation matrix representations are converted to DQs. Similar to SSD where transformations are multiplied with joint weights, DQs are also multiplied with

related joint weights and blended. The blending equation is called dual quaternion linear blending (DLB). DLB interpolate two rigid transformations along the shortest path, which means that a diversion is sometimes taken instead of a direct, straight rotation. Such diversions cause undesirable results in animation. DQS enables a valid rigid transformation where no collapse effects and volume loss occurs. In this study, we also implemented DQS beside SSD. So, we will give very detailed information about the DQS algorithm and its implementation in subsequent chapters.

1.3.2 Example-Based Methods and Shape Interpolation

Example-Based Methods (EBM) use many examples of shapes to achieve animation. Shape interpolation methods (SIM) are a large subset of EBMs. These examples can be the manual work of an artist, result of a realistic physical-based method such as FEM or a data from 3D scanning device.

SIM is a popular approach for especially facial expressions. It is also called shape blending or multi-target morphing. When applied to human body animation, SIM can be also driven by an underlying skeleton. It interpolates between many key shapes for deformation. These key shapes can be derived poses of one original shape, or they can be totally different shapes. For shape interpolation to work, vertices on the key shapes must be registered to each other. Registering shapes with each other is also a research area. Deformation occurs as these registered vertices are interpolated with a weighted summation. Key shapes are denoted by S_k where k is the index of key shapes. Interpolation is defined by the below blending:

$$\sum_{k=0} w_k S_k$$

Another version of this technique take a base shape denoted by S_o , and then use the difference of other key shapes with S_o when blending them. This function is shown below:

$$S_o + \sum_{k=0} w_k (S_k - S_o)$$

Basic shape interpolation methods suffer from some problems. Especially, deformation of limbs is quite inconsistent. When a limb is bent via a skeleton, the rigidity of the body can not be preserved. When shape interpolation is intended for facial animation, it has even some serious drawbacks. First of all, interpolation can not be smooth all the time. This occurs because motion path itself is only piecewise linear. This requires the animator to use key shapes between each four or five frames.

Compared to SSD and other geometric skeleton animation methods, most recent EBMs are capable of achieving more realistic results. But, their run-time efficiency is worse than that of SSD. Another important drawback is that for a successful animation, EBMs require a large database of high quality poses of shapes which is registered to a particular state of skeletal pose. We will investigate the EBMs proposed by Lewis, Corder and Fong (2000); Sloan, Rose and Cohen (2001) and Kry, James and Pai (2002).

Pose Space Deformation (PSD) is an EBM proposed by Lewis *et al.* (2000). PSD is a generalization and improvement over both shape interpolation and skeleton deformation methods. First of all, a base pose shape is determined by the rigid transformation of a body frame driven by the

underlying skeleton. PSD uniformly represents several types of deformations in a given pose space as control vertex displacements (denoted by $\bar{\delta}$) between a key pose shape and the base pose shape which is in the local coordinate frame of object. The difference of control vertices are calculated for every key shape and entered to pose space database. The key pose shape we mentioned is something like a muscle bulge of an arm prepared by the animator. Poses in the pose space can be defined by an underlying skeleton or it can be even defined by an abstract system of controls. This means that abstract deformation controls are basis for pose space and span it. The level (such as no bulge, middle bulge, or full bulge) of the muscle bulge can be an instance to the abstract control and so, it is a dimension and basis for pose space. For a given particular pose in the pose space, an interpolation method is used to find the interpolated difference vector by using the abstract control levels as input parameter. To give an example, a 0.5 level of bulge control will result with approximately a half muscle bulge displacement vectors according to interpolation function. Consequently, vectors in the pose space will be input parameters for the interpolation function. Displacement vectors calculated by interpolation function are added to the base pose shape for the final deformation.

The PSD algorithm necessitates efficient and well adjusted scattered data interpolation in high dimensional spaces. In their study, Lewis et al. used Radial Basis Function (RBF) to interpolate scattered data. A RBF generates a real value that depends on the distance of the given vector to the origin. Their choice of RBF is Gaussian which provide a “falloff” denoted by σ . It indicates the width of Gaussian in the PSD algorithm.

Real-time deformation is possible at decent resolutions because PSD synthesis cost is slightly more than the cost of classic shape interpolation. But, just like any other EBMs, this method is in need of a pose database. Some of the methods given below operate on some common basis, so understanding PSD will make it easier to understand them.

Shape by example is a method implemented by Sloan *et al.* (2001). Their method permits designer to define “adjectives” that describes example shapes. Each adjective becomes a basis for the space that they call “abstract space”. Subsequent to describing abstract space, each example shape is given an annotation. Goal of method is to generate shape $X(p)$ given vector p in the abstract space. The i_{th} key shape is represented by vector p_i in abstract space and its shape is $X(p_i)$. Given $p = p_i$, coefficient of i_{th} key shape in the interpolation function will be 1 where others are 0. So, $p = p_i$ will generate the i_{th} key shape $X(p_i)$. In their study, Sloan et al. used radial basis functions plus a hyperplane. Incorporating the hyperplane enables extrapolation which results in smoother interpolation. They claimed that they proposed a more efficient formulation which is fast enough for interactive (re)parameterization and runtime blending. It is also possible to map the implementation to graphics hardware.

Kry *et al.* (2002) proposed a method called Eigenskin. Similar to Lewis’s method, Eigenskin made use of differences of key shapes which they called pose corrections. They used Principal Component Analysis (PCA) to calculate minimum-error eigendisplacement basis. This

eigendisplacement basis was enough to represent this potentially large set of pose corrections. Since a vertex is not affected by every joint of skeleton, they preferred to apply PCA locally to avoid large number of important basis vectors. Interpolation was realized by the Gaussian RBF. Storing of a small number eigendisplacement basis leads to significant memory optimization and results in a better graphics hardware implementation.

Wang and Phillips (2002) took a different approach that is called Multi-weight Enveloping (MWE). MWE extends SSD so that it uses multiple weights. In MWE, each vertex has one weight for each coefficient of respective influencing joint's transformation matrix whereas in SSD, each vertex makes use of one weight per influencing joint. These weights are found by solving a linear least squares problem using a set of example shapes as input. Large number of weights can give rise to rank deficient (singularity) matrices in the least-squares solutions. Thought, it is possible, large number of weights will also complicate usage of graphics hardware.

Mohr and Gleicher (2003) developed another method which incorporates pseudo-bones to overcome some defects of SSD. This method first determines vertex influencing joints from the set of examples, and then uses a bilinear least-squares solver to fit an SSD model. A pseudo-bone is indeed related with a real bone. In the case where angle α between lower arm and upper arm decreases, biceps muscle bulges. To simulate the same effect, a perpendicular pseudo-bone is placed on the upper-arm. As angle α reduces, the pseudo-bone translates from upper-arm bone to bulge the skin. Method is also capable of handling collapse defects. After being trained with a set of examples, it is just another skeleton model with extra bones which operates as SSD. However the number of bone influences per vertex increases, which reduces rendering performance and may make hardware acceleration infeasible.

James and Twigg (2005) come out with an original method which is called Skinning mesh animations. They automatically identify statistically relevant bones by using nonparametric mean shift clustering of high-dimensional mesh rotation sequences. Subsequently, robust least squares methods are used to determine bone transformations, bone-vertex influence sets, and vertex weight values. As usual, efficient hardware rendering is supported. Besides, rest pose editing and deformable collision detection are also available.

1.3.3 Physical-Based Methods

Human body is a complex structure composed of rigid and non-rigid components, where especially non-rigid parts are hard to simulate. Geometric based-models are computationally efficient and simple but unfortunately results are not realistic most of the time. Physical-based models offer more realistic simulation because of its embrace physical-laws applied to the human model based either on anatomy or hand-crafted by artist (Aubel & Thalmann, 2001). Since the skinning algorithms try to achieve realistic motion and human body shape (representation and deformation), physical-based algorithms is one good choice as it provides more versatile simulation for rigid-body (skeleton) kinematics, and non-rigid soft-tissue deformations (muscle, fat, skin) (Porcher-Nedel). Physical-based models play important role especially when animated human bodies undergo contact and collision

with the surrounding environment. Physical-based models employ biomechanical or mechanical (elastic, fluid etc.) models which defines a continuous system in space and time that are used in various engineering disciplines (Aubel & Thalmann, 2000, 2001; Michael, Patrick, Joe, & Karan, 2005). Continuous problem is first discretized which leads large system of simultaneous ordinary differential equations. These discrete ordinary differential equations are solved in time-steps by numerically integrating using appropriate numerical solvers (explicit, semi-implicit, fully-implicit, and quasistatic). Two key aspects are here computational complexity and the stability of the system. You can increase time-step for gaining efficiency but system may become unstable due to large time-steps (Joseph, Eftychios, Geoffrey, & Ronald, 2005). System may compose of different material types with different characteristics (elasticity, viscosity, mass and damping factor), where for human body these materials refer to bones, muscle, fat, and skin etc. This variety (200 bones connected with many joints surrounded by non-rigid soft tissues) is hard to understand for modeling it and it also causes complexity which leads to more computationally complex models (Ferdi, Richard, Wayne, & Stephen, 1997). Since modeling human body in an exact way is not feasible, assumptions and approximations are needed. Assumptions such as mass less and purely force-based muscles o constant mechanical characteristics along the certain tissue can be made. Normally characteristic of tissue varies between individuals due to some factors (age, sex, etc) and it also varies by location within the individual. Example approximations are Hill-type muscle model, b-spline solids, modal analysis, and muscle strands which are used in biomechanics and also mass-spring(-damper) systems (MSD) and finite element methods (FEM) (Aubel & Thalmann, 2001).

Hill-type muscle model assumes the length of tendon remains constant as the muscle is stretched. Total muscle force $f_m = f_p + f_c$ where f_p is parallel and f_c is contractile element, Here, $f_p = \max(0, k_s(\exp(k_c e) - 1) + k_d \dot{e})$ where k_s and k_c are elastic coefficients, k_d is damping coefficient and $\dot{e} = (l - l_0)/l_0$ is strain of the muscle, l is the length and l_0 is the slack-length, $f_c = aF_l(l)F_v(l)$ where $0 \leq a \leq 1$ is the

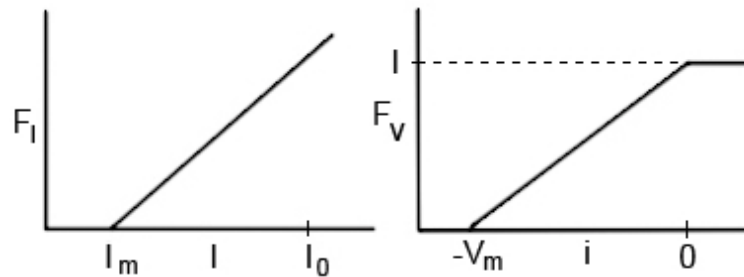


Figure 10 Linearized Hill-type model. (a) Force-length relation, (b) Force-velocity relation

activation level of the muscle, F_l is force-length and F_v is force-velocity. These expressions can be linearized for simplicity as seen in Figure 10.

In one of the first physical-based model, muscle is embedded in the Free Form Deformation (FFD) lattice and muscle deformation is achieved via deforming this lattice. Spring-mass model is the

key idea of this FFD model where lattice points are connected with springs. System is expanded with additional diagonal springs to keep geometric configuration stable. Later on Thalmann extends this model with angular springs, action line, and muscle shape concepts. In Figure 11.a, 1D spring-mass system is used for modeling human arm movement but different spring-mass model topologies like 2D are possible. Parameters of ellipsoid are obtained using spring-mass system and muscle is deformed accordingly (Aubel & Thalmann, 2000, 2001).

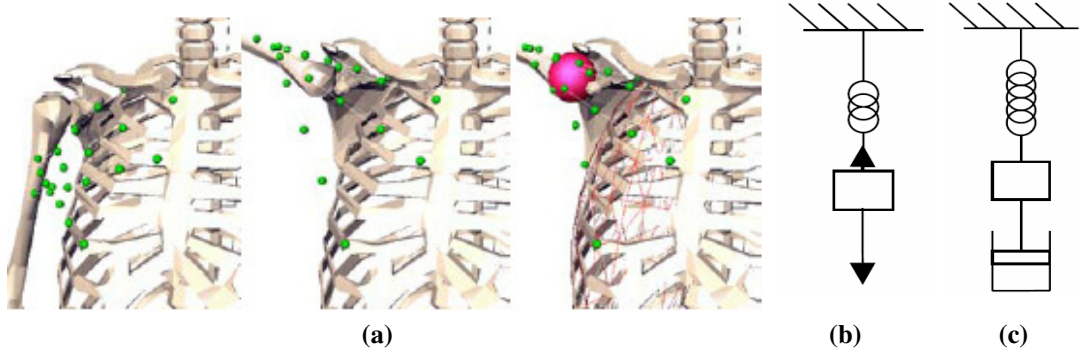


Figure 11 (a) 1D Mass-spring model applied on human skeleton, (b) Spring-mass, (c) Spring-mass-damper (Courtesy of (Aubel & Thalmann, 2000))

Basics of mass-spring model are explained below:

$$F = -k\Delta x \rightarrow m\ddot{x} = -k\Delta x \rightarrow m\ddot{x} = K \frac{(x_0 - x)}{|x_0|}$$

where Young's modulus $K = \frac{\sigma}{\epsilon}$, $\sigma = \frac{F}{A}$ is stress, and $\epsilon = \frac{\Delta l}{l_0}$ is strain

The above system is unrealistic since it does not take into account the resistance to motion due to friction in the spring or air resistance. Hence once it started its motion it will continue moving forever. Damping can be introduced to incorporate all resistances into account.

$$m\ddot{x} = K \frac{(x_0 - x)}{|x_0|} - \gamma\dot{x}$$

where γ is the damping factor.

Mass-spring(-damper) models requires little computation, simple to understand, and realistic for small deformations but need careful design of topology, not accurate physical model for tissue properties, and defining spring parameter may be difficult. The stiffer the spring, the smaller time step is used for numerical solver which can be problematic for bone structures. Collision is another aspect which must be taken into account using efficient algorithms.

Shape of tissue has potential energy and work done by motion transfers energy to tissue, hence new tissue shape when energy is transferred can be found as tissue deformation. FEM can be used for such purpose by defining the problem as (a) relating potential energy to displacement of tissue from rest position (strain energy), (b) relating work done as a function of tissue displacement and (c) computing the tissue shape when system is at equilibrium. It can be defined by shape elements (via

triangulation), shape function, or energy function. FEM allows modeling of complex soft tissue deformations and non-linear tissue properties more accurately, and employment of desired mechanical model. Major drawback of FEM is its high computational complexity therefore FEM is not suitable for real time applications (Joseph, et al., 2005). Hybrid methods are proposed for accurate and efficient systems

Recently anatomy-based models which based on anatomic concepts and motion observation gain popularity for human body simulations. Internal structures are modeled to capture important effects such as muscle bulges (i.e. biceps, Figure 12) that result from joint articulation. Generally these anatomic structures are animated using physics-based models which provide increased realism with the compromise of increased complexity (Ferdin, et al., 1997).

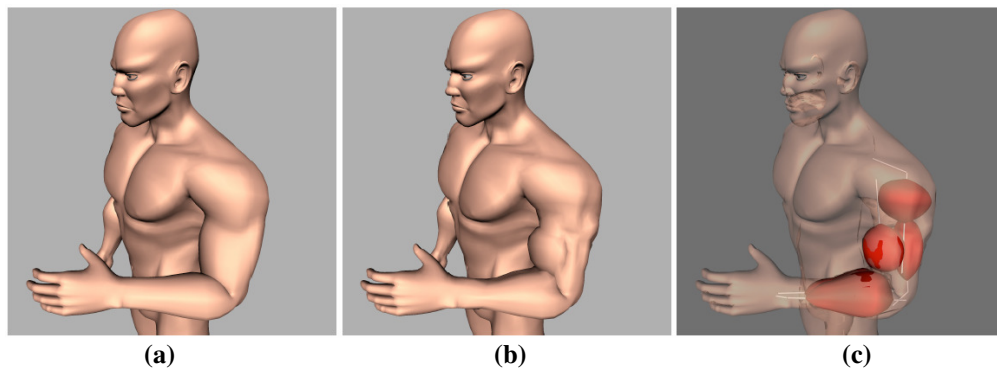


Figure 12 (a) Smooth skinning, (b) Anatomic deformation and (c) Underlying musculature (Courtesy of (Michael, et al., 2005))

Modeling of a human body using anatomical concepts employs kinematic models, physical-based models, and behavioral models. Human body contains over 200 bones that allow muscle attachment definition, which is approximately half of the body, and rigid skeleton that determines the general shape of human body. Among the anatomical systems that determine the outer shape, the musculature is the most complex one. Muscles are arranged side by side and in layers on top of bones and other muscles. It determines the shape of outer surface which is skin. Muscle can be divided into three major types; skeletal muscles, cardiac muscles, and smooth muscles. Muscle produces a force over a bone since it is attached to bones at least two joint points where it may be fixed or movable (insertion) type (Ferdin, et al., 1997; Nedel & Thalmann, 1998). This compound and complex system can create various movements as seen in Figure 13 such as;

- (a) rotation around one axis,
- (b) compound rotation around two or three axis,
- (c) translation in one to three dimensions,
- (d) rotations combined with translations,
- (e) axis sliding during rotation.

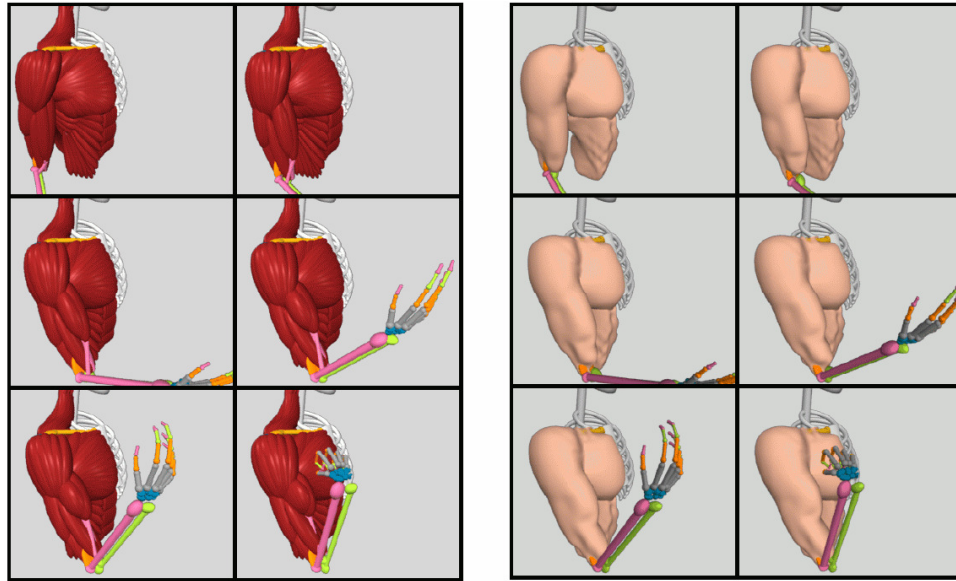
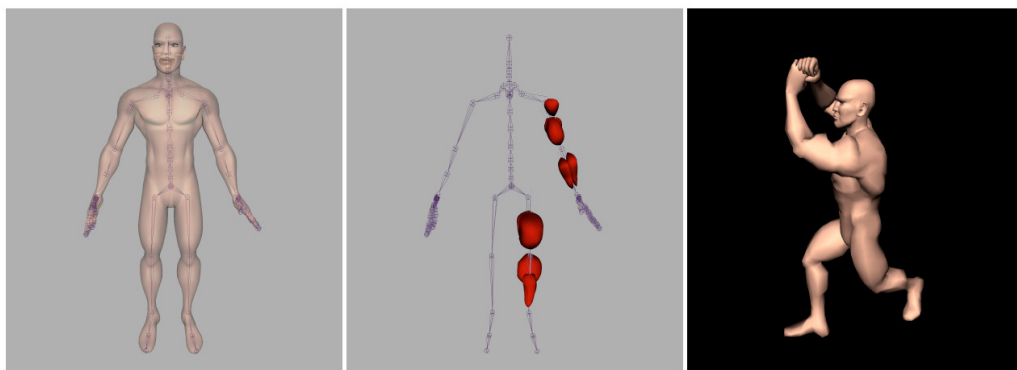


Figure 13 Behavior of the various muscle models with flexion at the elbow joint and resulting skinning (Courtesy of (Ferdin, et al., 1997))

These allowable motions within physical constraints lead to high level of mobility hence complex motion control system is needed. Isolation of body parts can be assumed for simplification but in reality human body works in synergy which means such simplifications may sacrifice realistic animation. Shoulder, spine, forearm, and head are typical examples which accuracy is sacrificed for simplicity. Using flexible surface near joint is poor approximation due to large set of possible deformations which occurs in its vicinity. (Ferdin, et al., 1997) Anatomy-based models generally use layered approaches where human body is divided into three major layers (Figure 14);

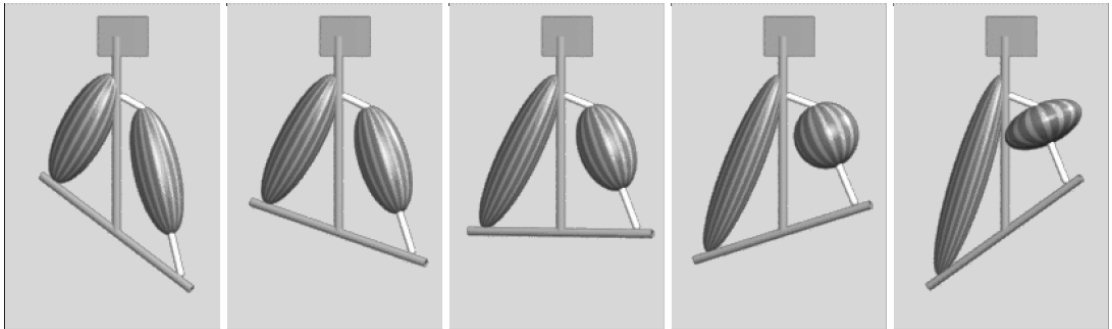
- (a) skeleton - rigid body,
- (b) intermediate layer – muscle, fat and various soft tissues,
- (c) skin – outer surface.



(a) (b) (c)
Figure 14 (a) Skeleton and surface, (b) Skeleton and muscles, (c) Skin
(Courtesy of (Michael, et al., 2005))

Stick figure models, surface models, and volume models are the other alternatives which can be used separately or within layered approach. Usually ellipsoids are (Figure 15) used for volume models

since they resemble musculature well and are easy to represent and manipulate mathematically. Metaballs or implicit functions are used in combination with applied musculature model to triangulate and render skin (Aubel & Thalmann, 2001).



**Figure 15 Ellipsoid musculature model with different poses
(Courtesy of (Ferdin, et al., 1997))**

Each one of these models can be used to model muscles, fatty tissue and skin. Between the skin and fascia or between deep organs fatty tissue slides relatively freely over the fascia whereas skin clings tightly to the fat. Fatty tissues are non-linearly viscoelastic, do not resist much to tension, and are considered incompressible. Skin is non-homogenous, anisotropic, non-linear viscoelastic, and nearly incompressible material. Age, obesity and exposure are some factors to determine its mechanical properties. Skin is the outer layer and unlike the fat layer it shows preventive characteristics for protecting internal structures by resisting to stress (Nedel & Thalmann, 1998).

CHAPTER 2

REPRESENTATIONS: QUATERNIONS AND DUAL- QUATERNIONS

2.1 Overview

We have already given brief information about transformation matrices such as rotation, translation and scaling matrix in the previous chapter. Especially, GDMs rely on rotation and translation data. Though working solely with translations is not so problematic, incorporating rotations can be quite a burden. Our main goal is to achieve a virtual human animation. So, one must find a convenient way to represent the configuration of the skeleton in a particular time frame. Wrong choice of representation can lead to many orientation and key frame interpolation problems in the animation process. In this chapter, we will give brief information about the well known orientation representations, their advantages and disadvantages. Between these representations, we will lay emphasis on quaternion which is a prerequisite for DQ. Consequently, we will give detailed information about DQ which is used in our implementation.

2.2 Orientation Representations

Orientation representations are at the heart of computer animation. It is quite an old subject. Preliminary representations suffered from some problems such as gimbal lock and erroneous interpolation of transformations. Later on, more convenient ways are proposed to represent the arbitrary orientation of an object in 3D space. More detailed information about these orientation representations can be found in Parent (2002).

Matrix representation is one primitive orientation representation. In 3D space an orientation can be represented by a 3x3 matrix. In Figure 16, two matrices and a halfway interpolation of them are shown. Matrix in Figure 16.a represents a 60° rotation around y axis and Figure 16.c represents a -60° rotation around y axis. Figure 16.b is the interpolated one.

$$\begin{matrix} \begin{bmatrix} 0.7660 & 0 & 0.6428 \\ 0 & 1 & 0 \\ -0.6428 & 0 & 0.7660 \end{bmatrix} & \begin{bmatrix} 0.7660 & 0 & -0.6428 \\ 0 & 1 & 0 \\ 0.6428 & 0 & 0.7660 \end{bmatrix} & \begin{bmatrix} 0.7660 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.7660 \end{bmatrix} \\ \text{(a)} & \text{(b)} & \text{(c)} \end{matrix}$$

Figure 16 Halfway interpolation of (a) and (c) results in (b)

Halfway interpolation of 60° and -60° is 0° rotation. Thus, matrix in Figure 16.b must be a zero matrix with no rotation effect. That shows us that matrix representation has serious problems while interpolating between animation key frames.

Another problematic representation is fixed angle representation. A 3D vector whose components are rotation angles about global axes (global coordinate) in fixed order is defined for orientation. Thus, (30, 45, -60) represents an object that rotates about x axis 30° , y axis 45° and z axis -60° respectively. This representation exposes the infamous gimbal lock problem. Consider the case where a 3D vector (0, 0, 1) is rotated by the fixed angle vector (90, 60, -30). 90° of rotation about transform the vertex to position (0, 1, 0). Since, result lies along the y axis, second rotation of 60° will have definitely no effect on vector position. Besides, when the vector (0, 0, 1) is exposed to the fixed angle rotation (90, 0, -90) or (0, -90, 0), new vector position will be (1, 0, 0) in both cases. It is clear that in such cases, degree of freedom reduces to two and a rotation can not be defined uniquely. This is the main reason which causes the gimbal lock. Gimbal lock is yet another reason for undesired interpolation. Also note that the order of axis selection affects the result of a rotation. So it is not possible to represent all rotations in the 3D space with just three parameters.

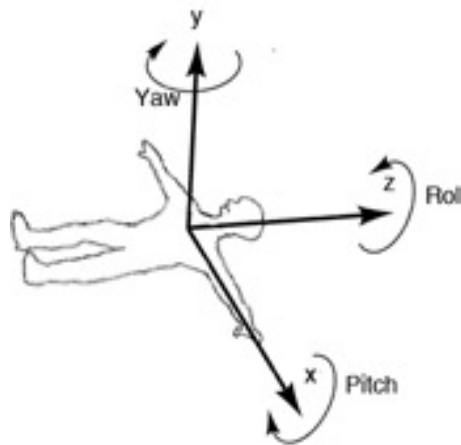


Figure 17 Euler's angle

Euler's angle representation is a very similar representation to fixed angle representation. Just like fixed angle, rotations are represented by three rotation angles. Euler's angle representation rotates the object in its own local coordinates. These local axes are known as Yaw, Roll and Pitch which are shown in Figure 17. In Parent (2002), it is proven that Euler's angle representation (p, y, r) is in reverse order in fixed angle representation (r, y, p) . Thus, they exactly depict same properties.

Axis-angle representation evolves from Euler's rotation theorem. Euler's theorem shows that any two orientations given for an object, one can be derived from the other by a pure rotation about a single axis. Angle-axis representation is composed of two parts: the rotation angle α and 3D vector a representing the axis which the object rotates about. Vector a can be shown as (a_x, a_y, a_z) . Hence, axis-angle is a four dimensional representation denoted by (α, a) or (α, a_x, a_y, a_z) . Figure 18 shows how a point is rotated around an axis by angle of α .

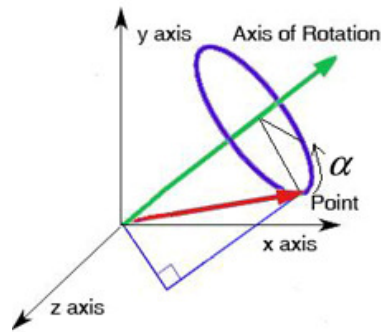


Figure 18 A point rotated around an axis by angle α

Axis-angle representation prevents gimbal lock. Besides, it is really a very intuitive way of representing an orientation.

2.3 Quaternions

Quaternions were first described by the Irish mathematician Sir William Rowan Hamilton in 1843. Indeed quaternion representation and axis-angle representation are very similar. Both are represented by four dimensional vectors. Quaternion also implicitly represents the rotation of a rigid object about an axis. Quaternion representation is not as intuitive as axis-angle representation. But, it provides better means of key frame interpolation and doesn't suffer from the problems that we mentioned just like axis-angle representation. The definition of a quaternion can be given as (s, m) or (s, x, y, z) where m is a 3D vector. In reality, quaternions are like imaginary numbers with the real scalar part s and imaginary vector m . Thus, it can be also written as $s + xi + yj + zk$. This is similar to 2D rotations being represented by standard imaginary number and quaternion operations are similar to operations of standard imaginary numbers. There are conversion methods between quaternions, axis-angle and rotation matrix. Common operations such as addition, inner product etc. can be defined over quaternions. Derivations and proofs about quaternions can be also found in (Eberly, 2006). Given the definitions of q_1 and q_2 :

$$q_1 = s_1 + x_1i + y_1j + z_1k \text{ or } q_1 = (s_1, m_1)$$

$$q_2 = s_2 + x_2i + y_2j + z_2k \text{ or } q_2 = (s_2, m_2)$$

Addition operation is defined as

$$q_1 + q_2 = (s_1 + s_2, m_1 + m_2) = (s_1 + s_2) + (x_1 + x_2)i + (y_1 + y_2)j + (z_1 + z_2)k,$$

dot (inner) product operation as

$$q_1 \bullet q_2 = \langle q_1 | q_2 \rangle = s_1 \cdot s_2 + m_1 \bullet m_2$$

and quaternion multiplication operation as

$$q_1 \cdot q_2 = (s_1 \cdot s_2 - m_1 \bullet m_2, s_1 \cdot m_2 + s_2 \cdot m_1 + m_1 \times m_2) \quad (\text{Equation 2. 1})$$

where \times indicates the cross, \cdot is scalar and \bullet is dot product. Quaternion multiplication is not commutative, but it is associative. And multiplication identity element is defined as $(1, (0,0,0))$. One can also perform the multiplication in the imaginary number domain by using the definition $q = s + xi + yj + zk$ and Table 1.

Table 1 Quaternion component multiplication

	1	I	j	k
1	1	I	j	k
I	i	-1	k	-j
J	j	-k	-1	i
K	k	J	-i	-1

Each quaternion has a conjugate q^* and an inverse (except zero quaternion) defined by

$$q^* = (s, -m) \quad \text{and} \quad (\text{Equation 2. 2})$$

$$q^{-1} = \left(\frac{1}{\|q\|} \right)^2 \cdot q^* \quad \text{where} \quad \|q\|^2 = s^2 + x^2 + y^2 + z^2 = q \cdot q^* = q^* \cdot q$$

Rotations are defined by unit quaternions. Unit quaternions satisfy $\|q\| = 1$. Since multiplication of two unit quaternions will be a unit quaternion, N rotations can be combined into one unit quaternion C_R as:

$$C_R = q_{R1} \cdot q_{R2} \cdot q_{R3} \cdots q_{RN}$$

It is also possible to rotate a vector directly by using quaternion multiplication. To do this, we must define 3D vector $v = (v_x, v_y, v_z)$ that we want to rotate in quaternion definition as $q_v = (0, v) = 0 + v_x i + v_y j + v_z k$. And, the rotated vector $v' = (v_x', v_y', v_z')$ can be defined as $q_{v'} = (0, v') = 0 + v_x' i + v_y' j + v_z' k$. Note that, in quaternion rotation equation q_R^{-1} can be replaced by q_R^* , since $q_R^{-1} = q_R^*$ is true for $\|q_R\| = 1$ (unit quaternion). So, rotation of q_v by quaternion q_R can be calculated as

$$q_V' = q_R \cdot q_v \cdot q_R^{-1} = q_R \cdot q_v \cdot q_R^* \quad (\text{Equation 2.3})$$

And, two rotations can be applied to the vertex v in quaternion math such as

$$q_V' = p_R \cdot (q_R \cdot q_v \cdot q_R^{-1}) \cdot p_R^{-1} = (p_R \cdot q_R) \cdot (q_v) \cdot (q_R^{-1} \cdot p_R^{-1})$$

$$c_R = p_R \cdot q_R \Rightarrow q_V' = c_R \cdot q_v \cdot c_R^{-1} \quad (\text{Equation 2.4})$$

where q_R and p_R are rotation quaternions and c_R is the combined rotation quaternion. The equation implies that vector v is first rotated by rotation represented by q_R and then p_R . Principal purpose of quaternions are interpolating and combining rotations successfully. In that sense, one can convert a combined and interpolated quaternion to a 4x4 affine rotation matrix to use with a 3D graphics API.

Now that we have some basic knowledge about quaternion mathematics, it is time that we look into conversion from axis-angle to quaternion. To convert an axis-angle (α, a_x, a_y, a_z) into quaternion (s, x, y, z) , we apply the formula:

$$q = (s, x, y, z) = \left(\cos\left(\frac{\alpha}{2}\right), \sin\left(\frac{\alpha}{2}\right) \cdot (a_x, a_y, a_z) \right)$$

$$= \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right) \cdot (a_x i + a_y j + a_z k) \quad (\text{Equation 2.5})$$

Thus, quaternion q defines a rotation about axis a (denoted by (a_x, a_y, a_z)) with an angle α . If axis a is a unit vector, then outcome quaternion will also be unit. From the definition we can infer that $q = -q$ since, rotating α degrees around axis a is equal to rotating $-\alpha$ degrees around axis $-a$. Proof can found in (Parent, 2002).

One of the most important issues is how to interpolate quaternions. An established interpolation function called SLERP is used for that purpose. It is a slow interpolation function. Function parameters are t , q_1 and q_2 which are respectively interpolation step, quaternion one and quaternion two. SLERP is a spherical interpolation, so it is much appropriate for interpolating rotations. Function is especially used for key frame interpolation. We will give more detail about SLERP in Section 4.2.

Another important issue is to convert the quaternions to the valid hand coordinate system. If rotations are given for the right-handed coordinate system, then they can be converted to left-handed coordinate system just by taking the conjugate.

2.4 Dual Quaternions

DQs are proposed by William Kingdon Clifford in 1873. Clifford called it biquaternion at that time. DQs are an extension of quaternions. It is important for us to understand them, since Dual quaternion skinning which we implemented operates with DQ mathematics. Additional information about them can be found in (Kavan, 2007) and ("Maths - Dual Quaternions,"). They represent both

rotations and translations whose composition is known as rigid transformation. They are represented by eight dimensional vectors:

$$(s, x, y, z, s_\varepsilon, x_\varepsilon, y_\varepsilon, z_\varepsilon) \quad (\text{Equation 2.6})$$

or by four dimensional vectors with dual elements such as:

$$(\hat{s}, \hat{m}) = (\hat{s}, \hat{x}, \hat{y}, \hat{z})$$

where (s, x, y, z) is non-dual part and $(s_\varepsilon, x_\varepsilon, y_\varepsilon, z_\varepsilon)$ is the dual part. DQs can be considered as quaternions whose elements are dual numbers such as \hat{s} that is denoted by $\hat{s} = s + \varepsilon s_\varepsilon$. If we gather dual and non-dual elements together, we can form two separate quaternions. So, a DQ is indeed composed of two quaternions. One quaternion is the non-dual part q , other is the dual part q_ε . q is the quaternion vector (s, x, y, z) and q_ε is the quaternion vector $(s_\varepsilon, x_\varepsilon, y_\varepsilon, z_\varepsilon)$. \hat{q} can be written as $\hat{q} = q + \varepsilon q_\varepsilon$ where ε is the dual unit satisfying condition $\varepsilon^2 = 0$. Thus, we can expand a DQ as

$$\hat{q} = q + \varepsilon q_\varepsilon = s + xi + yj + zk + \varepsilon(s_\varepsilon + x_\varepsilon i + y_\varepsilon j + z_\varepsilon k)$$

2.4.1 Properties of Dual Quaternions

If we define two dual DQs \hat{q}_1 and \hat{q}_2 :

$$\hat{q}_1 = q_1 + \varepsilon q_{\varepsilon 1}, \hat{q}_2 = q_2 + \varepsilon q_{\varepsilon 2}$$

then, DQ multiplication can be defined as

$$\hat{q}_1 \cdot \hat{q}_2 = (q_1 + \varepsilon q_{\varepsilon 1}) \cdot (q_2 + \varepsilon q_{\varepsilon 2}) = q_1 \cdot q_2 + \varepsilon(q_1 \cdot q_{\varepsilon 2} + q_2 \cdot q_{\varepsilon 1})$$

where the term $\varepsilon^2 \cdot q_{\varepsilon 1} \cdot q_{\varepsilon 2} = 0$ because of the definition $\varepsilon^2 = 0$. And, all other terms can be multiplied as any other quaternion just like (Equation 2. 1) in previous section.

The dual conjugate (analogous to complex conjugate) $\overline{\hat{q}}$ is denoted by s

$$\overline{\hat{q}} = q - \varepsilon q_\varepsilon \quad (\text{Equation 2.7})$$

This conjugate operator can lead to the definition of inverse of \hat{q}^{-1} which is

$$\hat{q}^{-1} = \frac{1}{\hat{q}} = \frac{\overline{\hat{q}}}{\hat{q} \cdot \hat{q}} = \frac{1}{q} - \varepsilon \frac{q_\varepsilon}{q^2}$$

when $q \neq 0$. A dual number without a non-dual part ($q = 0$) is called purely dual number and it has no inverse.

Let's analytically solve $\hat{q} \cdot \hat{q}^{-1}$ to verify that stated \hat{q}^{-1} is really the inverse of \hat{q} ,

$$\hat{q} \cdot \hat{q}^{-1} = (q + \varepsilon q_\varepsilon) \cdot \left(\frac{1}{q} - \varepsilon \frac{q_\varepsilon}{q^2} \right) = \frac{q}{q} - \varepsilon \frac{q_\varepsilon}{q^2} + \varepsilon \frac{q_\varepsilon}{q^2} = \frac{q}{q} = 1$$

A second conjugation operator is defined for DQs. It is the classical quaternion conjugation and is denoted by $\hat{q}^* = q^* + \varepsilon q_\varepsilon^*$ where conjugation of dual and non-dual quaternion parts is (2.2). Combining this two conjugation operator will lead to the formulization of DQ transformations on 3D points. Use of both conjugation on \hat{q} can be denoted as $\overline{\hat{q}}^*$. By the definitions (2.2), (2.6) and (2.7),

$$\overline{\hat{q}}^* = (s, -x, -y, -z, -s_\varepsilon, x_\varepsilon, y_\varepsilon, z_\varepsilon) \quad (\text{Equation 2.8})$$

By using the second conjugate operator, we can define the DQ norm as

$$\|\hat{q}\| = \sqrt{\hat{q}\hat{q}^*} = \sqrt{\hat{q}^*\hat{q}} = \|q\| + \varepsilon \frac{\langle q | q_\varepsilon \rangle}{\|q\|}$$

DQ norm conforms to the multiplicative property $\|\hat{q}_1\hat{q}_2\| = \|\hat{q}_1\| \cdot \|\hat{q}_2\|$. A DQ which satisfies the conditions $\|\hat{q}\| = 1$ and $\langle q | q_\varepsilon \rangle = 0$ is a unit DQ.

DQs naturally inherit some properties of regular quaternions. Just like regular quaternions, $\hat{q} = -\hat{q}$ represent the same transformation. Unless it is considered during DQ interpolation, this may lead to unsightly animation.

2.4.2 Rotation with Dual Quaternions

DQ transformation formula is defined as a series of DQ multiplication which similar to quaternion transformation formula (2.3). DQ transformation formula is

$$\hat{q}_V' = \hat{q}_R \cdot \hat{q}_V \cdot \overline{\hat{q}_R}^* = (\hat{q}_R \cdot \hat{q}_V) \cdot \overline{\hat{q}_R}^* = \hat{q}_R \cdot (\hat{q}_V \cdot \overline{\hat{q}_R}^*) \quad (\text{Equation 2.9})$$

Since, DQ multiplication is associative, computing $(\hat{q}_R \cdot \hat{q}_V)$ or $(\hat{q}_V \cdot \overline{\hat{q}_R}^*)$ as first is irrelevant.

Given that $q_\varepsilon = 0$, a DQ represents only a 3D rotation ($\hat{q} = q$). Such DQs can be used to define rotation of a 3D vector $v = (v_X, v_Y, v_Z)$. v can be defined as a DQ in the $\hat{q}_V = 1 + \varepsilon(v_X i + v_Y j + v_Z k)$ form. Then, rotation of v by \hat{q}_R can be denoted by the equation (2.9) where \hat{q}_V' represents the DQ from of transformed vector $v' = (v_X', v_Y', v_Z')$ and can be expressed as $\hat{q}_V' = 1 + \varepsilon(v_X' i + v_Y' j + v_Z' k)$. If $q_\varepsilon = 0$ then, by definitions (2.8) and (2.2) it is true that $\overline{\hat{q}}^* = q^*$ is satisfied and $\hat{q} \cdot \hat{q}_V \cdot \overline{\hat{q}}^* = q \cdot \hat{q}_V \cdot q^*$ can be expanded to

$$\begin{aligned} \hat{q}_V' &= q \cdot (1 + \varepsilon \cdot (v_X i + v_Y j + v_Z k)) \cdot q^* = q \cdot q^* + \varepsilon \cdot q \cdot (v_X i + v_Y j + v_Z k) \cdot q^* \\ 1 + \varepsilon(v_X' i + v_Y' j + v_Z' k) &= 1 + \varepsilon \cdot q \cdot (v_X i + v_Y j + v_Z k) \cdot q^* \\ (v_X' i + v_Y' j + v_Z' k) &= q \cdot (v_X i + v_Y j + v_Z k) \cdot q^* \end{aligned}$$

which leads to the quaternion multiplication equation (2.3) for rotation denoted by $q_V' = q_R \cdot q_V \cdot q_R^*$. In Figure 19, rotation of a triangular prism around an axis is shown. Numbers indicate the time frames of animation. “1” and “9” label the initial and final positions of the object respectively. Note that, dashed lines darken as the time frame index increases.

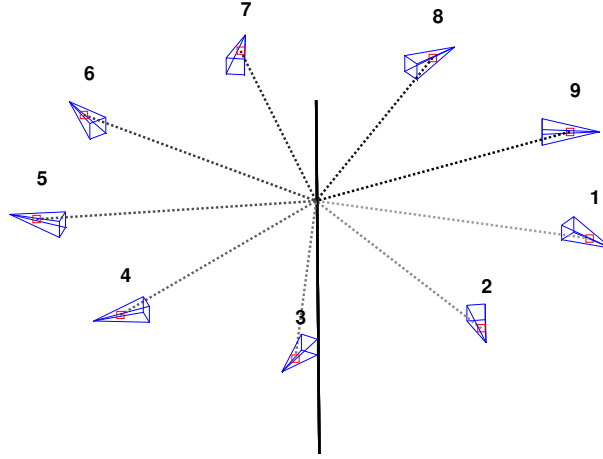


Figure 19 Rotation of a triangular prism around an axis in 3D space

Example 2.1: Since we have gone through the operations about DQs representing rotations, we can give a numerical example.

We have a 3D position vector $v = (1,1,0)$. What we want is to rotate v 180° around the axis $a = (10,10,10)$.

Step1: Convert all representations to DQ representation

Convert v to DQ: representation $\hat{q}_V = (1,0,0,0,0,1,1,0) = 1 + \varepsilon \cdot (1i + 1j + 0k)$

Convert axis-angle to DQ: We have a data in axis-angle representation $(180^\circ, 10, 10, 10)$ which means 180° rotation around $a = (10, 10, 10)$. So, we must form a rotation DQ \hat{q}_R whose non-dual part is a standard rotation unit quaternion q_R and dual part $q_\varepsilon = 0$. To do so we must convert the axis-angle representation data $(180^\circ, 10, 10, 10)$ to q_R . Since we want our DQ to be unit DQ, we better make use a_u which is unit vector of a . Unit form of a is $a_u = (0.5774, 0.5774, 0.5774)$. Axis-angle can be converted to a quaternion by the formula (2.5)

$$q_R = \cos\left(\frac{180}{2}\right) + \sin\left(\frac{180}{2}\right) \cdot (0.5774i + 0.5774j + 0.5774k)$$

$$\hat{q}_R = q_R = 0.5774i + 0.5774j + 0.5774k = (0, 0.5774, 0.5774, 0.5774)$$

from definitions (2.8) and (2.2),

$$\overline{\hat{q}_R}^* = q_R^* = -0.5774i - 0.5774j - 0.5774k = (0, -0.5774, -0.5774, -0.5774)$$

Step2: Using DQ transformation formula (2.9), calculate the solution

Now, we have everything we need to calculate the q_V' which is the rotated DQ form of v' from the formula (2.9). Quaternion multiplications can be done by using operator definition (2.1) or Table 1.

$$\begin{aligned}\hat{q}_V' &= \hat{q}_R \cdot \hat{q}_V \cdot \bar{\hat{q}}_R^* = (\hat{q}_R \cdot \hat{q}_V) \cdot \bar{\hat{q}}_R^* \\ \hat{q}_R \cdot \hat{q}_V &= (0.5774i + 0.5774j + 0.5774k) \cdot (1 + \varepsilon(1i + 1j + 0k)) \\ \hat{q}_R \cdot \hat{q}_V &= 0.5774i + 0.5774j + 0.5774k + \varepsilon(-1.1547 + -0.5774i + 0.5774j) \\ (\hat{q}_R \cdot \hat{q}_V) \cdot \bar{\hat{q}}_R^* &= (0.5774i + 0.5774j + 0.5774k + \varepsilon(-1.1547 + -0.5774i + 0.5774j)) \\ &\cdot (-0.5774i - 0.5774j - 0.5774k) \\ \hat{q}_V' &= 1 + \varepsilon(0.3333i + 0.3333j + 1.3333k)\end{aligned}$$

Since result DQ is in form $\hat{q}_V' = 1 + \varepsilon \cdot (v_X' i + v_Y' j + v_Z' k)$, we can extract the result position vector $v' = (v_X', v_Y', v_Z')$ as

$$v' = (0.3333, 0.3333, 1.3333)$$

Note: Since a_u is a unit vector, q_R is a unit quaternion satisfying condition

$$\|q_R\| = \sqrt{(\cos(\alpha))^2 + \sin(\alpha)^2 \cdot \|a_2\|^2} = 1$$

2.4.3 Translation with Dual Quaternions

DQs also represent translation. A DQ defined as $\hat{q}_T = 1 + \frac{\varepsilon}{2}(t_X i + t_Y j + t_Z k)$ corresponds to the translation vector $t = (t_X, t_Y, t_Z)$. As defined before \hat{q}_v is the DQ representation of position vector v and \hat{q}_v' is the DQ representation of translated vector $v' = (v_X + t_X, v_Y + t_Y, v_Z + t_Z)$. So, the translation t on the vector v can be computed by $\hat{q}_v' = \hat{q}_T \cdot \hat{q}_v \cdot \bar{\hat{q}}_T^*$. Fortunately, by the definition (2.8) we have $\bar{\hat{q}}_T^* = \hat{q}_T$. So that, we can expand the translation equation to

$$\begin{aligned}\hat{q}_v' &= \hat{q}_T \cdot \hat{q}_v \cdot \hat{q}_T = \\ &\left(1 + \frac{\varepsilon}{2}(t_X i + t_Y j + t_Z k)\right) \cdot (1 + \varepsilon(v_X i + v_Y j + v_Z k)) \cdot \left(1 + \frac{\varepsilon}{2}(t_X i + t_Y j + t_Z k)\right) \\ \Rightarrow \hat{q}_v' &= \hat{q}_T \cdot \hat{q}_v \cdot \hat{q}_T = 1 + \varepsilon((v_X + t_X)i + (v_Y + t_Y)j + (v_Z + t_Z)k)\end{aligned}$$

Figure 20-a shows a simple dual quaternion translation. Nine triangular prisms are connected with dashed lines to an axis in Figure 20-a. A translation is not related to any axis, but that axis will just be used as a reference to make a comparison between Figure 20-a and Figure 20-b in Section 2.4.3

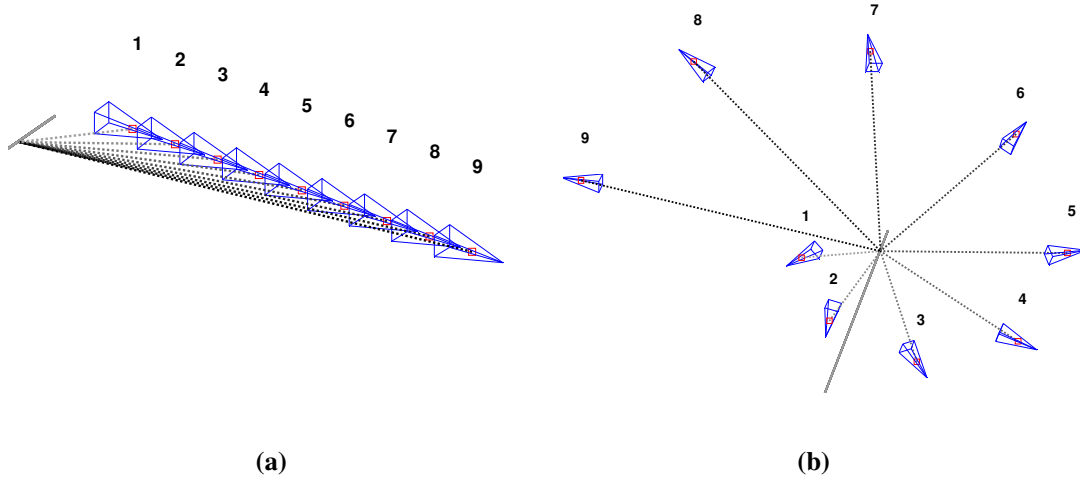


Figure 20 In (a) there is only translation and in (b) rotation around the axis is added to same translation

Example 2.2: We will make a translation example by using DQs similar to Example 2.1. Steps are same.

We have a 3D position vector $v = (-8, 12, -5)$. What we want is to translate v by $t = (5, 9, 9)$ using translation DQ.

Step1: Convert all representations to DQ representation

Convert v to $\hat{q}_V = (1, 0, 0, 0, 0, -8, 12, -5) = 1 + \varepsilon \cdot (-8i + 12j - 5k)$

Convert translation t to $\hat{q}_T = 1 + \frac{\varepsilon}{2}(5i + 9j + 9k) = 1 + \varepsilon(2.5i + 4.5j + 4.5k)$.

by definitions (2.8), $\bar{\hat{q}}_T^* = 1 + \frac{\varepsilon}{2}(5i + 9j + 9k) = 1 + \varepsilon(2.5i + 4.5j + 4.5k)$

Step2: Using DQ transformation formula (2.9), calculate the solution

$$\hat{q}_V' = \hat{q}_T \cdot \hat{q}_V \cdot \bar{\hat{q}}_T^* = (\hat{q}_T \cdot \hat{q}_V) \cdot \bar{\hat{q}}_T^*$$

$$\hat{q}_T \cdot \hat{q}_V = (1 + \varepsilon(2.5i + 4.5j + 4.5k)) \cdot (1 + \varepsilon(-8i + 12j - 5k))$$

$$\hat{q}_T \cdot \hat{q}_V = 1 + \varepsilon(-5.5i + 16.5j - 0.5k)$$

$$(\hat{q}_T \cdot \hat{q}_V) \cdot \bar{\hat{q}}_T^* = (1 + \varepsilon(-5.5i + 16.5j - 0.5k)) \cdot (1 + \varepsilon(2.5i + 4.5j + 4.5k))$$

$$\hat{q}_V' = 1 + \varepsilon(-3i + 21j + 4k)$$

so the result is $v' = (-3i, 21j, 4k)$

Note: If $\hat{q}_V = 1 + \varepsilon(-8i + 12j - 5k)$ was multiplied by $\hat{q}_T = 1 + \varepsilon(5i + 9j + 9k)$, the result would be the same. But, using half of the a translation as DQ such as $\hat{q}_T = 1 + \frac{\varepsilon}{2}(t_x i + t_y j + t_z k)$ on formula (2.9) makes it possible to combine translations and rotations

2.4.4 Dual Quaternions Representing Both Rotation and Translation

Transformation represented by DQs can be combined into one DQ similar to the quaternion combination (2.4). Assuming \hat{p} and \hat{q} are two transformation DQs and q_v is the a position DQ, their combined transformation c can applied to q_v just as

$$\begin{aligned}\hat{q}_v' &= \hat{p} \cdot (\hat{q} \cdot q_v \cdot \overline{\hat{q}}^*) \cdot \overline{\hat{p}}^* = (\hat{p} \cdot \hat{q}) \cdot (q_v) \cdot (\overline{\hat{q}}^* \cdot \overline{\hat{p}}^*) \\ \hat{c} &= \hat{p} \cdot \hat{q} \Rightarrow q_v' = \hat{c} \cdot q_v \cdot \overline{\hat{c}}^*\end{aligned}\quad (\text{Equation 2.10})$$

It is vital to understand that which transformation will be applied to q_v first. From the associative rule and parenthesis in (2.10), we infer that q_v will be first multiplied by \hat{q} from left and $\overline{\hat{q}}^*$ from right to generate a new position. This new position will then be multiplied by \hat{p} from left and $\overline{\hat{p}}^*$ from right to generate final position \hat{q}_v' . This proves that most inner transformation of the equation is applied first with an inside to outside manner. So, the first transformation is \hat{q} and the second one is \hat{p} .

Up to now, we have separately defined how translation and rotation operations can be done. To complete the work, we have to define an equation that will combine both translation and rotation. We already know that multiplying unit DQs will always result in another unit DQ. So, we will multiply a unit rotation DQ and a unit translation DQ. A unit rotation DQ is $\hat{q}_R = q_R$ and a unit translation DQ is

$$\hat{q}_T = 1 + \frac{\mathcal{E}}{2}(t_X i + t_Y j + t_Z k). \quad (\text{Equation 2.11})$$

and their composition is

$$\hat{q}_T \cdot \hat{q}_R = \left(1 + \frac{\mathcal{E}}{2}(t_X i + t_Y j + t_Z k) \right) \cdot q_R = q_R + \frac{\mathcal{E}}{2}(t_X i + t_Y j + t_Z k) \cdot q_R \quad (2.12)$$

It can be proved from the equation (2.12) that any unit DQ $\hat{q} = q + \mathcal{E}q_\mathcal{E}$ can be separated to its translation and rotation components.

$$\hat{q} = q + \mathcal{E}q_\mathcal{E} = q_R + \frac{\mathcal{E}}{2}(t_X i + t_Y j + t_Z k) \cdot q_R$$

Rotation is represented by the non-dual part $q = q_R$. And, with a little quaternion math it can proved that \hat{q}_T (2.11) can be represented as $2 \cdot q_\mathcal{E} \cdot q^*$ by the derivation

$$\begin{aligned}\mathcal{E}q_\mathcal{E} &= \frac{\mathcal{E}}{2}(t_X i + t_Y j + t_Z k) \cdot q_R \Rightarrow 2 \cdot q_\mathcal{E} \cdot q_R^* = (t_X i + t_Y j + t_Z k) \cdot q_R \cdot q_R^* \\ &\Rightarrow 2 \cdot q_\mathcal{E} \cdot q^* = (t_X i + t_Y j + t_Z k)\end{aligned}$$

where $q_R^* = q_R^{-1}$ for quaternions.

In Figure 20-a (in Section 2.4.3), we have shown a translation. The triangular prism was translating away from the reference axis. In Figure 20-b, a rotation around that reference axis is also combined into the same transformation. Note that while prisms are rotating around the reference axis, they also move away from it.

Example 2.3: We will make a combined transformation example with both translation and a rotation around an axis.

We have a 3D position vector $v = (3,2,5)$. What we want is to translate v by $t = (1,2,-5)$ then rotate it 180° around axis $a = (10,10,10)$ by using combined transformation DQ.

Step1: Convert all representations to DQ representation

Convert axis-angle $(180^\circ,10,10,10)$ to rotation DQ

$$q_R = \cos\left(\frac{180}{2}\right) + \sin\left(\frac{180}{2}\right) \cdot (0.5774i + 0.5774j + 0.5774k)$$

$$\hat{q}_R = q_R = 0.5774i + 0.5774j + 0.5774k = (0,0.5774, 0.5774, 0.5774)$$

Convert v to $\hat{q}_V = (1,0,0,0,0,3,2,5) = 1 + \varepsilon \cdot (3i + 2j + 5k)$

Convert translation t to $\hat{q}_T = 1 + \frac{\varepsilon}{2}(1i + 2j + -5k) = 1 + \varepsilon(0.5i + 1j - 2.5j)$.

Step2: Using DQ multiplication combine rotation and translation DQs into \hat{c}

$$\hat{c} = \hat{q}_R \cdot \hat{q}_T = (0.5774i + 0.5774j + 0.5774k) \cdot (1 + \varepsilon(0.5i + 1j - 2.5j))$$

$$\hat{c} = 0.5774i + 0.5774j + 0.5774k + \varepsilon(0.5774 - 2.0207i + 1.7321j + 0.2887k)$$

$$\hat{c}^* = -0.5774i - 0.5774j - 0.5774k + \varepsilon(-0.5774 - 2.0207i + 1.7321j + 0.2887k)$$

Step3: Using DQ transformation formula (2.9), calculate the solution

$$\hat{q}_V' = \hat{c} \cdot \hat{q}_V \cdot \hat{c}^* = (\hat{c} \cdot \hat{q}_V) \cdot \hat{c}^*$$

$$\hat{c} \cdot \hat{q}_V = (0.5774i + 0.5774j + 0.5774k + \varepsilon(0.5774 - 2.0207i + 1.7321j + 0.2887k)) \cdot (1 + \varepsilon \cdot (3i + 2j + 5k))$$

$$\hat{c} \cdot \hat{q}_V = 0.5774i + 0.5774j + 0.5774k + \varepsilon(-5.1962 - 0.2887i + 0.5774j - 0.2887k)$$

$$(\hat{c} \cdot \hat{q}_V) \cdot \hat{c}^* = (0.5774i + 0.5774j + 0.5774k + \varepsilon(-5.1962 - 0.2887i + 0.5774j - 0.2887k)) \cdot (-0.5774i - 0.5774j - 0.5774k + \varepsilon(-0.5774 - 2.0207i + 1.7321j + 0.2887k))$$

$$\hat{q}_V' = 1 + \varepsilon(1.3333i + 1.3333j + 5.3333k)$$

The result is $v' = (1.3333, 1.3333, 5.3333)$

2.4.5 Behaviors of Dual Quaternion Transformations

In this section, we will discuss the behaviors and affects of dual quaternion transformations by giving some figures. In Section 2.4.3, we already show translation in Figure 21-a and in Figure 21-b we add the rotation affect to it. In Figure 21-b we can not observe any screw affect since the prisms only translate away from the axis. In Figure 21-b, the screw movement occurs just by adding a translation along the reference axis to the rotation shown in Figure 21-a.

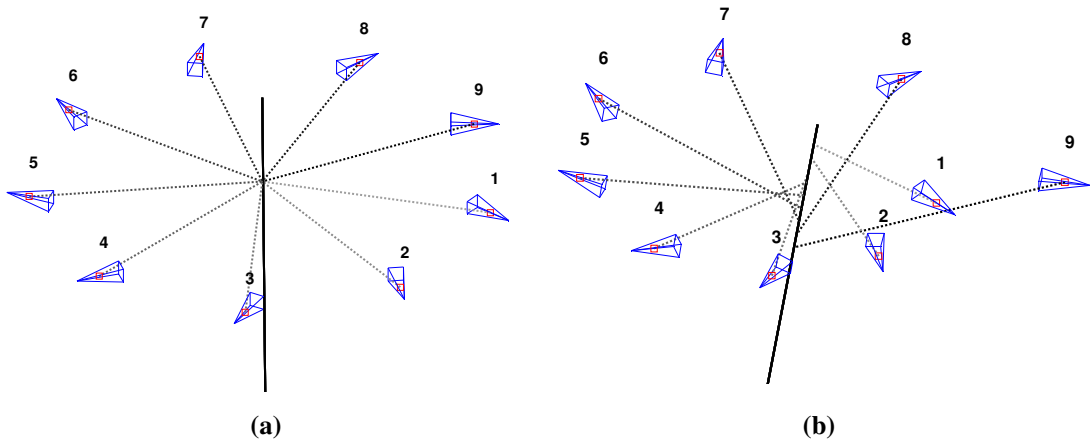


Figure 21 (a) shows rotation only and (b) shows both translation and rotation

In Figure 22, another screw movement is shown from different view points. DQs in form of (2.12) can make screw movement. Screw movement can not achieved by any other representation we mention in this chapter.

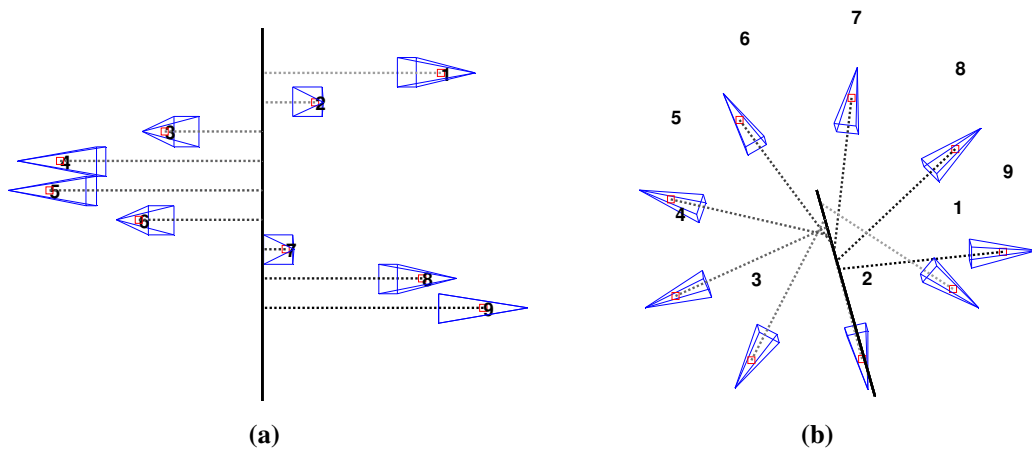


Figure 22 (a) and (b) shows the same screw movement from different angles

A three way of transformation is shown in Figure 23 from two different views. Prisms are translating, rotating and being affected by axis change. Note that in Figure 23-a, prisms move in manner that they form a shape similar to an ellipse. This is due to the axis change. Prism is following the reference axis since it is rotating around the axis.

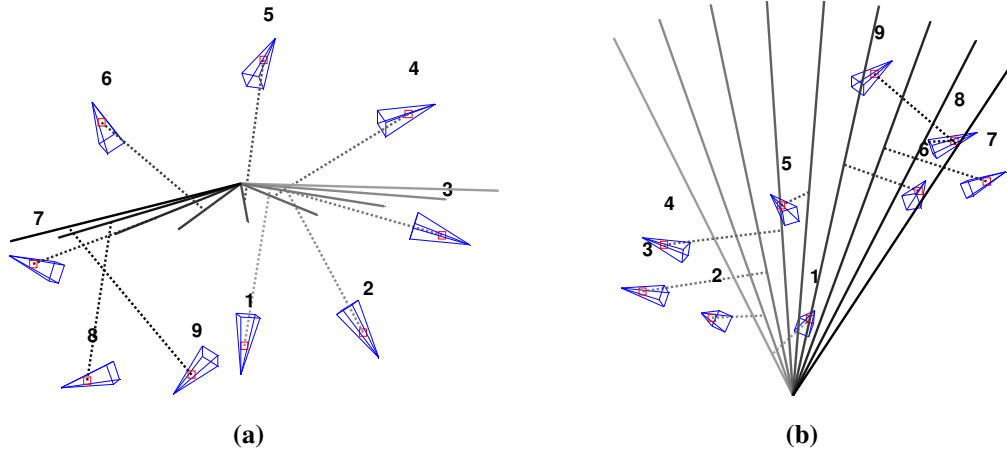


Figure 23 (a) and (b) shows the effects of both translation, rotation and axis change

Apparently, DQ is a quite capable representation. In Section 4.4, we will explain how it can be used in virtual 3D character animation and discuss why it doesn't suffer from some defects of SSD such as elbow collapse.

CHAPTER 3

SKELETON ANIMATION WITH SKELETON SUBSPACE DEFORMATION AND DUAL-QUATERNIONS SKINNING

3.1 Overview

In this chapter, we will give the implementation details about both SSD and DQS. Our tool makes use of these two GMs. To create an animation of a virtual character, one must compose and display separate deformed states of a human character model into a logical sequence. But, we will only focus on how we can create one state of deformed character model at a particular time t in the animation sequence. In the end, this is what a GM does to deform a mesh model according to the particular skeletal configuration at time t . We will first give information about the skeletal structure, the mesh model structure and the data that binds them. Then, we will explain how SSD and DQS operate on such data by explaining each step of algorithms and give pseudo codes if necessary

3.2 Articulated Skeleton Structure

All geometric deformation methods use control points or similar approaches to deform a model (3D mesh model). This approach has many advantages over the ease of use during shape deformation. First of all, it will be much easier and intuitive for a designer to deform the whole model through very few control inputs compared to the all vertices of model. Skeleton is an articulated structure to drive the deformation of a model to serve the common purpose. It abstracts the details of the model deformation from the animation controller whether it is a human designer or automated computer procedure. Another great advantage of using skeleton is to reduce the data representation to define the model deformation.

A skeleton is a structure composed of bones which have mass, volume and many other physical attributes. These bones are attached to each other by joints which are composed of ligaments, cartilage tissue, etc. Anatomical and physical methods make use these to some extent. But, all of these issues are irrelevant when it comes to GMs. A skeleton is only a virtual articulated structure which is composed of virtual bones and joints, and defines a direct deformation on the related mesh model

without conforming physical rules. In most of the applications such as games, they are not even supposed to be rendered in the scene. In the cases where bones are presented, they are generally represented by basic geometric shapes (triangular prisms, cylinders, etc) that align between two joint positions. Bones are expected, but not obliged, to be fixed length. There are parent-child relations defined for both bones and joints. The most top level joint of the joint hierarchy is known to be root joint.

In GM case, a skeleton is basically defined by the local transformation of each bone. The position of a joint is directly determined by its own local transformation and local transformations of its parents and ancestors. So, the local transformation of root joint will affect the whole skeleton. As an example, we will show how a skeletal pose in 2D space can be defined by only translations and rotations. First of all, we define a skeleton with its joint and bone names and its hierarchy in Figure 24.

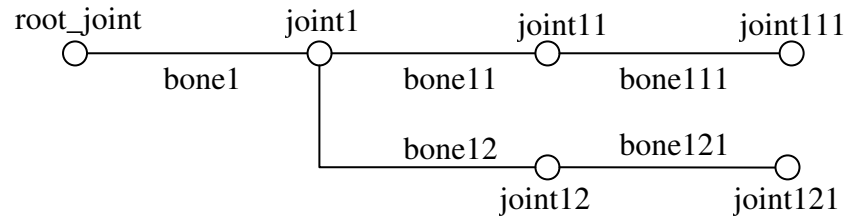


Figure 24 A sample bone hierarchy

The bone structure will be formed by a recursion defined by the pseudo code:

```

Function CalculateBoneVertexPosition(currentBone, combinedTransformation )
begin
    positionZero = [0 0 0 1];
    combinedTransformation =
        combinedTransformation * currentBone:localTransformation;
    currentBonePosition = combinedTransformation * zeroVec;
    for each childBone of currentBone
        CalculateBoneVertexPosition(childBone, combinedTransformation);
end

```

The recursion begins with the root bone (bone1) which is the top level bone of skeletal structure. So, the initial input of *currentBone* is bone1. Identity transformation must be used for the initial *combinedTransformation* input parameter. In each step of recursion, *currentBone:localTransformation* is combined with the input parameter *combinedTransformation*. The combined transformation will define the bone in the current recursion step. The joint positions at the end of the bones are calculated by transforming zero vertex with *combinedTransformation*. This concept is to be comprehended well because it is in the heart of SSD and DQS.

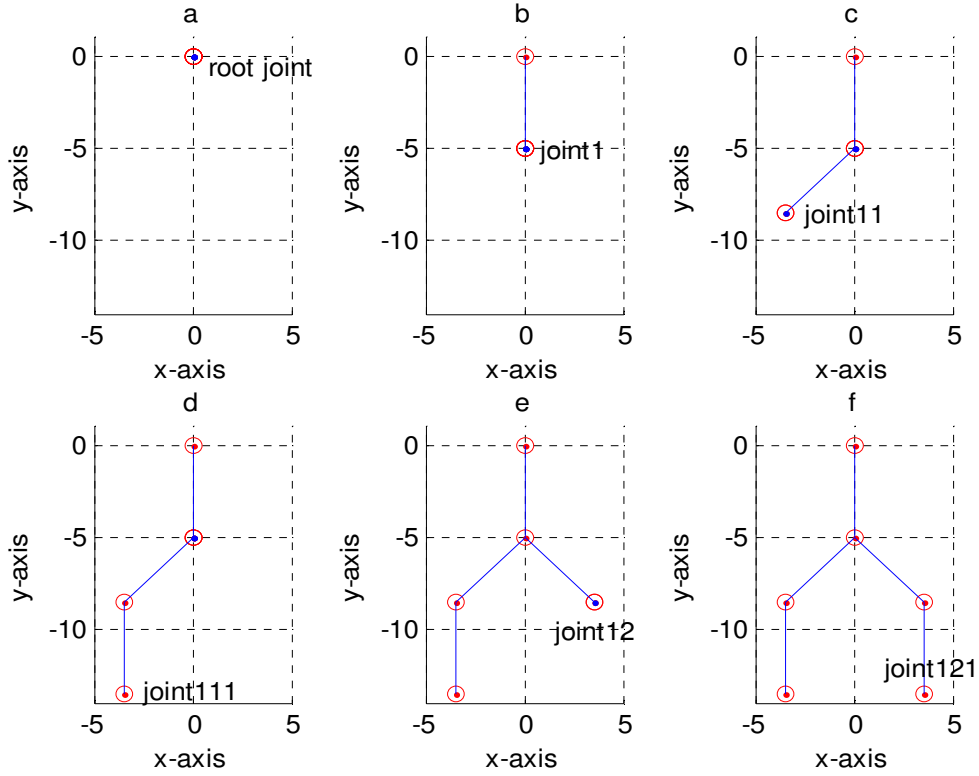


Figure 25 Step by step evaluation of the recursive skeleton formation

Figure 25 shows a 2D example of recursive construction steps of a skeletal configuration. Here, the skeleton is created solely by translations and rotations. The local transformation is a combination of local translation and rotation. Note that translation vector is in format $[x \ 0]^T$ which means all bones are initially aligned on x axis and x denotes the bone length. In Figure 25.f, the skeleton is totally formed. In that example, all bone translations (lengths) are same, and it is the vector $[5 \ 0]^T$. Bone local rotations are defined as:

bone1 (root_joint \rightarrow joint1): -90°	bone12 (joint1 \rightarrow joint12): 45°
bone11 (joint1 \rightarrow joint11): -45°	bone121 (joint12 \rightarrow joint121): -45°
bone111 (joint11 \rightarrow joint111): 45°	

Given the local translations and rotations, we can show how the position of joint111 is determined in Figure 25.d. In the d step, the recursion will lead to a combined transformation such as:

$$\text{combinedTransformation} = (\text{bone1 : rotation} * \text{bone1 : translation}) * (\text{bone2 : rotation} * \text{bone2 : translation}) * (\text{bone3 : rotation} * \text{bone3 : translation})$$

and position of *bone111* is calculated by

$$\text{joint111Pos} = \text{combinedTransformation} * \text{positionZero}$$

So, starting from *positionZero* (origin), joint111 is translated by $[5 \ 0]^T$, rotated by -90° , translated by $[5 \ 0]^T$, rotated by -45° , translated by $[5 \ 0]^T$ and rotated by 45° . In the end, all

these steps transforms joint111 from origin $(0,0)$ to $joint111Pos (-3.5355,-13.5355)$. If we keep the rotations same and give each bone different lengths such as

$$\text{bone1 (root_joint} \rightarrow \text{joint1): } [4 \ 0]^T$$

$$\text{bone11 (joint1} \rightarrow \text{joint11): } [2 \ 0]^T$$

$$\text{bone111 (joint11} \rightarrow \text{joint111): } [6 \ 0]^T$$

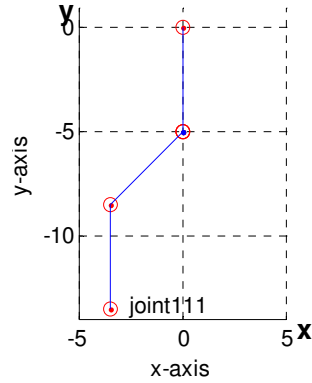


Figure 26 Skeletal configuration in the third recursion step with new bone lengths

then joint111 would have been at position $(-1.4142,-11.4142)$ just like in Figure 26.

Transformations other than rotations or translations can also be used for local transformation. Note that every joint initially stands at the origin and its local translation represents the bone originated by this very joint that aligns in x axis. There is no restriction about choice of axis. If each bone should be align on a particular axis such as $[1/\sqrt{2} \ 1/\sqrt{2}]^T$, then the translation vector should be defined as $[\sqrt{2} \ \sqrt{2}]^T$ to represent a bone with length of 2. Different kind of representations can be used to define a skeleton as we will explain in DQS. The same principle applies to 3D skeleton as well

3.3 Skeleton Subspace Deformation

We have already given brief information about SSD in section 1.3.1.1. In this section, we will explain SSD and its implementation in detail and make further analysis by giving examples and figures. First of all, let us remind the SSD blending equation.

$$\bar{v} = \sum_{k=0}^{n-1} w_k M_k M_{Rest,k}^{-1} v$$

The most problematic term in the equation is possibly $M_{Rest,k}^{-1}$. We mentioned that $M_{Rest,k}^{-1}$ transforms vertex v to the local coordinate of bone k in the rest pose. We will call $M_{Rest,k}^{-1}$ the skin offset transformation (SOT). To simplify things, we will assume that the vertex v on the skin (2D

mesh model) is influenced by only one bone with index k (no weights used). We will give a 2D example with a skeleton and a skin vertex v to make clear how things work, about SOT. In Figure 27.a, a 2D skeleton in dress pose is presented. That pose of skeleton is used to bind each vertex of the skin to related bones by determining bone influences on the vertex. The dress pose skeleton is formed and positioned in such a way that it is enveloped by the skin.

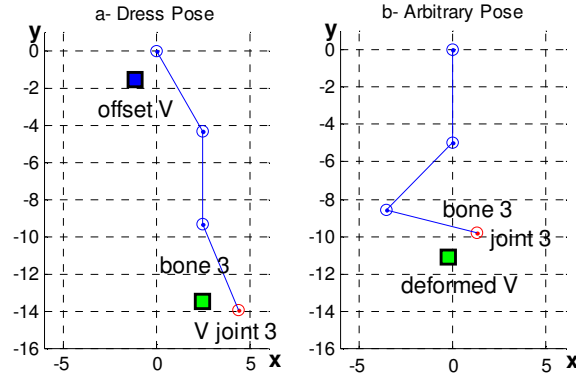


Figure 27 Use of SOT

In our example, vertex v denoted by the green square in Figure 27.a represents a vertex on the enveloping skin which is also in dress pose. And it is only influenced by bone 3 ($k = 3$). The dress pose of the skeleton is defined by the transformations

$$\begin{aligned} \text{bone1 (root_joint} \rightarrow \text{joint1): } & -60^\circ \ \& \ [5 \ 0]^T \\ \text{bone2 (joint1} \rightarrow \text{joint2): } & -30^\circ \ \& \ [5 \ 0]^T \\ \text{bone3 (joint2} \rightarrow \text{joint3): } & 22.5^\circ \ \& \ [5 \ 0]^T \end{aligned}$$

As described in previous section, the position of joint 3 (4.4134, -13.9495) is calculated by using

$$\begin{aligned} \text{combinedTransformationDress}_3 &= (\text{bone1 : rotation} * \text{bone1 : translation}) * \\ & (\text{bone2 : rotation} * \text{bone2 : translation}) * (\text{bone3 : rotation} * \text{bone3 : translation}) \\ \text{joint3Pos} &= \text{combinedTransformationDress}_3 * \text{positionZero} \end{aligned}$$

Consequently, joint 3 can also be transformed back to the origin by using the inverse of the combined transformation defined by

$$\begin{aligned} \text{combinedTransformationDress}_3^{-1} &= (\text{bone3 : translation}^{-1} * \text{bone3 : rotation}^{-1}) * \\ & (\text{bone2 : translation}^{-1} * \text{bone2 : rotation}^{-1}) * (\text{bone1 : translation}^{-1} * \text{bone1 : rotation}^{-1}) \\ \text{positionZero} &= \text{combinedTransformationDress}_3^{-1} * \text{joint3Pos} \end{aligned}$$

Here, $\text{combinedTransformationDress}_3^{-1}$ is the SOT of bone 3. In our example, skin vertices influenced by bone 3 are rotated by -22.5° , translated by $[-5 \ 0]^T$, rotated by 30° , translated by $[-5 \ 0]^T$, rotated by 60° and translated by $[-5 \ 0]^T$ in order to be transformed to local

coordinate frame of bone 3. In Figure 27.a, the vertex v (2.5,-13.5) is the dress pose (original) position denoted by green rectangle and $offsetPosV_3$ is the offset position which is computed by transforming vertex v by SOT_3 . $offsetPosV_3$ is denoted by blue rectangle and it resides in the local coordinate frame of bone 3. We calculate the value of $offsetPosV_3$ (-1.1475, -1.5957) by the equation.

$$offsetPosV_3 = SOT_3 * v = combinedTransformationDress_3^{-1} * v$$

Dress pose of a skeleton and skin pair is determined before the SSD animation and remains unchanged during the animation. Therefore, SOTs of each bone can be precalculated and reused during animation.

To deform a skin vertex, it is enough to find the combined transformation of the bone k in an arbitrary skeleton configuration and apply it to the $offsetPosV_3$. In Figure 27.b, deformed position of vertex v is represented by green rectangle for the arbitrary pose of skeleton. The arbitrary pose is defined by the transformations

$$\text{bone1 (root_joint} \rightarrow \text{joint1): } -90^\circ \ \& \ [5 \ 0]^T$$

$$\text{bone2 (joint1} \rightarrow \text{joint2): } -45^\circ \ \& \ [5 \ 0]^T$$

$$\text{bone3 (joint2} \rightarrow \text{joint3): } 120^\circ \ \& \ [5 \ 0]^T$$

Given such skeleton configuration, the deformed position of v which is influenced by Bone 3 will be transformed to its deformed position $deformedPosV_3$ (-0.2274,-11.0740) along with Joint 3 (1.2941, -9.8296).

$$deformedPosV_3 = combinedTransformationArbitrary_3 * offsetPosV_3$$

Given the equation $offsetPosV_3 = SOT_3 * v$, the relation between $deformedPosV_3$ and v can be defined as

$$deformedPosV_3 = combinedTransformationArbitrary_3 * SOT_3 * v$$

If we convert our representation to standard SSD representation

$$M_k = combinedTransformationArbitrary_k,$$

$$M_{Rest,k}^{-1} = SOT_k,$$

$$\hat{v}_k = deformedPosV_k$$

$$w_3 = 1 \text{ and } w_k = 0 \ \forall k, k \neq 3$$

$$\bar{v} = \sum_{k=0}^{n-1} w_k M_k M_{Rest,k}^{-1} v = w_3 M_3 M_{Rest,3}^{-1} v = \hat{v}_3, \text{ for the given example}$$

This time, we will observe the affect of multiple influencing bones by incorporating weights. We will give an example similar to previous one, but this time dress pose skin vertex v will be influenced

by both Bone 2 and Bone 3. Influences of bone 2 and bone 3 on v are $w_2 = 0.333$ and $w_3 = 0.667$ respectively. In Figure 28.a the skeleton in dress pose is bound to the skin vertex v denoted by the black triangle.

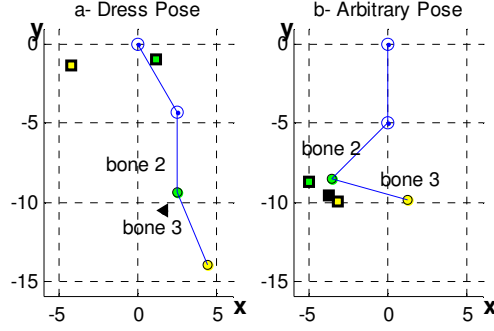


Figure 28 Weighted Blending

According to the dress pose of skeleton, SOTs of bone 2 and bone 3 are derived. Subsequently, offset positions of v are calculated for bone 2 and bone 3 which are denoted by green and yellow squares in Figure 28.a respectively. In Figure 28.b, offset positions denoted by green and yellow squares are transformed to their deformed positions \hat{v}_2 and \hat{v}_3 by their respective bones. One can infer that final deformed position \bar{v} can be derived from the weighted interpolation of \hat{v}_2 and \hat{v}_3 by looking at the expanded SSD equation

$$\bar{v} = \sum_{k=0}^{n-1} w_k M_k M_{Rest,k}^{-1} v = \sum_{k=0}^{n-1} w_k \hat{v}_k = w_2 \hat{v}_2 + w_3 \hat{v}_3$$

In Figure 28.b, \bar{v} which is denoted by black square is on the line connecting \hat{v}_2 and \hat{v}_3 . Note that, \bar{v} and is closer to \hat{v}_3 than \hat{v}_2 . This is due to the ratio between distance from \hat{v}_2 to \bar{v} and distance from \hat{v}_3 to \bar{v} are inversely proportional with the ratio between w_2 and w_3 . The relation can be denoted by $\|\bar{v} - \hat{v}_2\| / \|\bar{v} - \hat{v}_3\| = w_3 / w_2$

Now that we have more insight about SSD blending equation, we can move to implementation. Since SSD deformation is driven by bone transformations, we use a recursive routine analogous to the previous section to traverse the bones of the skeleton. This time, the pseudo code is a bit more complicated.

Function CalculateDeformedVerticesOfBone (currentBone, combinedTransformation)

begin

combinedTransformation =

combinedTransformation * currentBone:localTransformation;

currentBoneTransformation =

combinedTransformation * currentBone:SOT;


```

DeformInfluencedVerticesOfCurrentBone(currentBone, currentBoneTransformation)
for each child bone, childBone, of currentBone;
    CalculateDeformedVerticesOfBone (childBone, combinedTransformation);
end

```

Second pseudo code belongs to function *DeformInfluencedVerticesOfCurrentBone* and used to deform the vertices influenced by the *currentBone*.

```

Function DeformInfluencedVerticesOfCurrentBone( currentBone,
                                currentBoneTransformation)
begin
    for each influenced vertex of currentBone
        begin
            deformedVertex = deformedVertex + currentBoneWeight *
            currentBoneTransformation * originalVertex
        end
    end
end

```

Each bone contains a reference list to the influenced dress pose vertices along with the respective vertex weight list. In this pseudo code, $w_k \hat{v}_k$ is calculated for each influenced vertex v of bone k and is added to the deformed vertex \bar{v} . For this implementation to function correctly, all of the deformed vertices must be initialized to zero vertex. To deform a whole skin:

```

Iterate over all deformed vertices and set each to zero vertex;
CalculateDeformedVerticesOfBone (rootBone, identityTransformation );

```

We used a generalized transformation representation, but in SSD algorithm every transformation is represented by a matrix. In 3D case, transformations are 4x4 affine matrices and vertices are 4x1 vectors.

The linear blending of vertices causes some serious defects. Since SSD is widely used in virtual human animation, these defects have names such as collapsing elbow, twisting elbow and candy wrapper. We will explain the reasons of collapsing defect by giving two 2D examples in Figure 29 and Figure 30.

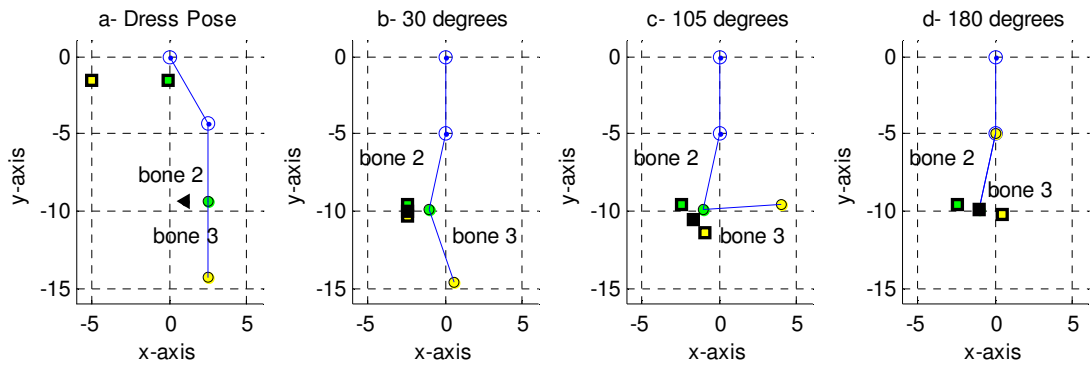


Figure 29 Collapsing Defect

Figure 29.a is the binding pose. The vertex weights are $w_2 = 0.5$ and $w_3 = 0.5$. According to the conjecture, bone 3 is at 0° in this state. In b, bone 3 adducts towards bone 2 with 30° which causes the deformed vertex \bar{v} to get a bit close to joint 2. In c, distance from \bar{v} to joint 2 decreased significantly because of the angle 105° . And finally, they totally overlap in d when it comes to 180° . Note that, bone 3 also overlaps with bone 2.

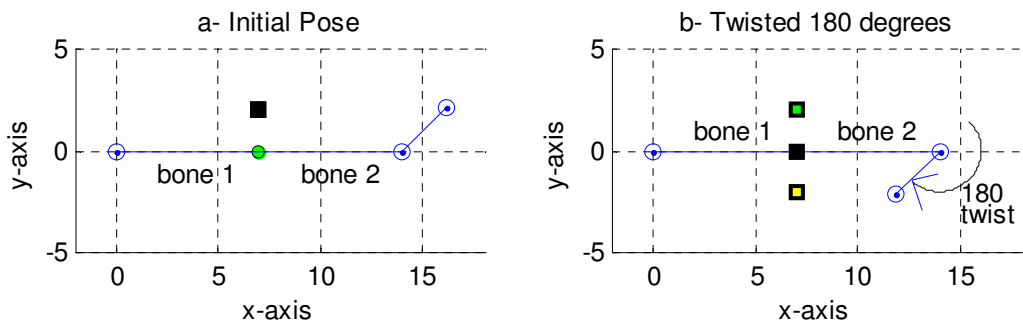


Figure 30 Collapsing defect when upper arm twisted 180 degrees

Another example is twisting elbow defect in 3D space. In Figure 30.a, there is a 3D skeleton in its initial pose. Bone 1 is analogous to lower arm and bone 2 is analogous to upper arm. In the initial pose upper arm is not twisted at all, so the deformed vertices \hat{v}_1 , \hat{v}_2 and blended deformation vertex \bar{v} are all located on the same position denoted by black square in a. Bone 2 is twisted around its own axis for 180° in b. \hat{v}_1 (green rectangle) keeps its previous position where \hat{v}_2 (yellow rectangle) rotates to its symmetrical position about the joint connecting bone 1 and bone 2. Given the weights $w_1 = 0.5$ and $w_2 = 0.5$, \bar{v} (black rectangle) is positioned on the joint, causing a collapse affect. Same defect can be observed when legs are twisted 180° which is shown in Figure 31



Figure 31 A collapse is observed as leg twists 180 degrees

The examples that we gave lead to serious collapse defects and volume losses of skin. Linear vertex or matrix blending causes erroneous outcomes. Researchers always try to come up with methods to overcome the undesired affects of linear blending while keeping the simplicity of deformation. So, it still remains an open research area.

3.4 Dual-Quaternions Skinning

In section 1.3.1.2, we already make an introduction to DQS. DQS make use of DQs to ensure a rigid deformation where no collapse and volume loss defects of SSD is observed. In his study, Kavan (2007) defined how an ideal transformation blending should be for a rigid skinning. He formalized and summarized the properties of such blending algorithms. Consequently, he proposed the DQS method which has these properties. While overcoming the SSD defects, DQS still retains its simplicity and real-time efficiency. For the same reasons, it is also possible to successfully map the implementation to graphics hardware.

Another great advantage of DQS is that it is easy to adopt an application using SSD to use DQS method. Because, DQS is able to use the SSD input data. To do so, one must convert the input representation to DQ representation. Most of these conversions may not pose any problem. In direct-x (.x) files, for example, rotations are already represented as quaternions. And translations are represented as 3x1 vectors. These representations can be easily converted to DQs. On the other hand, conversion of some input representations may be troublesome or even impossible. SOTs in .x files are represented as matrices composed of transformations which are not affine alongside the translations and rotations. Hence, there is no possibility for exact conversion from the matrix representation to DQ representation and data loss is inevitable. Before we begin with implementation details, we will go through how to convert between representations. In the previous chapter, we have covered how to create a DQ from an axis angle representation and translation vector. To continue with, we should take a look at how we can decompose a 4x4 affine matrix into 3D rotation and translation components. We assume that matrix only contains rotation and translation.

$$\begin{bmatrix} M_R & V_T \\ 0^T & 1 \end{bmatrix}$$

This is 4x4 affine matrix where 0^T is the transpose of 3x1 zero vector. M_R is the 3x3 rotation matrix. V_T is the 3x1 translation vector. On this basis, M_R is converted to quaternion to compose the non-dual part of the DQ where V_T is used for dual-part of the DQ. Eberly (2006) gives detailed information about the relation of quaternions, axis-angle representation and rotation matrices. Conversion methods and their derivations are also available in that source. Since all skin vertices are transformed by matrix multiplication, conversion from DQs to matrices is crucial. Given the DQ $(s, x, y, z, s_\epsilon, x_\epsilon, y_\epsilon, z_\epsilon)$ and a 4x4 matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

the relation between two can be defined as rotation matrix M_R

$$\begin{aligned} a_{11} &= 1 - 2 \cdot y^2 - 2 \cdot z^2 & a_{12} &= 2 \cdot x \cdot y - 2 \cdot s \cdot z & a_{13} &= 2 \cdot x \cdot z + 2 \cdot s \cdot y \\ a_{21} &= 2 \cdot x \cdot y + 2 \cdot s \cdot z & a_{22} &= 1 - 2 \cdot x^2 - 2 \cdot z^2 & a_{23} &= 2 \cdot y \cdot z - 2 \cdot s \cdot x \\ a_{31} &= 2 \cdot x \cdot z - 2 \cdot s \cdot y & a_{32} &= 2 \cdot y \cdot z + 2 \cdot s \cdot x & a_{33} &= 1 - 2 \cdot x^2 - 2 \cdot y^2 \end{aligned}$$

and translation vector V_T

$$\begin{aligned} a_{14} &= 2 \cdot (-s_\epsilon \cdot x + x_\epsilon \cdot s - y_\epsilon \cdot z + z_\epsilon \cdot y) \\ a_{24} &= 2 \cdot (-s_\epsilon \cdot y + x_\epsilon \cdot z + y_\epsilon \cdot s - z_\epsilon \cdot x) \\ a_{34} &= 2 \cdot (-s_\epsilon \cdot z - x_\epsilon \cdot y + y_\epsilon \cdot x + z_\epsilon \cdot s) \end{aligned}$$

After matrix is generated, it can used just like in the SSD implementation and it is also possible to use matrices with existing graphics API. Of course, converting all vertices to a dual-quaternion and doing quaternion multiplication and converting the results back to standard vectors is another option. But, it is not as efficient as matrix multiplication, since graphic APIs are capable of fast matrix multiplication.

Choice of blending function is very vital for the success of DQS. In his work, Kavan proposed, analyzed and compared two blending and an interpolation function. First one is Screw Linear Interpolation (ScLERP), which is the DQ version of famous SLERP. ScLERP interpolates two unit DQs where SLERP interpolates two unit quaternions. Formulation and further explanation about SLERP is given in Section 4.2. SLERP is introduced by Shoemake (1985). Problem with ScLERP is that it is not convenient for blending more than two DQs. Second one is DLB, which is very similar to QLERP. The relation between DLB and QLERP is similar to the relation between ScLERP and SLERP. DLB linearly blends DQs where QLERP linearly blends quaternions. DLB can blend more than two DQs, which is essential for the DQS algorithm. Since DLB and ScLERP works with DQs which define screw motion, it would be hard to visualize their relation. So, we will present a visual comparison by using SLERP and QLERP instead. We will use the word speed and term t (time), because interpolation is originally a subject of computer animation domain. Note that, DQS normally uses bone weights instead of t for interpolation or blending. In Figure 32, it is clearly shown that SLERP is an accurate interpolation with constant speed of angular distance.

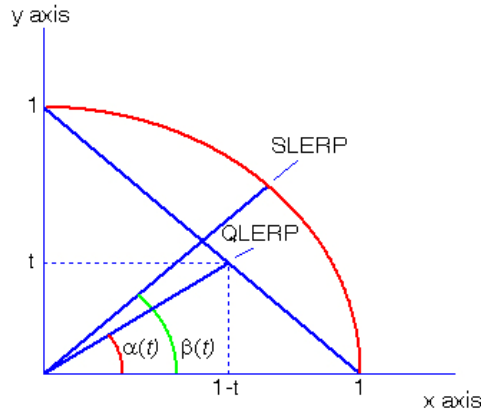


Figure 32 QLERP vs. SLERP

Because, the ratio of arc traveled and the angle covered in interval t is constant. But, QLERP just approximates this circular movement with a linear movement. Hence, QLERP exposes variable speed. Let's explain the subject with an example. Assume that we are trying to make a rotation about an axis which is constant during the animation. We start the animation from 30° at time $t = 0$ and end it at 230° at time $t = 200$. SLERP interpolation ensures that at each time interval $\Delta t = 1$ we get closer 1° to 230° . But QLERP can not produce a constant degree of rotation at each time interval $\Delta t = 1$. $\alpha(t)$ indicates the degree that is rotated by QLERP and $\beta(t)$ is the angle is rotated by SLERP at time t . Even so, QLERP approximates well enough and seems to suffice for the task. All of these apply for DLB and ScLERP in the DQ case. Besides, DLB is faster than ScLERP in computation. To improve the accuracy and provide constant speed, DLB is modified into Dual quaternion Iterative Blending (DIB). DIB iteratively approximates ScLERP. Due to the desired accuracy, it may take a few or more iterations to converge. Derived from DLB, DIB is capable of blending more than two DQs. Moreover, it provides a constant speed just like ScLERP. But, iterative procedure is time consuming. Since DLB has enough accuracy and low computational complexity, we decided to use DLB instead of DIB in our implementation. DLB equation is denoted by

$$\hat{q}_B = DLB(w; \hat{q}_1, \dots, \hat{q}_n) = \frac{w_1 \cdot \hat{q}_1 + \dots + w_n \cdot \hat{q}_n}{\|w_1 \cdot \hat{q}_1 + \dots + w_n \cdot \hat{q}_n\|}$$

where each w is the influencing bone weight for the vertex v and \hat{q}_B is a unit DQ. There is another delicate issue about blending that needs attention. Negative of a DQ is equal to itself ($\hat{q} = -\hat{q}$) and DLB is a linear blending function. Therefore, two DQs that normally strengthen the affect of a transformation could negate each other. To prevent that from happening, other DQs in the blending equation must be adjusted according to a reference DQ. The first DQ can be chosen as the reference \hat{q}_{REF} and then

if $\langle \hat{q}_{REF} | \hat{q} \rangle > 0$ then use \hat{q}

if $\langle \hat{q}_{REF} | \hat{q} \rangle < 0$ then use $-\hat{q}$

where \hat{q} is one of DQs in the blending function other than \hat{q}_{REF} . This idea is not a new one indeed, the two quaternion in the SLERP are assumed to conform to the same condition $\langle q_1 | q_2 \rangle > 0$. A DQS implementation for MAYA software package proposes a technique on this ("Dual Quaternion skinning in Maya,"). Choosing the DQ belonging to the bone with the highest influence on the given vertex as the \hat{q}_{REF} would offer better blending. We follow this technique in our implementation. The most important subject about DQS blending is that its result is just another DQ. Since DQs doesn't have any scaling effect, there won't be any volume loss on the 3D object which is deformed by DQS algorithm. The issue about SSD algorithm was that during weighted linear blending of rotation and translation matrices, the result matrix can contain scaling affects especially at the vertices close to joints.

We have everything needed to proceed to the implementation. In the initialization phase, list of influencing bones for each vertex is created and sorted in decreasing influence order. If the given SOT can not be converted to DQ (this is because combined transformations which only contain translations and rotations can be exactly converted to DQ), use the translation DQ as the SOT that would translate skin (original) vertices to their skin offset positions (local coordinate) for each influencing bone. As we mentioned before, the bone SOTs in the direct-x files can not be converted to DQs. These steps will be done once, but next steps will be repeated in each time frame. As the first step of deformation, combined transformation of each bone must be calculated recursively as suggested in Section 4.2. Transformations are first converted to DQ. Then DLB of influencing bones \hat{q}_B is calculated for each vertex. These steps are done with the pseudo code

```
// Initialization phase only
Create influencing bone list and sort them for each vertex
If bone SOTs can not be converted to DQs
begin
    Transform skin vertices to influencing bone local coordinate frames
end
    ⋮
// for each time frame deformation call DoDQS function
Function CalculateLocalTransformationsDQ (currentBone, combinedTransformation )
begin
    combinedTransformation =
        combinedTransformation * currentBone:localTransformation;
    convert transformation of currentBone (bone with index k ) to DQ,  $\hat{q}_k$ ;
    store  $\hat{q}_k$  to use it in blending function;
    for each child bone, childBone, of currentBone
        CalculateLocalTransformationsDQ (childBone, combinedTransformation);
end
```

end

Function DoDQS ()

begin

for each skin vertex v

begin

// to calculate blended DQ \hat{q}_B call blending function

// where there are n influencing bones for vertex v , in our case we call DLB

$\hat{q}_B = DLB(w; \hat{q}_1, \dots, \hat{q}_n);$

// convert \hat{q}_B to transformation matrix M_T

$M_T = DQ2Matrix(\hat{q}_B);$

// calculate deformed vertex \bar{v} by multiplying original vertex v with M_T

*// deformedVertex = blendedTransformation * originalVertex;*

$\bar{v} = M_T \cdot v$

end

end

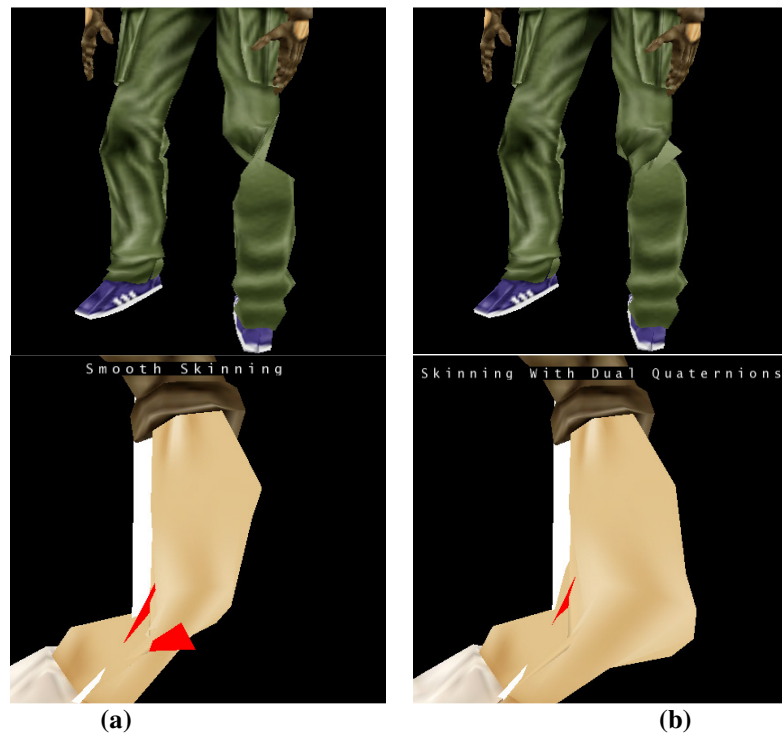


Figure 33 Note the volume loss and collapses in SSD (a) and compared to DQS (b)

Here, we give the comparisons of SSD and DQS from our tool. In Figure 33, you can see how DQS overcome defects of SSD

CHAPTER 4

ANIMATING THE SKIN AND ANIMATION DATA

4.1 Overview

Until now, we only focus on how to deform the skin for a given skeleton configuration. But, our ultimate goal is to animate a virtual character. To animate literally means to “to give life to”. Animating something is to move it or make it seem to move. Animation occurs when a series of still images are displayed in rapid sequence. The presentation rate (frequency) of still images is called frame rate or playback rate. The eye-brain complex of the observer assembles the sequence of still images and perceives it as continuous movement. A single image instantly presented to a viewer will leave an imprint of itself in the visual cortex for a short period of time which is called the positive afterimage. This phenomenon is known as persistence of vision. Because of persistence of vision, a sequence of closely related still images at a fast enough rate induces the sensation of continuous imagery. Even so, it is not continuous; the afterimages fill in the gaps between the images. Unless play back rate is fast enough, the image starts to flicker. Depending on conditions, the frame rate to maintain the persistence of vision varies. This rate is referred to as the flicker rate. There is another rate related to animation. This one is called sampling rate that is defined by the number of different images displayed per second. Sampling rate determines how jerky and rough the motion is. Two well known color formats, Phase Alternate Lines (PAL) and National Television Standards Committee (NTSC) use frame rates 25 and 29.97 respectively. These frame rates are accepted to be enough to prevent flicker Parent (2002). To accomplish smooth animation, the deformation method must operate with minimum 25 frame sampling rate or even more. This requires a deformation model with low computational complexity. Other important issues in animation are timeline, key-framing and frame interpolation. We will discuss these topics in Section 5.2. After comprehending these, virtual character animation will become more complete.

Another important subject about the animation is the animation data source. We will discuss what kind of data is needed to animate a 3D virtual character and the methods to creating such data. We will also explain what kind of data formats and sources we have used in our thesis study.

4.2 Key-Framing and Frame interpolation

An animation has a hierarchical structure. Entire animation is called the production time line. Production divides into major episodes with an associated staging area which is called a sequence. A shot is a continuous camera (CGI uses virtual cameras) recording. And as we already know, frames stand for the still images to be displayed successively in animation Parent (2002). Structure of the animation is shown in Figure 34. Professional 3D animation packages such as MAYA, MotionBuilder and 3D Studio Max conforms to this structure.

PRODUCTION TIME LINE																			
SEQUENCE A							SEQUENCE B												
Shot 1			Shot 2				Shot 1					Shot 2				Shot 3			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Figure 34 Structure of an Animation Production

A special set of frames are called key frames (also known as extremes). Originally, in animation process, master animators identifies and produce key frames to aid in confirmation of character development and image quality. Consequently, associate and assistant animators produce the frames between the keys; this is called tweening (in-betweening). In computer graphics, an animator identifies the key frame and computer automatically does the tweening and generates intermediate frames between two key-frames by using frame interpolation and physical rules if any defined.

In our tool we use only frame interpolation techniques to do the tweening, since we implement only GMs. Key frame interpolation is generally done by using the time parameter t which conforms to the condition $0 \leq t \leq 1$. For $t = 0$ and $t = 1$, current frame is exactly equal to the first and second key frames respectively. As time increases from 0 to 1, generated still image becomes more similar to second key frame image. Interpolation is applied to the parameters which defines the generated image. For instance, transformations which define the bones of skeletons are interpolated in our case. In Chapter 2, we investigate and discuss the representations. We mentioned that choice of orientation have significant affect on the success of the interpolation and ultimately, success of animation. We see that axis-angle or quaternion representation should be used for the better interpolation. In the axis-angle representation, interpolation of the first key axis-angle (α_1, a_1) and second key axis-angle (α_2, a_2) should done by interpolating the rotation angles α_1, α_2 and the 3D axis vectors a_1, a_2 separately. To interpolate axis a_1 and a_2 , we first retrieve the angle between a_1 and a_2 by using

$$\phi = \cos^{-1} \left(\frac{a_1 \cdot a_2}{|a_1| \cdot |a_2|} \right)$$

Subsequently, orthogonal vector a_o is calculated by cross product of a_1 and a_2 ($a_o = a_1 \times a_2$). At time step t , angle ϕ_t between the axis a_1 and a_2 about vector a_o is found by applying linear interpolation formula $\phi_t = (1-t) \cdot \phi_1 + t \cdot \phi_2$. And finally interpolated axis is derived from $a_t = R_B(t \cdot \phi) \cdot a_1$. Parent (2002).

Interpolation of quaternions is not a linear, but a spherical one. We used SLERP to spherically interpolate quaternions of key frames q_1 and q_2 at time step t . We obtain theta by using formula $\theta = \cos^{-1}(q_1 \bullet q_2)$. When the condition $q_1 \bullet q_2 \geq 0$ is satisfied, then interpolation along the shortest rotation path is acquired. If $q_1 \bullet q_2 < 0$ is the case, then $-q_2$ should be used instead for the shortest path. SLERP function is defined as

$$SLERP(q_1, q_2, t) = (((\sin(1-t) \cdot \theta)) / (\sin \theta)) \cdot q_1 + (((\sin(t \cdot \theta)) / (\sin \theta)) \cdot q_2$$

Scaling and translation transformations are generally represented by 3D vectors. Linear interpolation $V_t = (1-t) \cdot V_1 + t \cdot V_2$ is enough to successfully interpolate these transformations. Linear interpolation is first used for animation by Burtnyk and Wein (1971). For a successful animation, one should avoid using matrix interpolation directly because it produces erroneous results. If quaternion, scaling and translation data can be decomposed from the matrix without loss of data, they should be interpolated instead of matrices.

In Table 2, an animation of a single bone is given with four key frames. Each key frame i is defined by translation, rotation and scaling transformations.

Table 2 Animation key frames of bone defined by quaternion rotation, scaling and translation

i	Time T_i ms	Quaternion q_i	Scaling Sc_i	Translation Tr_i
1	0	q_1	Sc_1	Tr_1
2	200	q_2	Sc_2	Tr_2
3	400	q_3	Sc_3	Tr_3
4	600	q_4	Sc_4	Tr_4

Given the time $T = 260$ ms, first i and second $i+1$ index of the key frames are found according to the condition $T_i < T < T_{i+1}$. Here, $200 < 260 < 400$, so $i = 2$. Then interpolation step t is calculated by formula

$$t = \frac{T - T_i}{T_{i+1} - T_i} \text{ where } 0 \leq t \leq 1.$$

For $T = 260$, $t = (260 - 200) / (400 - 200) = 0.3$. Then, interpolated translation vector Tr_t and scaling vector Sc_t are denoted by $Tr_t = (1-t) \cdot Tr_i + t \cdot Tr_{i+1}$ and $Sc_t = (1-t) \cdot Sc_i + t \cdot Sc_{i+1}$ respectively. So, interpolated translation and scaling for time

$T = 260$ is $Tr_I = 0.7 \cdot Tr_2 + 0.3 \cdot Tr_3$ and $Sc_I = 0.7 \cdot Sc_2 + 0.3 \cdot Sc_3$. We can obtain q_I by using SLERP

$$q_I = \left(\frac{(\sin(0.7) \cdot \theta)}{(\sin \theta)} \right) \cdot q_2 + \left(\frac{(\sin(0.3) \cdot \theta)}{(\sin \theta)} \right) \cdot q_3$$

where $\theta = \cos^{-1}(q_2 \bullet q_3)$. After we acquire interpolated transformations related to bone k , we finally combine them to acquire local transformation of that bone. The combining order is important, since combination of transformations is not commutative. It should be done in the order, translation, rotation and scaling. If we represent the local transformation in matrix form as in SSD, it would be

$$M_k = M_{SCALING} \cdot M_{ROTATION} \cdot M_{TRANSLATION}$$

The only term calculated for each time T is the local transformations of the bones. Virtual character is animated by calculating skeleton configuration in each animation frame and deforming the skin via DM such as SSD or DQS.

4.3 Animation Data

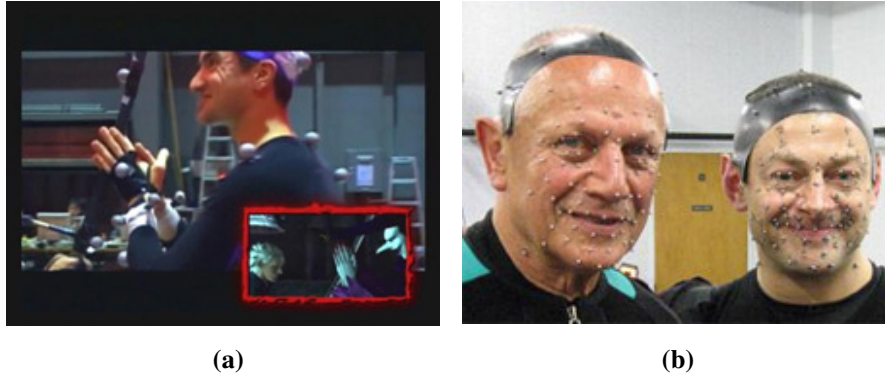
SSD and DQS operate with bone transformation data. In the previous section, we explained how transformation data such as rotation, scaling and translation can be used to animate a 3D virtual character via key-framing and interpolation. There are various file formats which contains such information. When we first started the implementation of our tool, we used the '.x' (DirectX) file format which is proposed by Microsoft and used in DirectX framework to get the animation data. Indeed '.x' files usually comes as a full package which contains information about animation, 3D virtual character mesh data, skeleton hierarchy, bone weights, etc. For that reason, it came to have a complex structure and it is a laborious to parse and use it. If you are using DirectX framework of Microsoft on the other hand, you will be able to use their libraries to easily read '.x' files. Unfortunately, this wasn't the case for us, since we make our implementation with OpenGL. As a result, we have parsed the file all by ourselves. We made use of a web site (Coppens) to do so. Thanks to Paul Coppens for his efforts for preparing a nice tutorial about parsing '.x' files. Information on '.x' files can be found in (Coppens).

For the tool we developed, it is of no importance whether bone transformation data is generated with a professional 3D animation tool by an artist or captured in a motion capture laboratory (mocap lab). One of our purposes is to demonstrate movement of a particular human in virtual environment for medical purposes. So that, a physician would be able to inspect a patients movements for analysis and diagnosis by the help of the tool we provide or students under medical education would use it to inspect and learn the movement patterns which belongs to different kind of gait disorders. This is why we should make use of mocap. We have investigated whether there was any available mocap data about gait disorders, but we failed to find any. Then we have realized that we would have to retrieve our own mocap data. For that reason, we make contact with our colleagues in Koç University. They let us use their mocap lab to capture medical gait disorder data. It was a multi-camera mocap system which is also known as optical mocap system. We will give more detail about motion capture systems

in Section 4.5. To briefly explain it, we can say that an optical system records any moving object which has reflective markers attached and tracks these markers to extract their 3D position data. Unfortunately, 3D position data can't be directly used in character animation. To do so, we have to find some means to convert this data into the appropriate format which can be used by our animation framework. We came up with a simple and intuitive method, but it didn't produce satisfactory results. We will give the details about our method of converting 3D marker position data into transformation representation in Section 5.6.2. Successful mapping of 3D marker positions data to bone transformations isn't a straightforward process (Zordan & Van Der Horst, 2003). Since proposing such an algorithm was not one of our purposes, we decide to use MotionBuilder 7.5 which is professional animation software to extract bone transformation from marker data and export the processed motion capture data as Biovision hierarchical (BVH) data file. BVH files contain skeleton hierarchy and bone transformations at each time frame. Zordan and Van Der Horst (2003) proposed a method for extracting bone transformations. Brief discussions about the approaches which MotionBuilder and Zordan & Van Der Horst (2003) use will be made in Section 5.6.3. Further explanations about BVH file format is given in (Meredith & Maddock, 2000).

4.4 Motion Capture

Mocap or Motion tracking is a digitally movement recording technique. In general sense, mocap is the process of recording a live motion event through the placed markers (or sensors) on or near each joint of the body. The recorded data is then converted into usable mathematical terms by tracking these key points, regions or segments in space over time and combined to obtain a three dimensional representation of the performance. Related software records angles, velocities, accelerations and impulses, providing an accurate digital representation of the movement. Consequently, we can present a live movement or performance with a virtual and digital performance by using this technology. The captured object could be anything that makes some motion in the real world. Mocap started as a photogrammetric analysis tool in biomechanics research in the 1970s and 1980s. Later on, it expanded into education, training, sports, etc. As the technology matured, computer animation for cinema, video, advertising and game production became as its widespread use. Mocap is very suitable to achieve lifelike movement than manual animation. Besides, it saves time and money. Motion capture technology is used in several productions such as Lord of the Rings, Polar Express and Happy Feet. Most of the character animations in games such as Devil May Cry, Heavenly Sword and God of War are captured in mocap labs as well. Figure 35 shows how games take advantage of mocap technology. The examples given in Figure 35 are captured from optical mocap systems. As you can see the scale of the markers can vary according to the goal.



(a) (b)
Figure 35 Optical mocap data used in games such as Devil May Cry (a) and Heavenly Swords (b)
 (Courtesy of CAPCOM (a) and Ninja Theory (b))

4.5 Motion Capture Systems

There are different ways of capturing motion. Different systems use different input devices such as markers, sensors, etc. As we stated before, markers (or sensors) are positioned on the key points that best represent the motion of the subject's different parts. How these input devices are placed on subject also differ according to the mocap system used. It is possible to divide mocap systems into three main categories such as

- 1) Optical Systems
- 2) Mechanical Systems
- 3) Electromagnetic (magnetic) Systems

Optical Systems: These are camera-based systems, where the cameras are the sensors and the reflective markers are the sources. These kinds of systems use cameras that digitize different views of the movement. By using computer vision and image processing algorithms, it is possible to extract the 2D positions of markers from each camera shot for a particular time of the movement. Since it is possible to construct 3D position from at least two 2D position inputs, a mocap system can retrieve 3D positions of all markers by using images retrieved from multiple cameras. Affective field of view where the movement can be captured and how successful the data can processed depends on the number of cameras used and their resolutions. Of course there are other important factors such as choice of markers, the configuration of the mocap lab, the quality of the tracking software, etc. Optical systems were primarily developed for biomedical applications such as analysis of athletic performance and sports injuries. Performers can act more freely because markers used in optical systems don't prevent movements of them. But, markers sometimes become unavailable to be tracked since they are visually blocked camera recording. After recording complete, it takes time for the software to extract 3D positions from images. It is possible to record movements in larger spaces relative to other systems.

Electromagnetic systems use static magnetic transmitter and magnetic receivers which are worn by performers to record movement. Since each sensor can produce 6 degree of freedom output, fewer sensors relative to optical systems are sufficient. Unless affected by external magnetic fields, positions are accurate. Data acquisition is instant since no process is needed to be applied on sensor data.

Sampling rate is insufficient for fluid and fast movements. Especially, older electromagnetic systems require performers to wear uncomfortable apparatus such as cables which prevents freedom of movement.

Last factor is the mechanical system. Performer has to attach exoskeleton like suit which is composed of straight metal pieces to her/his body by hooking their back. As the performer moves, the exoskeleton suit moves in the same way. The movement of exoskeleton is recorded. There are no magnetic fields or obstacles to prevent successful recording. Since exoskeletons movements are relative to its own coordinate system, there is no way to know the orientation or the position of the performer.

It is also possible to design combinations of two or more of these technologies to reduce the inherent limitations. New technologies are also becoming available, ultimately aiming to make a real-time tracking of an unlimited number of key points or all the segments of the person with no space limitations at the highest frequency possible with the smallest margin of error

CHAPTER 5

FEATURES OF THE ANIMATION TOOL

5.1 Overview

In this Chapter, we will give detailed information about the features of our animation tool. And in Chapter 6, we will explain what we have done so far and what the outcomes were. In order to see the big picture, we will briefly explain all of the steps in the whole process pipeline and show the explanations, outputs and inputs of each step are given in Figure 36.

5.2 Interaction with Objects in our Animation Tool

Our animation tool renders 3D dimensional scenes and projects it as a 2D image to be displayed on the monitor. The tool has some capabilities for mouse interaction and keyboard input. Mouse interaction in 3D applications is not trivial as in most 2D applications. The details about how mouse selection mechanism operates are given in Section 5.2.1. We can at least state that mouse operations are directly affected by the multiple transformation which are applied to the objects and the scene. Though mouse selection can be handled by OpenGL mechanisms, dragging a 3D object over the scene requires more effort. This will be explained scene in Section 5.2.2. Mouse and keyboard handlers we implemented make use of object oriented concepts such as polymorphism. Briefly, polymorphism provides us the means to define separate mouse and keyboard handlers for various kinds of object with different interaction needs with ease. There are two kinds of objects rendered in our scenes which have different properties. These are skeleton objects and 3D mesh objects. They have separate mouse handlers classes such as MeshMouseHandler and SkeletonMouseHandler classes which are extended from AbstractMouseHandler class

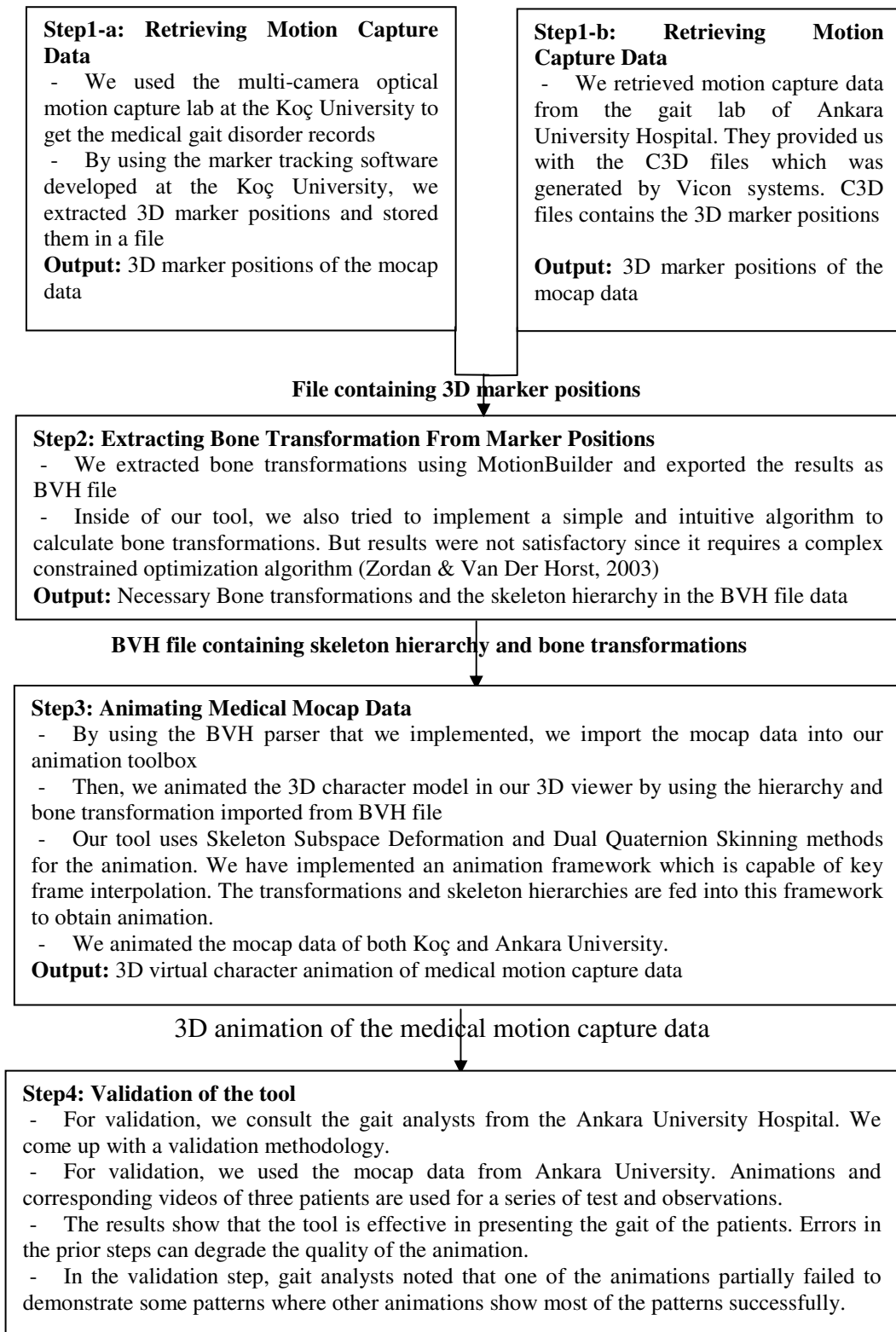


Figure 36 Summary of steps from motion capture to validation.

5.2.1 Mouse Selection in the 3D Scene

There is a handy mouse selection mechanism that is called *selection*. This mechanism provides a special rendering mode for selection. While in this rendering mode, OpenGL detects all the primitive objects in the viewing frustum and put their identification numbers (names) in a buffer in an order so that first object in the view towards the near plane of frustum (Figure 37) will be the first element of the buffer. These IDs are assigned to each primitive by using API functions. One can make use of this rendering mode so that rendering frustum will be determined by the mouse position. It is not a must to select only the objects under the mouse pointer. By adjusting the center, width and height parameters of the view, it is possible extend the mouse selection area to select primitive objects close to mouse. These parameters are passed to an OpenGL API function. A sample code written in C++ from our tool implementation is given in Appendix B.1 to explain how mechanism operates

As we already mentioned OpenGL fills the buffer with the hit records related to every single primitive rendered in select mode (`glRenderMode(GL_SELECT)`). There is no requirement about primitive objects having a unique name (ID). So, a group of primitives could be considered as one object in the mouse selection process. Finally using the count of hits and the IDs in the buffer, it is possible get appropriate response from the mouse event.

5.2.2 Mouse Drag Action

Dragging an object over the scene is another mouse interaction issue. This problem can not be solved by only OpenGL selection mechanism. Imagine that you have selected an object which is at 3D coordinates $(3, -4, 7)$ and 2D screen coordinates $(50, 28)$, then you dragged the your mouse over your screen to 2D screen coordinates $(65, 20)$. Unless a rule is defined there is no way to tell which 3D coordinate the object is dragged. First thing which must be considered is the viewing frustum. Viewing frustum is the region of space that is displayed on the screen. Objects outside that box are out of the view range and are not rendered on the screen. Viewing frustum is constrained by the near and far planes. Think mouse pointer as a laser gun that fires a ray when you click on mouse button. This ray penetrates first the near plane and then the far plane of the viewing frustum. This ray can be represented by a line function in the viewing frustum in Figure 37. The position and orientation of this line can be calculated from 2D coordinates of mouse pointer and the viewing frustum. Perspective projection matrix shapes the volume of frustum and modelview matrix determines its position and orientation.

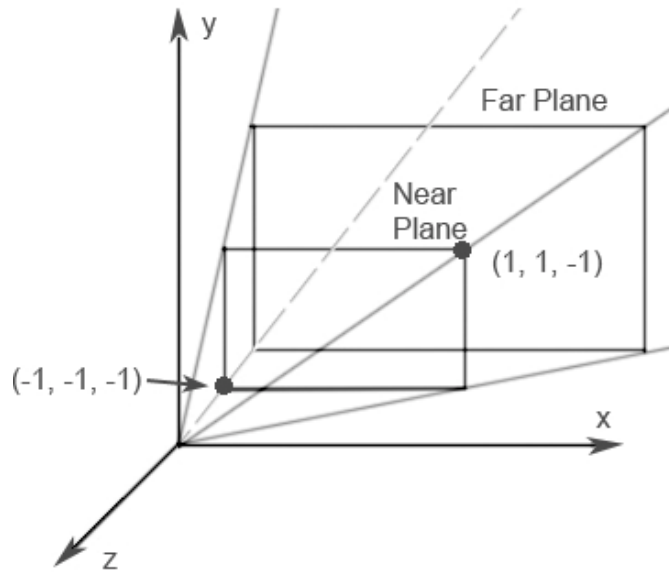
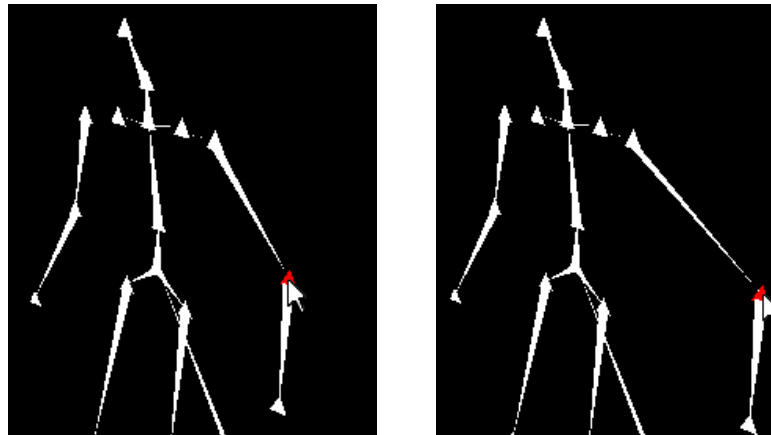


Figure 37 Viewing frustum

The model-view matrix determines how the vertices of OpenGL primitives are transformed to eye coordinates and the projection matrix transforms these eye coordinates to clip coordinates. Projection matrix, modelview matrix and the 2D screen position of mouse pointer determines the parameters of the line function which we liken to a ray. The c++ implementation of calculating the parameters of the line function coming from the mouse pointer and passing through the near and far planes is given in Appendix B.2.

After the consequent calls to the functions which are to retrieve projection of mouse pointer on near plane (`retrieveProjectionofMousePointerOnNearPlane`) and line function that is calculated by using the inverse of view matrix (`retrieveLineFuntionDataFromInverseOfViewMatrix`), line function parameters are finally retrieved. Now we have a line function coming out from our mouse pointer composed of infinitely many points. Now getting back to our example where an object is moved by dragging mouse to $(65,20)$ coordinates, which 3D point should we choose out of infinitely many line points? To make the dragging operation more usable and intuitive, we implement it such that an object could be only dragged along one axis at a time. To do so we must define a plane function which is aligned along the 'x' axis such as 'xy' or 'xz' plane. This plane should pass through the 3D point $(3,-4,7)$ which was the position of the object before dragging operation. The point where we drag the object should be chosen as the intersection point of the mouse ray and the plane we defined. From keyboard, the user inputs 'X', 'Y' or 'Z' buttons to determine axis for dragging. The line and plane intersection is calculated by the function given in Appendix B.3.

In Figure 38, a mouse drag event is shown where elbow joint is dragged from its initial position in (a) to final position (b). Note that the selected joint changes its color.



(a) (b)
Figure 38 A drag event along x axis on elbow joint from (a) to (b)

5.3 Animation Infrastructure of the Tool

In Section 4.2, we explained the animation basics and key-frame interpolation. Reading animation data from BVH or DirectX files, our tool creates a data structure that contains animation key-frame data. This data structure is composed of the instances of AnimationSet, Animation, TranslationKey, RotateKey, ScaleKey class instances. On the top of this data structure stands AnimationSet. Each AnimationSet structure contains a whole animation sequence data as seen in Figure 39. AnimationSet contains Animation instances which are registered to each bone.



(a) (b)
Figure 39 (a) is a walking animation sequence and (b) is a jogging animation sequence

Each Animation instance contains the transformations of the bone that it is registered through the animation sequence. These transformations are contained under three distinct lists in Animation instance. These are translation, rotation, and scaling lists composed of key-frame elements TranslationKey, RotateKey, ScaleKey respectively. Each key in the list has a time value to be used as the interpolation parameter. As we explained in Section 4.2, it is possible to calculate the transformation by using these time values and key-frame transformations. Interpolation of TranslationKey and ScaleKey elements are calculated by linear interpolation. To interpolate

RotationKey, SLERP is used where each RotationKey is represented by quaternions. Finally, interpolated transformation of these are combined into one to find the bone transformation at a particular time t

5.3.1 Animation Using Input Files

During this thesis study, we parsed three kinds of file for animation. First one was '.x' file format of Microsoft's DirectX graphics library. The second one was the '.bvh' file format which is a widely used format suggested by Biovision. The last one was an unformatted and raw data file which only contains joint positions of a skeleton. We will give details about how second file format is handled in Section 5.6.1.

There exist two kinds of '.x' file format which are binary and ASCII. We only parse a ASCII '.x' file. Even the ASCII format was quite complex. We parsed '.x' file based on the tutorial given in (Coppens). DirectX file format contains all the necessary information for a 3D animation. It contains

- Skeleton hierarchy and bone names
- SOT of each bone in matrix representation
- Vertex data of the mesh, normal of these vertices
- A reference to the texture file of the mesh, and related texture coordinates
- Bone weights of vertices
- And for animation, it contains the bone transformation at each key-frame and time of these key-frames

Animating '.x' files were really successful, every necessary data was provided and the data was well prepared. For instance, bone weights were most likely calculated by using professional software. Details about DirectX files can be found in (Coppens).

BVH was the file format which we used to display gait disorders. This file format contains only

- Skeleton hierarchy and bone names
- For animation, it contains the bone transformation at each key-frame and time of these key-frames

Unlike '.x' files BVH files doesn't have any mesh, bone weight and SOT information. By using the data of a BVH file, only a skeleton could be animated. In order to achieve a complete animation, we decided to use the mesh data from the '.x' file. But, we still needed the bone weights and SOTs for a 3D character model animation. So, we added features to calculate and assign bone weights to each vertex. We will explain how bone weights are calculated in Section 5.5. We also added the feature to calculate SOTs. Calculation of SOTs will be explained in Section 5.7.

5.3.2 Deformation by User Input

It is also possible to deform the mesh by direct user input. To do so, user must select a bone to transform by using mouse. By using keyboard, user can rotate the selected bone around global x, y and z axis. Unfortunately, this is not a comfortable way to transform a bone. Bone transformation should be done by mouse input and selected bone should be rotated around its own local axis not

around the global axis. Using bone local axis is a more intuitive way to deform the model. As user rotates the bones of the skeleton, character model will deform in the same way. This event is shown in Figure 40.

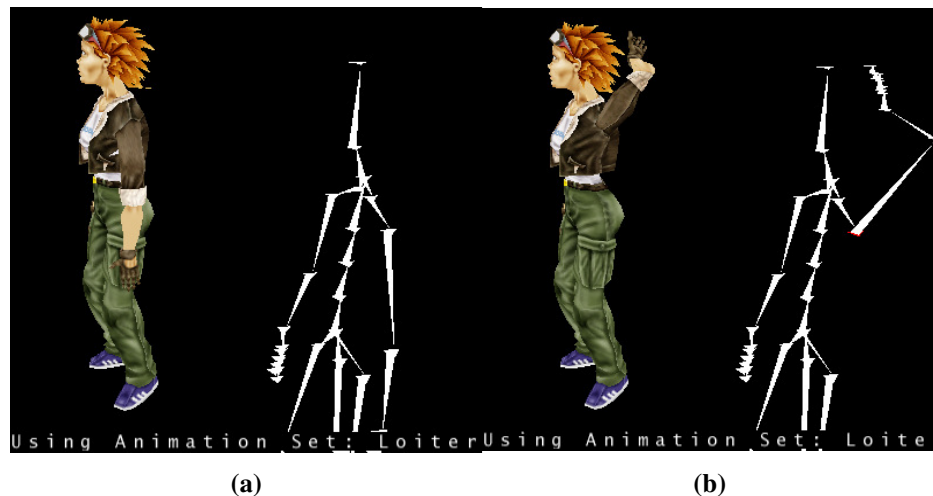


Figure 40 By using direct user input, model is deformed from its left upper arm and left forearm. Initial model is displayed in (a) and manually deformed one is in (b)

After skeleton bones are manually deformed by the user, these deformations propagate to the animation sequences when the animation is played.

5.4 Moving Over the Scene

We implemented a camera class for our tool to move over the scene. The class operates on quaternions. It has attributes such as HeadingDegree, PitchDegree and ForwardVelocity. PitchDegree and HeadingDegree are used to define quaternions to rotate around x axis and y axis respectively. To calculate the orientation of the camera, these two quaternions are multiplied. PitchDegree and HeadingDegree can be modified by directional buttons of keyboard. The orientation of the camera determines a direction which it looks towards. This direction vector can be used to move the camera. Camera speeds up and slows down in that direction by using 'W' and 'S' buttons respectively. By using the ForwardVelocity attribute and direction of camera, new position (translation) of camera is calculated. Finally camera translation and orientation are used to calculate modelview matrix. Same scene is shown from different camera orientation and positions in Figure 41.



(a) (b)
Figure 41 Same scene is displayed from different camera views (a) and (b)

5.5 Calculating Bone Weights

As mentioned already, SSD and DQS deform character by using bone weights. When we first animate the model in the DirectX file, bone weights for each vertex was available for the skeleton hierarchy given in the '.x' file. But, the medical mocap data we retrieved represented a different skeleton hierarchy. To animate the model with a skeleton hierarchy, we had to calculate and assign bone weights to vertices. Bone weights are inverse proportional with the distance between a bone and a vertex. So, we calculate the Euclidian distances between the vertices and the bones. Assuming that a bone is a line fragment which has no volume and thickness, we utilized the distance formula of a point to a line. We will give the details of distance calculation in Section 5.5.1. Afterwards, these distances are used to calculate weights which we will explain in Section 5.5.2. For this method to work, skeleton must be placed in the 3D character model and well aligned with the model just like in Figure 42. Alpha blending can be activated to make model transparent by pressing 'B' button. Modifying skeleton inside to an opaque model is not possible. Note that best weight assignment is done on the character in the T-Pose.

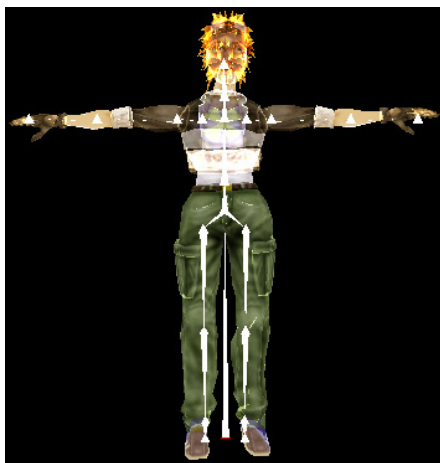


Figure 42 Skeleton is aligned in T-Pose to the transparent model in T-Pose

5.5.1 Calculating Vertex Bone Distances

We assume that bones can be approximated by to be line functions. Line functions are infinitely long, but our bones are not. They have a starting and ending point. While calculating distance d , this fact should be considered. In Figure 43-a, distance of the vertex v to the bone can is equal to the distance between v and its projection v_p . Bone is a line segment between the start joint p_s and end joint p_e . When v_p is out of that interval, we must use the distance to the start or end point as shown in Figure 43-b

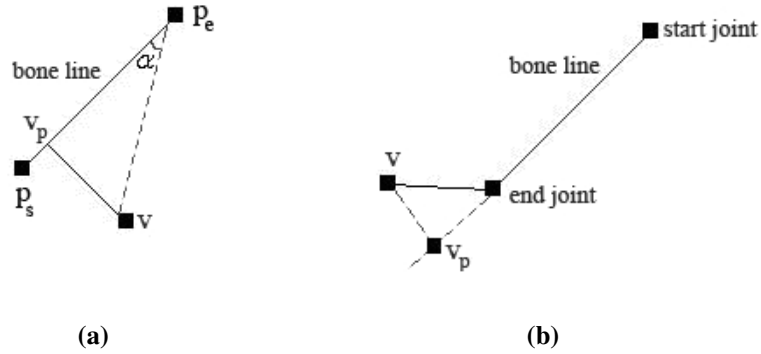


Figure 43 Same scene is displayed from different camera views (a) and (b)

There is a relation between vectors $(p_e - p_s)$ and $(p_e - v)$. Such that

$$(p_e - p_s) \bullet (p_e - v) = \|p_e - p_s\| \cdot \|p_e - v\| \cdot \cos(\alpha) \quad (5.1)$$

where α is the angle in between. From trigonometry

$$\|p_e - v_p\| = \|p_e - v\| \cdot \cos(\alpha) \quad (5.2)$$

By using (6.1) and (6.2), we can derive the equation

$$\|p_e - v_p\| = \frac{(p_e - p_s) \bullet (p_e - v)}{\|p_e - p_s\|} \quad (5.3)$$

Another geometric relation is already known

$$v_p = p_e + \|p_e - v_p\| \cdot \left(\frac{p_s - p_e}{\|p_e - p_s\|} \right) \quad (5.4)$$

From (5.3) and (5.4), we can derive the equation

$$v_p = p_e + \frac{(p_e - p_s) \bullet (p_e - v)}{\|p_e - p_s\|^2} \cdot (p_s - p_e)$$

where p_e , p_s and v are known variables. So, distance to bone can be defined as

$$d = \begin{cases} \|v - v_p\| & \max(\|p_e - v_p\|, \|p_s - v_p\|) \leq \|p_e - p_s\| \\ \min(\|p_e - v\|, \|p_s - v\|) & \text{otherwise} \end{cases}$$

5.5.2 Bone Weight Calculation Algorithm

In Section 5.5.1, we formulated how to find the distance of a vertex to a bone. By using distances, we will propose an algorithm to calculate bone weights. We will calculate the weights that belong to vertex v . Assume that v is influenced by three bones and distances of v to bone1, bone2 and bone3 are denoted by d_1 , d_2 and d_3 , respectively. Similarly weights of bone1, bone2 and bone3 for v are w_1 , w_2 and w_3 , respectively. All weights must be normalized so that $w_1 + w_2 + w_3 = 1$. To calculate w_1 , w_2 and w_3 , we will follow these steps

- 1- Find and store the distance d_{\min} closest (most influencing) bone to v

$$d_{\min} = \min(d_1, d_2, d_3)$$

- 2- Calculate intermediate weights w'_1 , w'_2 and w'_3 after choosing an appropriate base b ,

$$w'_1 = \frac{1}{b^{(d_1/d_{\min})}}, \quad w'_2 = \frac{1}{b^{(d_2/d_{\min})}} \quad \text{and} \quad w'_3 = \frac{1}{b^{(d_3/d_{\min})}}$$

- 3- Calculate normalized weights w_1 , w_2 and w_3 from w'_1 , w'_2 and w'_3 . To do that, we must find the sum of intermediate weights $w'_{sum} = w'_1 + w'_2 + w'_3$ and

$$w_1 = w'_1/w'_{sum}, \quad w_2 = w'_2/w'_{sum} \quad \text{and} \quad w_3 = w'_3/w'_{sum}$$

The base b is selected so that smooth weight transitions around joint points are possible and far vertices to the joints are only affected by one bone. We used $b = 100$

5.6 Reading and Animating Raw Mocap File

In Section 5.3.1, we mentioned that there was an unformatted and raw data file which only contains 3D joint positions of a skeleton. The output of the software developed for optical motion capture lab in the Koç University was the 3D marker positions. We made a Matlab implementation to map these marker positions to a skeleton by converting them to joint positions of that skeleton. This mapping was not a generic one; it was a case specific and 'ad hoc heuristic' solution. Output was a simple text file which contains a 3D joint position of skeleton (three floating point values) on each line. If there are n joint nodes, first n lines give the joint positions for first key-frame, second n lines give joint positions for the second key-frame and so on. So there shall be $m \times n$ number of lines in the text file representing an animation with m number of key-frames for a skeleton that has n number of joints. This was our first attempt to animate a model by using the mocap data. We made a literature survey to find a means to convert 3D joint or marker positions to bone transformations. Papers we found about motion capture data did not give any details about this procedure. They just mentioned that they somehow converted the raw mocap data to bone transformations for their own purposes and avoided to explain the process about bone transformations. Assuming that it is a straightforward procedure, we tried to come up with our own method. We find a simple solution

which is inspired by the algorithm used for calculating skeleton transformations in the SSD and DQS methods. We will give details about this algorithm in Section 5.5.1. We will give the results of our algorithm and explain why it fails. Since we weren't able to achieve a successful bone transformation extraction by using our algorithm, we had to use commercial software to do so. We will give how we finally extract the bone transformations in Section 5.5.3

5.6.1 Calculating Bone Transformations

In Section 4.2, we explained how bone transformations are combined and propagated to the child bones. As a result of these combined bone transformations, joint positions could be calculated. Calculating bone transformations is an inverse problem where bone transformations are calculated from given joint positions. We will explain the algorithm using a 2D example. In Figure 44-a, all joint positions are given. Our goal is to calculate the rotation and translation that each bone should go through to reach these joint positions. We will assume that all bones are initially aligned along the x-axis (reference axis). In Figure 44-a, we calculate the angle between x-axis and the *jointI* by the formula $\alpha = \arctan(\text{jointI.y}/\text{jointI.x})$. We found that $\alpha = -90^\circ$. Since we want to transform *jointI* and other joints to their local coordinates, we will transform all of the joints by using the inverse rotation $\alpha = 90^\circ$. After this transformation is applied to all joint positions, new joint positions are shown in Figure 44-b. Since *jointI* is aligned with x-axis. We can translate the *jointI* to origin by using the $v_T^{-1} = (-\| \text{jointI} \|, 0)$ which is the inverse of its translation vector $v_T = (\text{jointI}, 0)$. We register the transformation $v_T = (\| \text{jointI} \|, 0)$ and rotation $\alpha = -90^\circ$ to *jointI*. Note that in Figure 44-c, applying inverse transformations of *jointI* to all other joints results in transforming of *jointII* to its local coordinates. Same procedure will be applied for *jointII* and *jointIII*. The point here is to transform every bone to its local coordinates where its local transformations can be calculated.

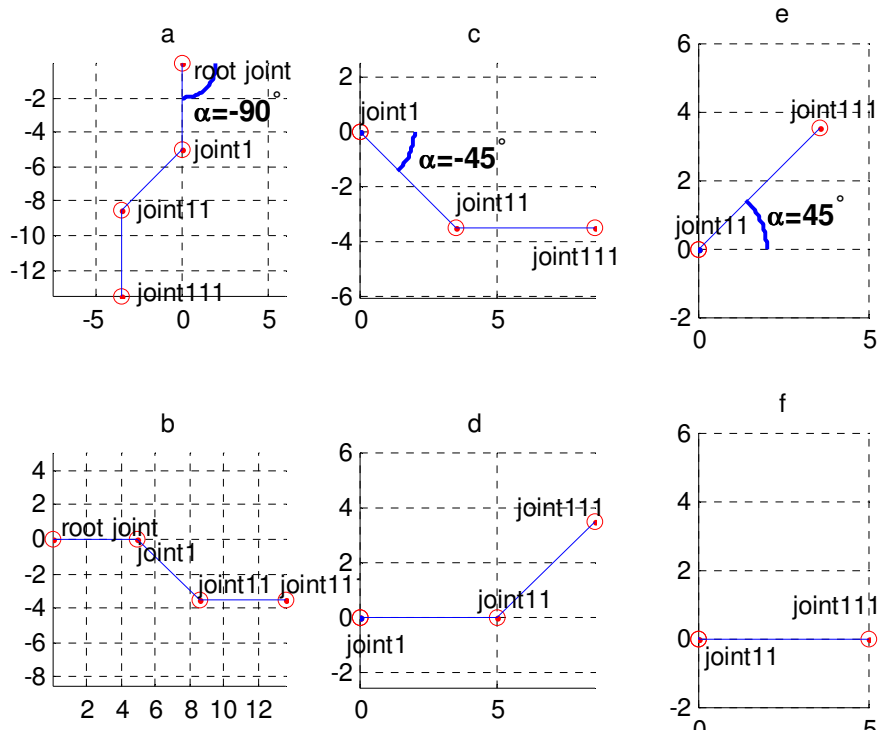


Figure 44 from given joint positions in (a) each bone is transformed back to its local frame, joint111 is transformed to its local coordinate in (e)

To explain how algorithm operates in 3D space, we will give a Matlab code segment. Related Matlab function to calculate the rotation quaternion between two vectors is given in Appendix B.4. Same function is integrated in our C++ code. The function `dualQuaternionBetweenTwoVectors` calculates the rotation DQ which rotates `vec1` so that it becomes `vec2`. Unit vectors are used as parameters. Since we use the reference axis as the x-axis, `vec1` is $(1,0,0)$ and `vec2` is the unit vector defined as $\text{bone_vector} / \|\text{bone_vector}\|$ where `bone_vector` defines the bone in its local coordinates. Calculating the translation is done just like in the given example. After the rotation is calculated by function `dualQuaternionBetweenTwoVectors`, its inverse is used so that the joint position is rotated onto x-axis. Afterwards translation vector v_T can be defined as $v_T = (\|\text{bone_vector}\|, 0, 0)$.

5.6.2 Outcomes of our Bone Transformation Calculation

The output of this algorithm produces exact results when animating a skeleton, but animating a 3D character model produces unsatisfactory results. Algorithm is affected very much from body orientation. There are other issues as well. Let's explain it given an example using shoulder bone defined between Clavicle and shoulder joints. Imagine that, a skeleton which is aligned along the y axis is rotated around the y-axis from the spine bones. This produces a result shown in Figure 45. Rotation is applied to spine joint transformed the skeleton (a) to pose (b).

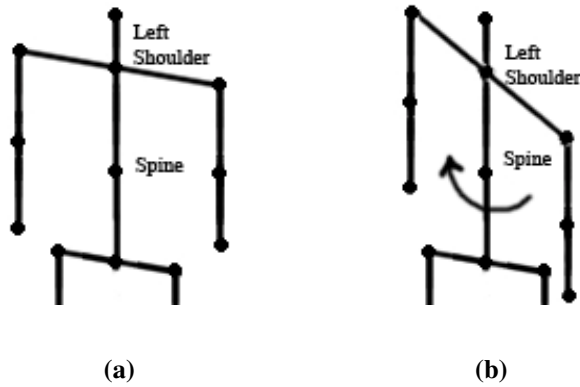


Figure 45 Rotation is applied to (a) spine joint translated the skeleton to pose (b)

Our algorithm can't resolve the fact that it is indeed the spine joint that rotates α degrees around the y-axis, instead the algorithm assumes that right shoulder rotates α degrees around the y-axis and left shoulder rotates $-\alpha$ degrees around the y-axis. Shoulder bone is almost a fixed bone which is a constraint for the skeleton. It seems that this problem could be solved by mapping joint positions to a realistic skeleton model and developing a constrained optimization algorithm. The deformation result is shown in Figure 46. Note the deformation at the shoulder regions, this is due to the reasons we explained.



Figure 46 Outcomes of our bone transformation calculation algorithm

5.6.3 Alternative Solution for Bone Transformation Calculation

To animate the medical mocap data, it was mandatory for us to find a way to extract the necessary bone transformations. Since this was a difficult problem, we decide to use professional software to extract the bone transformations. We decided to use MotionBuilder for this purpose. We converted the raw marker position data files to FCAccess Analyzer Trace File (TRC) format which are formatted marker position files to import and process the data in MotionBuilder. TRC ('.trc') file format is proposed by Ancot Corporation. After a number of steps, we finally managed to extract the bone transformations and export them as BVH ('.bvh') file.

Algorithms and methods used in MotionBuilder are indeed undocumented, so there is no way to tell exactly which approaches they are using to extract bone transformations. In their study, Zordan and Van Der Horst (2003) stated that they believe systems such as MotionBuilder use inverse kinematics (IK) based approaches. They claim that these approaches often leave indicative side effects, such as knees and elbows that never fully extend. They said “These systems are often unintuitive to control and lead to unexpected solutions due to ad hoc heuristics” (Zordan & Van Der Horst, 2003, p. 245). In their study, they attached marker positions to associated skeleton joints with virtual springs. By using a constrained optimization algorithm subject to the constraints such as internal torques and external forces, they calculated the joint-angles that carry the system to the equilibrium state.

5.7 Calculating Skin Offset Transformations

Another issue is the SOTs which is necessary for the SSD and DQS deformation. Before animating the bone transformations in BVH file, we had to assign bone weights to vertices and SOTs to bones. SOT is a transformation which transforms the bone and the related model vertices to the bones local coordinates. In a BVH, a line beginning with ‘OFFSET’ contains the offset value of that bone.

```
JOINT Neck
{
  OFFSET      0.00  18.65  0.00
  CHANNELS 3 Zrotation Xrotation Yrotation
```

In the example, the offset of Neck is (0,18.65,0). Offset is the displacement of the end joint of the bone from the end joint of its parent bone in the dress pose. It can be used as a reference axis vector. In Section 5.5.1, we highlighted that we used x-axis as the reference axis. At each key frame, transformations should be calculated relative to the x-axis. It is the same for the offset vector of a bone. By using the offset of each bone, dress pose of the skeleton can be defined.

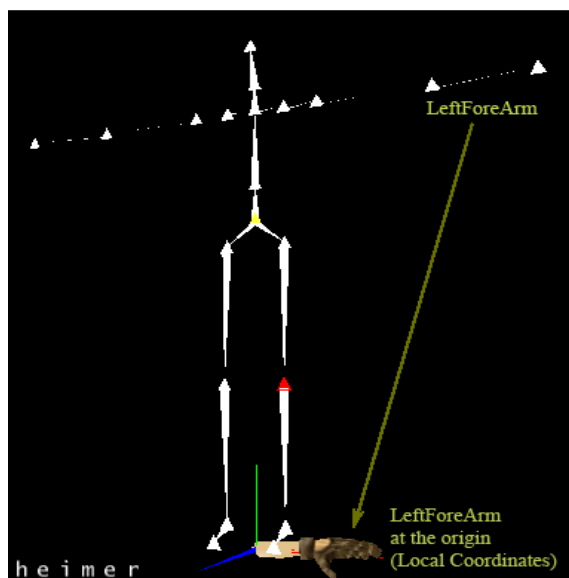


Figure 47 The affect of SOT

This pose is important since weight assignment and SOT calculation depend on dress pose. The algorithm in Section 5.5.1 can be used to calculate the SOT of the bone. Note that, offset vector of the bone shall be used instead of x-axis to extract SOTs. Figure 47 indicates affect of SOT of the bone on the vertices which are influenced by that bone. Clearly, the vertices influenced by the left fore arm are transformed to the origin when the SOT of the LeftForeArm bone is applied.

CHAPTER 6

RESULTS, CONCLUSIONS AND FUTURE DIRECTIONS

6.1 Outcomes of the Animation Tool

Our animation tool produces only visual outputs. One of our main goals is to produce a gait animation with movement patterns as close to the video recordings as possible. In this section, we will try to compare the outcomes of our animation tool with real video display. We will also show the limitations of our results and explain the reasons. Before discussing the final outputs of the tool, we shall discuss intermediate outputs of each step.

At the data acquisition step, we acquired medical gait data from two separate sources. At first, we acquired data from the mocap lab of Koç University. We captured data of four different gait disorders. Three of these were performed by us. These gait disorders were Alzheimer, Footdrop and Hemipad. We found one real patient who had multiple sclerosis (MS) in its early phase. We will explain how we acquired data from Koç University in Section 6.1.1.

Gait laboratory in Ankara University Hospital provided us more medical gait data. They use a professional system, Vicon, for extracting 3D marker positions. By using another tool, they convert these marker positions to rotations and store them in GCD file format. Instead of the GCD files we preferred C3D format supported by MotionBuilder®. We obtained four sets of C3D data three of which had accompanying gait video recordings.

The markers were only placed on lower part of the body, so in our animation upper part of the body was static. Besides, two of the patients used walking assist apparatus which can not be seen in our animations. One of the patients had polio (poliomyelitis), two of them had Cerebral palsy (CP) and last one had Osteoarthritis (OA). We give brief information about MS, polio, CP and OA in the Appendix A. Since marker positions are given in C3D files, we did not have to repeat the steps in Section 6.1.1. We explained how we converted 3D marker positions in C3D files to bone transformations in Section 6.1.2. After we obtained animations, we come up with an evaluation methodology with the guidance of Dr. Güneş Yavuzer. We explained our evaluation methodology in Section 6.2.

6.1.1 Motion Capture Process

We acquired motion capture data from a multi-camera optic mocap laboratory at Koç University in August 2007.. The lab was being used for another purpose which decreased the illumination and the markers had poor reflectance properties. As a result, we experienced some problems while extracting 3D marker positions and had to manually enter some of the 2D positions of markers on mocap images with the mouse. We attached 18 markers on our bodies during the motion capture. Manually tracking even a small portion of these markers for a short while was a troublesome task due to high frame rates.. A screenshot of the tracking tool is given in Figure 48.



Figure 48 A screenshot from the marker tracking tool developed in Koç University.

We used an eight camera optic mocap system. As it can be seen in second camera image, most of the legs are out of view. Body of the performer also acts as an obstacle between the camera and some markers which is a common problem of optical systems. Manually tracking markers was not only time consuming but the quality of the data was also inferior for the same reasons. We could not completely prevent the flicker of the markers. As it is obvious from Figure 48, the effective view of the cameras was small. Problems we stated forced us to capture very short motions with insufficient number of markers.

6.1.2 Acquiring Bone Transformation in MotionBuilder

We utilized mocap capabilities of MotionBuilder. We managed to learn it to a degree to convert marker positions to bone transformations. Motion builder has an actor object which marker points are mapped to. To do so, actor object must be shaped and oriented to be aligned with the markers as much as possible as shown in Figure 49. After this step markers are mapped to some particular body parts of the actor object. It is possible that better bone transformations could be achieved with more

experienced users of MotionBuilder. The data acquired from Koç and Ankara University did not have enough number of markers. With more markers, the output of MotionBuilder could be improved.

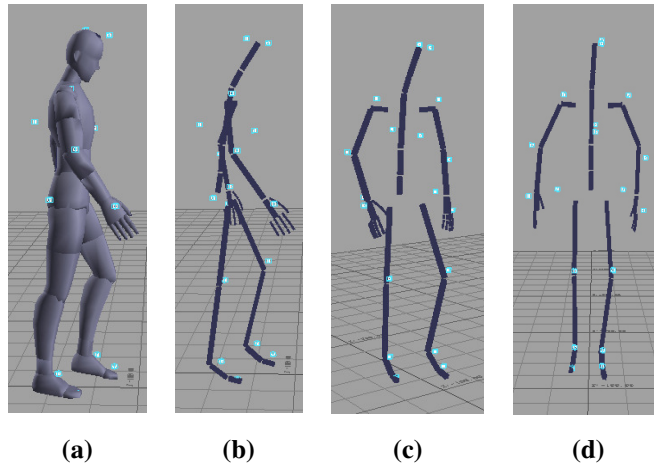


Figure 49 An actor is shaped and oriented to be aligned with the markers as much as possible. Displayed from different views (MotionBuilder)

There are some straightforward steps after mocap data is mapped to actor. Since markers were flickering, we had to smooth the data that was acquired from Koç University before exporting BVH file which contains bone transformations. Smoothing successfully prevented flickers but it also eliminated some fine details of the animation which was undesirable. Fortunately, mocap data acquired from Ankara University Hospital did not have such problems since Vicon systems were used in their laboratory.

6.1.3 Final Outcomes

We implemented a parser to read BVH files and loaded the 3D character mesh data from the DirectX file. Afterwards, skeleton structure must be bound to the character mesh model as explained in Section 5.5. As a result bone weights and SOTs are calculated. We observed that animations were satisfactory enough to observe obvious gait patterns. But more complex and detailed gait patterns could not be observed. We will give more detail about the performance of our tool in Section 6.2.2 where the validation methodology is explained. The limitations were mostly due to the problems such as the insufficient number of markers, noise and flicker in the mocap data which we explained in Sections 6.1.1 and 6.1.2. There are also some problems with our weight assignment. Professional software packages such as Maya and 3D Studio Max provide tools for an artist to manually fine tune the weights of the 3D character model. It was not easy to compete with the performance of such software packages. Our weight assignment algorithm mostly causes problems at the shoulder and hip regions. In Figure 50, defects originated from weight assignment are shown. The defect occurs because of the location of mesh vertices at the side of the chest. Normally, these vertices should be influenced only by spine bones. But our algorithm assigns more weights for the upper arm bone since these vertices are closer to the upper arm bone not to the spine bone.

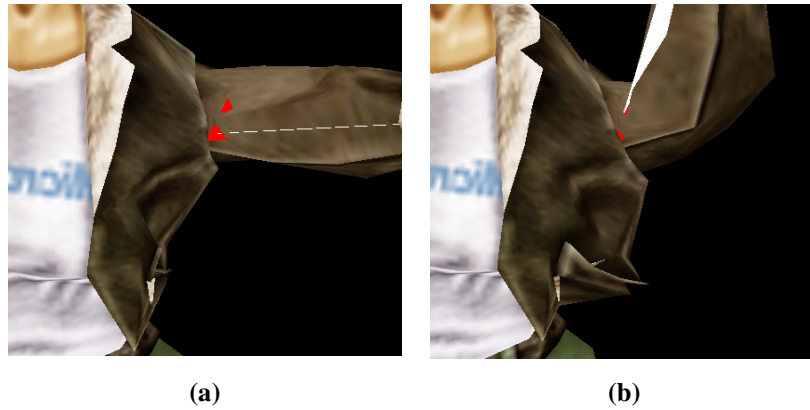


Figure 50 When rotated from (a) to (b), upper arm causes defects at the side of the chest because of the our weight assignment algorithm

Since legs are long cylindrical shapes, they are less affected by the imperfection of our weight assignment algorithm. Consequently, data acquired from Ankara University Hospital could be used successfully.

With the current capabilities of the animation tool, it would be possible to animate relatively more complex and detailed gait patterns if a better mocap data with more markers and less noise were provided. We give animation outcome of our Alzheimer mocap data in Figure 51

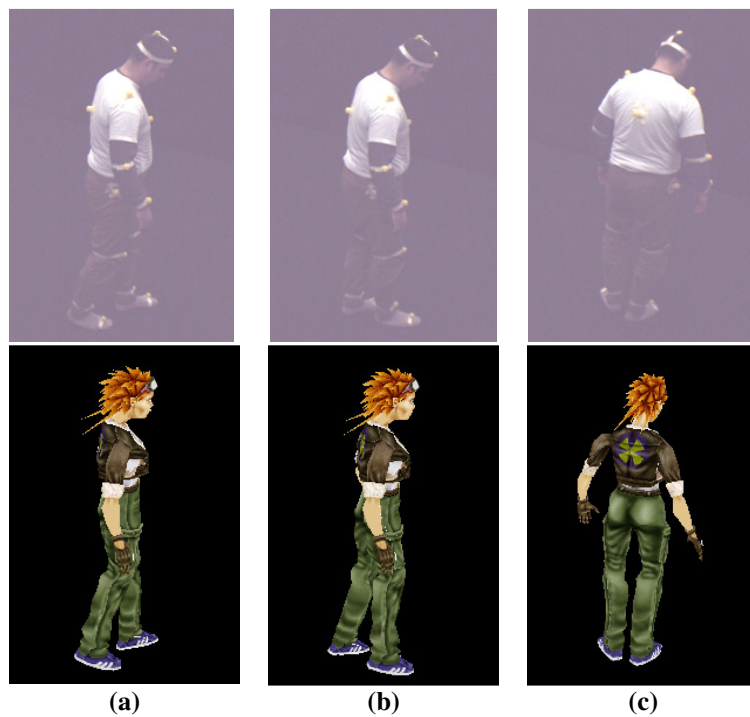


Figure 51 Outcome of the mug Alzheimer animation is shown for different moments

Animation outcome of MS mocap data is given in Figure 52



Figure 52 Outcome of the multiple sclerosis (MS) animation is shown for different moments

Our tool produced successful animation when appropriate bone weights, detailed bone structure and compatible proportions between mesh and skeleton were provided by the DirectX file. This proves that SSD and DQS algorithms are indeed adequate to animate detailed gait patterns.

Another problem is about the proportions of the skeletal structure exported from MotionBuilder. Joint transformations over the frames are calculated relative to the initial (dress pose) defined for the skeleton in MotionBuilder. For that reason, especially modifying the bone lengths to fit the skeleton inside the character mesh during the bind phase causes undesirable deformation results. Deformation results could be improved by matching the proportions of 3D mesh model with dress pose skeleton. Consequently, inconsistent proportions of mesh and skeleton leads to deformation defects while binding mesh to skeleton. As a result, many trials are needed to match proportion, since skeleton was modified in MotionBuilder not in our tool.

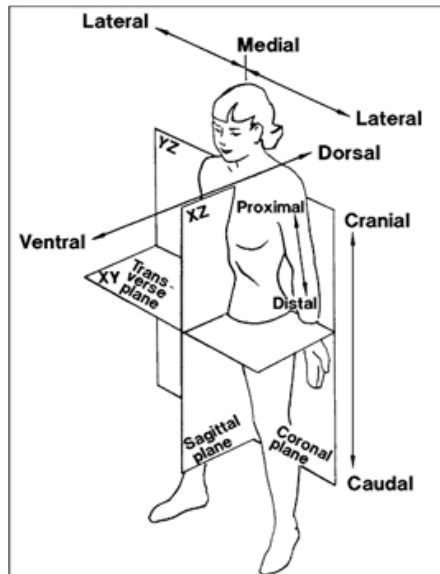
6.2 Validation of the Tool

In this section, we will have the developed tool evaluated by physicians. We will explain our validation methodology in Section 6.2.2. Before we pass to our validation methodology, a brief introduction to human gait is in order.

6.2.1 Brief Information about the Human Gait

In this section, we will give brief information about human gait such as anatomical planes of the human body, joints of lower body and phases of gait cycle. This information could be useful for a better understanding of our evaluation methodology.

There are three planes of human body. These are sagittal, coronal (frontal) and transverse (horizontal) planes. These panes are shown in Figure 53.



**Figure 53 Anatomical planes of the human body
(Courtesy of (Associates, 1978))**

Anatomical planes are important for gait analysts because they are used as references for joint rotations. The joints of lower the body are pelvis, hip, knee, ankle joints which are shown in Figure 54.a. The rotation axes of hip are shown in Figure 54.b.

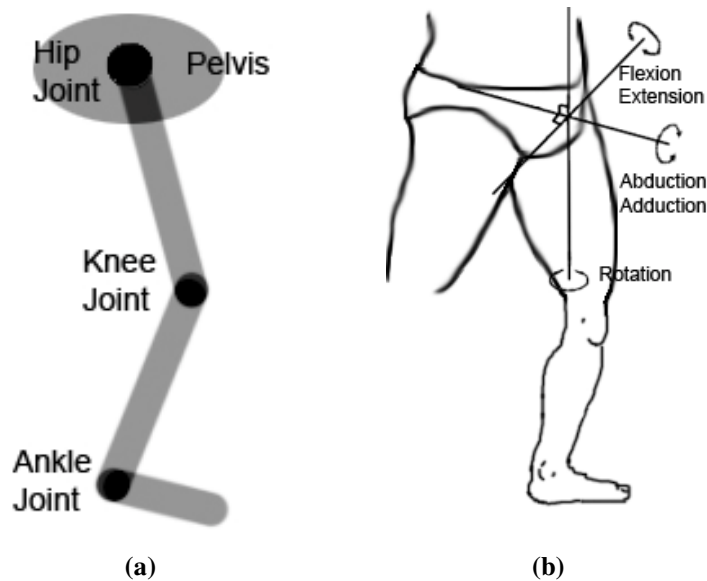


Figure 54 (a) Lower body joints and (b) hip rotation axes

Joints of the lower body are pelvis, hip, knee and ankle. Although rotations of joints have various names such as pelvic tilt and hip flexion, note that they represent a rotation on the sagittal plane. So there is no common name for a rotation on a particular plane, it can vary from joint to joint. Table 3 depicts the rotations of each joint.

Table 3 Descriptions of Joint Angles

Angle	Anatomical Plane
Pelvic Tilt	Sagittal
Pelvic Obliquity	Coronal
Pelvic Rotation	Transverse
Hip Flexion	Sagittal
Hip Abduction	Coronal
Hip Rotation	Transverse
Knee Flexion	Sagittal
Ankle Dorsiflexion – plantar flexion	Sagittal
Ankle Rotation	Transverse

Another important aspect of gait is its cycle. The cycle begins when one foot contacts the ground and ends when same foot contacts the ground again. The cycle has two main phases, stance and swing phase. During stance phase, the foot is always in contact with the ground. During swing phase the foot swings as the name implies and it has no contact with the ground. In Figure 55, phases of left and right leg show a full gait cycle. While right leg (darker one) is in swing phase, left leg (brighter one) is in stance phase. Stance phase is composed of loading response (LS) which is also known as initial contact, midstance (MSI), terminal stance (TSt) and preswing (PSw). Swing phase is composed of initial swing (ISw), Midswing (MSw) and terminal swing (TSw). A special case is called double support when one leg is in LS and other one is in PSw phase which means both feet contact the ground.

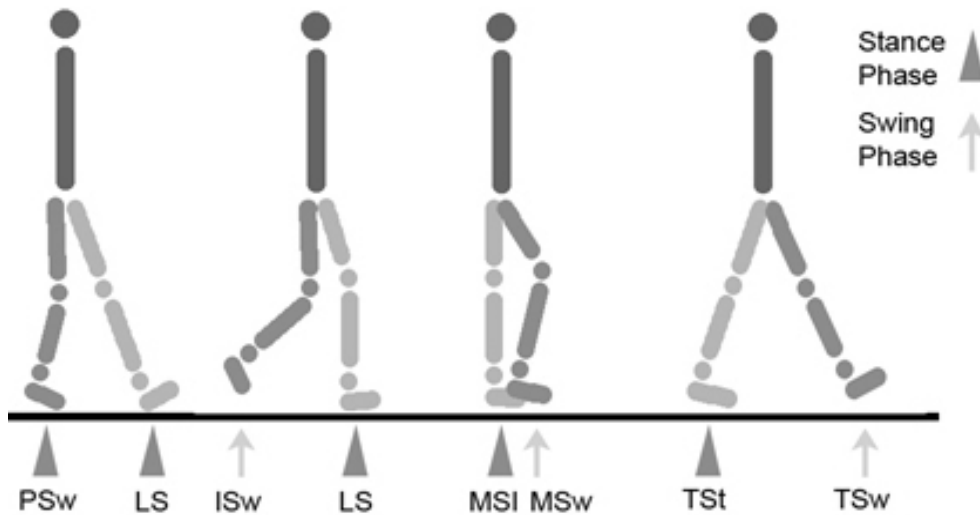


Figure 55 Human Gait Cycle

6.2.2 Validation Methodology

To validate our 3D animation tool, we applied three methods. We used three gait videos of the patients and animations of these videos. Each video was associated with an animation. There are some points about the animations that must be emphasized. Since we did not have any markers placed on the upper part of the body for these patients, upper part of our 3D model was static where only lower

part of the model was animating. Besides, walk assist apparatus used by patients could not be seen in the animations.

In our first method, we formed a Panel composed of three members. Each Panel Member (PM) has different attributes.

1. Panel Member 1 (PM1): First PM was a physician who was a gait analyst. She was aware of the facts about animations such as upper part of body being static and absence of walk assist apparatus.
2. Panel Member 2 (PM2): Second one was a physician who was not gait analyst.
3. Panel Member 3 (PM3): Third one was not a physician and not specialized in gait analysis, but she was aware the facts about animations.

We asked each PM to watch the videos and animations of patients. We arbitrarily named the videos and the animations as A, B, C and 1, 2, 3, respectively. After a PM watched a video, he/she watched all the animations in the order 1, 2 and 3. Then he/she matched the video with one of the three animations. He/she repeated the process for all three videos and was free to change his/her prior choices until the end. In Table 4, we gave the answers of PMs for each video. As it can be seen, first PM who was most qualified mismatched video A with animation 3 and video C with animation 1, but the correct matchings were between A-1, 2-B and 3-C. This indicates that the method needs to be further investigated.

Table 4 The answers of PMs for each video

Videos	PM1	PM2	PM3
A	3	1	1
B	2	2	2
C	1	3	3

The first panel member told that she was confused with the movements of the upper body which she tried to deliberately disregard. This indicates that the upper body should either be animated properly or should be omitted from the animations completely. As such the first method cannot be used as a validation method. So we will refer the second and third methods as the first and second validation methods.

In our first validation method, we asked another PM who was a physician and a gait analyst to watch the videos and animations. We will refer to him as Panel Member 4 (PM4). This time he was told about the correct matching between each animation and video. PM4 knew the facts about the animation just like PM1. He was asked to analyze, compare and rate the rotations of each joint on both coronal and sagittal planes. Each joint rotation was given points 1, 2 and 3 which stand for no similarity, partial similarity and exact correspondence respectively. The ratings for video A, B and C are given in Table 5, Table 6 and Table 7 respectively.

Table 5 Ratings of all joints on sagittal and coronal plane for video A

Video A	Sagittal	Coronal
Pelvis	1	3
Hip	1	1
Knee	2	1
Ankle	3	2

Table 6 Ratings of all joints on sagittal and coronal plane for video B

Video B	Sagittal	Coronal
Pelvis	3	3
Hip	3	3
Knee	3	3
Ankle	3	3

Table 7 Ratings of all joints on sagittal and coronal plane for video C

Video C	Sagittal	Coronal
Pelvis	3	3
Hip	2	3
Knee	3	3
Ankle	3	3

In the second evaluation methodology, PM4 filled an observational gait analysis checklist which was developed by the Professional Staff Association of Rancho Los Amigos Medical Center (1989). Gait analyst indicates the presence of a gait defect by a (+) and its absence by a blank area. Only the unshaded cells are relevant, since shaded regions represent phases of the gait cycle where the given deficit can not be seen. For example, there is no possibility for the forward lean deficit to occur in swing phase so associated cells are shaded. We listed the observational gait analysis checklists for all videos and animations. Videos A, B and C corresponds to animations 1, 2 and 3 respectively.

Department of Rehabilitation Sciences, University of Oklahoma Health Sciences Center
 Observational Gait Analysis Checklist

Video A

	STANCE				SWING		
	LR	MSI	TSt	PSw	ISw	MSw	TSw
Trunk							
forward lean					■	■	■
backward lean					■	■	■
lateral lean (R/L)	+	+	+	+	■	■	■
Pelvis							
no forward rotation (R/L)		■	■				
no contralateral drop (R/L)			■	■	■	■	■
hiking (R/L)	■	■	■				
Hip							
inadequate extension	■	+	+	+	■	■	■
circumduction/abduction	■	■	■				
Knee							
excessive flexion	+	+	■	+	+	+	■
uncontrolled extension			■	■	■	■	
inadequate flexion	■	■	■				■
Ankle/Foot							
foot slap		■	■	■	■	■	■
forefoot contact		■	■	■	■	■	■
foot flat contact	+	■	■	■	■	■	■
late heel off	■	■	■	+	■	■	■
contralateral vaulting	■	■	■	■			

Adapted from Professional Staff Association of Rancho Los Amigos Medical Center. (1989).
Observational gait analysis handbook. Downey, CA: Author.

Video B

	STANCE				SWING		
	LR	MSI	TSt	PSw	ISw	MSw	TSw
Trunk							
forward lean					■	■	■
backward lean					■	■	■
lateral lean (R/L)					■	■	■
Pelvis							
no forward rotation (R/L)		■	■				
no contralateral drop (R/L)			■	■	■	■	■
hiking (R/L)	■	■	■				
Hip							
inadequate extension	■	+	+	+	■	■	■
circumduction/abduction	■	■	■				
Knee							
excessive flexion			■				■
uncontrolled extension			■	■	■	■	
inadequate flexion	■	■	■	+	+	+	■
Ankle/Foot							
foot slap		■	■	■	■	■	■
forefoot contact		■	■	■	■	■	■
foot flat contact		■	■	■	■	■	■
late heel off	■	■	■	+	■	■	■
contralateral vaulting	■	■	■	■			

Video C

	STANCE				SWING		
	LR	MSI	TSt	PSw	ISw	MSw	TSw
Trunk							
forward lean							
backward lean							
lateral lean (R/L)							
Pelvis							
no forward rotation (R/L)							
no contralateral drop (R/L)							
hiking (R/L)							
Hip							
inadequate extension							
circumduction/abduction							
Knee							
excessive flexion	+	+		+	+	+	
uncontrolled extension							
inadequate flexion							
Ankle/Foot							
foot slap							
forefoot contact							
foot flat contact	+						
late heel off				+			
contralateral vaulting							

Animation 1

	STANCE				SWING		
	LR	MSI	TSt	PSw	ISw	MSw	TSw
Trunk							
forward lean							
backward lean	+	+	+	+			
lateral lean (R/L)							
Pelvis							
no forward rotation (R/L)							
no contralateral drop (R/L)							
hiking (R/L)							
Hip							
inadequate extension		+	+	+			
circumduction/abduction							
Knee							
excessive flexion L	+	+					
uncontrolled extension R	+	+					
inadequate flexion							
Ankle/Foot							
foot slap							
forefoot contact							
foot flat contact	+						
late heel off				+			
contralateral vaulting							

Animation 2

	STANCE				SWING		
	LR	MSI	TSt	PSw	ISw	MSw	TSw
Trunk							
forward lean							
backward lean	+	+	+				
lateral lean (R/L)							
Pelvis							
no forward rotation (R/L)							
no contralateral drop (R/L)							
hiking (R/L)							
Hip							
inadequate extension							
circumduction/abduction							
Knee							
excessive flexion - L	+	+		+	+	+	
uncontrolled extension							
inadequate flexion - R				+	+	+	
Ankle/Foot							
foot slap							
forefoot contact							
foot flat contact - L	+						
late heel off				+			
contralateral vaulting							

Animation 3

	STANCE				SWING		
	LR	MSI	TSt	PSw	ISw	MSw	TSw
Trunk							
forward lean					■	■	■
backward lean					■	■	■
lateral lean (R/L)					■	■	■
Pelvis							
no forward rotation (R/L)		■	■				
no contralateral drop (R/L)			■	■	■	■	■
hiking (R/L)	■	■	■				
Hip							
inadequate extension	■	+	+	+	■	■	■
circumduction/abduction	■	■	■				
Knee							
excessive flexion	+	+	■	+	+	+	■
uncontrolled extension			■	■	■	■	
inadequate flexion	■	■	■				■
Ankle/Foot							
foot slap		■	■	■	■	■	■
forefoot contact	+	■	■	■	■	■	■
foot flat contact		■	■	■	■	■	■
late heel off	■	■	■		■	■	■
contralateral vaulting	■	■	■	■			

6.3 Discussions

The results of the validation methods are very important since they represent the performance of the animation tool. When we inspect Table 5, Table 6 and Table 7 for first validation methodology, we can see that animations 2 and 3 are quite successful in representing the rotations of lower body joints but animation 1 has problems. We believe this due to the incorrect initial mapping of the actor of MotionBuilder to its markers. We will also call this mapping process binding of the actor with markers. As we mentioned before, this mapping process has great influence on the success of animation. Indeed first validation method only proves that model is sufficient in representing gait patterns in rotation wise, which means that videos and their associated animations show very similar rotational patterns.

At first glance, results of second validation methodology seem to contradict with the first one since observation checklists of the animations and their associated videos have some mismatches for all animations. This means that some of the deficits in a video can not be observed in its associated animation or an originally non-existing deficit can be observed in animation. There are several reasons for the mismatches. First of all, the deficits pertained to trunk cannot be represented correctly since there are no markers attached on the upper body. Previously, we told that we bound the actor object with the markers in MotionBuilder. The orientation of actor frames (bones) has great influence on the deficits in the checklists. Considering the foot bone at the loading response (LR) phase, the incorrect orientation of foot during the binding process can cause different deficits to occur. For example, foot flat contact seems to occur if the foot frame of the actor makes higher dorsiflexion during binding the actor foot with foot markers. On the contrary, forefoot contact seems to occur if the foot frame of the actor makes more plantar flexion than usual. These kinds of deficits are directly affected by the initial orientation of the actor object during actor-marker bind. On the other hand, bone rotations are relative movements. Even if the initial orientation of the bone is incorrect, it will make approximate degree of rotations during the animations, which usually produces similar rotation patterns.

6.4 Conclusion

In our thesis study, we developed a simple and handy 3D virtual character animation tool. The tool uses bone transformations to generate animation. We decided to use OpenGL for development because of its platform independency. It will be easier to adapt the tool to the platforms other than Windows when it is required. We parsed and animated DirectX and BVH files. There are other BVH animators around but these animators are only capable of animating stick skeleton models. Our tool provides a simple mechanism to integrate the BVH motion capture animation data with a 3D character mesh. It is designed to be a lightweight and free tool in order to be easily distributed, installed and used.

We employed Alias MotionBuilder® to convert 3D sensor positions to bone transformations which are inputs for the developed tool. The results show that the tool is effective in presenting the gait of the patients. The minor discrepancy between the animation and the video is due to inaccurate (manual) localization of the sensors on the 3D skeleton model.

Professional software packages such as MotionBuilder, Maya, 3D Studio Max can also animate 3D character model by using motion capture files. But, these are quite expensive commercial software packages. Since they have many capabilities, they are heavyweight products which have a steep learning curve. Our tool can be convenient for just showing medical animations because of its being compact and free of charge.

Our animation tool can also be distributed to medical institutions as well. There are many gait disorders identified by inspecting the movements and gait of patients. For that purpose, gait analysts use camera records to store the visual data of their patients. During medical education, they show their students video recordings to explain gait patterns of gait disorders. As motion capturing and 3D virtual character animation techniques evolve, it will become more effective using 3D animation technologies instead of video recordings. It can also be possible to display the animation to students from a web interface. In a medical video recording, there are lots of irrelevant and insignificant data such as the image of the room where the patient walk or the texture of the cloth which patient wear and so on. Besides, image based records are infamous for occupying too much disk storage. Extracting significant and reduced data have always been a research area. 3D representation of the gait data serves these purposes. Distributing, sharing and displaying video recordings have always been disadvantageous for that reasons. Movement data in 3D transformation representation is very small sized compared to the video recordings. Another advantage of using 3D technologies is the interactivity they provide. It is possible to inspect the movements of the patient from different angles and distances where a video recording is not available. Privacy of patients is yet another issue in medical domain. Some patients require his/her data not to be exposed. Since an anonymous virtual character is used to represent the movement of the patient, this ethical issue will be solved to a certain degree. Besides, bone transformations can also be used for gait analysis.

Most of the implementation is made from scratch. This makes further improvements possible. Using other binary libraries in the implementation can limit the possibilities to an extent that third party libraries allow. The animation tool is yet a prototype which needs further improvements. However, the outputs of our current implementation indicate the potential of the tool and of 3D technologies for medical purposes.

6.5 Future Directions

This tool is just a prototype; hence many improvements can be done. A better animation is the main concern for our study. As we stated in Section 6.1.3, quality of acquisition and processing of the mocap data is very important. Another important issue is the ease of use. The interactivity must be improved for that purpose. Though not essential, it is possible integrate new deformation algorithms or make enhancement to existing ones.

6.5.1 Improvements for Motion Capture Data

Acquisition of the motion capture data is out of our scope, but processing the mocap data is an issue that concerns us. An intelligent and capable joint transformation calculation algorithm shall be

integrated within the tool. Such a solution will decouple the whole animation process from other software packages such as MotionBuilder. In Section 6.1.3 we emphasized that it is hard to modify the proportions of skeletal structure to fit the mesh model. Such problem occurs because skeletal structure is created and modified in MotionBuilder where character mesh is bound to skeleton in our animation tool. If the joint transformation calculation capability is integrated, it will be possible to overcome this problem. Besides, such capability is necessary for the animation tool to become complete.

6.5.2 Improving Existing Algorithms and Integrating New Ones

A more flexible and intelligent weight calculation method is necessary to enhance both SSD and DQS deformation. Not only the algorithm itself should be improved but also it should be possible for the user to further modify the weights by using a user friendly interface. It is time consuming to align the skeleton inside a character mesh. A new algorithm can be implemented to automatically place and orient the skeleton inside the mesh.

6.5.3 Improving User Interface

Ease of use is very essential for any software. As many other prototypes, the interface of our animation tool is yet to be improved. We have implemented the basics of mouse interaction but this should be further improved. Interaction with the objects in the scene must become more comfortable. We can give the current manual bone transformation process as an example. We use keyboard buttons to rotate bones. In addition to rotations, scaling and translations must be driven by mouse inputs. Manual transformations of bones should be done both relative to global axis and local coordinate axis of the bone. Professional software packages such as MotionBuilder, Maya and 3D Studio Max use mouse interaction interfaces similar to Figure 56

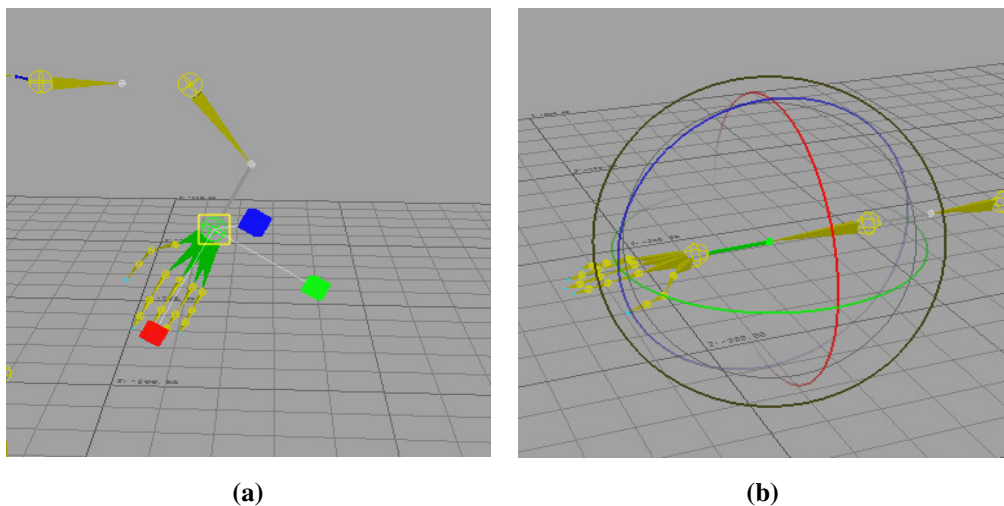


Figure 56 MotionBuilder skeleton scaling transformation by mouse is shown in (a) and rotation shown in (b)

Easily moving over the scene is another important feature, we provide it via use of keyboard where mouse control is a better alternative.

6.5.4 Platform Independency

One of the reasons why we chose OpenGL for development of our tool is that it is platform independent. Our tool is somewhat dependent on Windows platform, since we developed the tool by using Visual C++. One of our objectives was to use a platform independent Application Programming Interface (API) such as The OpenGL Utility Toolkit (OpenGLUT) in our project to decouple implementation from platform specific functions. For that purpose, we avoided using Microsoft Foundation Class (MFC) library in our implementation.

6.5.5 Performance Issues

Speed is not a major concern of our implementation but further optimization can be useful. As we explained in Section 5.3, at every time frame, bone transformations are interpolated and combined to calculate the local transformation of the bone. Then these local transformations are combined recursively to form the global transformations of bones. This is a time consuming process and is executed before each frame is rendered. Assuming that we have a definite and constant time interval, we can preprocess and calculate the global bone transformations for only once after animation data is read and initialized. At each frame transformations would be ready to be applied to vertices. This also makes it possible to use graphics adapter hardware. Most of the current graphics hardware has its own processor which is called Graphical Processor Unit (GPU). Precalculated global bone transformations could initially be sent to graphics hardware for once and by using fragment programming (GPU programming), DQS and SSD algorithms could be run on GPU. GPU is able to run such algorithms much faster than CPU does. This will also prevent the unnecessary CPU usage.

6.5.6 Web-Based Animation Tool

A more lightweight version of the animation tool can be implemented in Java to be used in web pages. This version will be only capable of animating a character mesh which is already bound to a skeleton. All necessary data will be available by streaming from a mocap and 3D character file server and no editing capabilities will be included. This version of the program is very suitable to consult a medical specialist about a patient's gait patterns. Doctors will also be able inspect the mocap data of their patients without having to go to the clinic or hospital. Such a tool can also be very useful for education of medical students because medical gait data would be made available to students. In addition, data would be delivered quickly to the client thanks to the compact size.

REFERENCES

- Alan, H. B. (1984). *Global and local deformations of solid primitives*. Paper presented at the Proceedings of the 11th annual conference on Computer graphics and interactive techniques.
- Alex, M., & Michael, G. (2003). *Building efficient, accurate character skins from examples*. Paper presented at the ACM SIGGRAPH 2003 Papers.
- Allen, J., Wyvil, B., & Witten, I. (1989). A method for direct manipulation of polygon meshes. *Proceedings of Computer Graphics International*, 451-469.
- Associates, W. (1978). *Anthropometric Source Book Volume I: Anthropometry for Designers* (NASA RP-1024). *Webb Associates, Yellow Springs, OH (NTIS No. N79-13711)*.
- Aubel, A., & Thalmann, D. (2000). Realistic Deformation of Human Body Shapes. *Computer Animation and Simulation 2000: Proceedings of the Eurographics Workshop in Interlaken, Switzerland, August 21-22, 2000*, 125-135.
- Aubel, A., & Thalmann, D. (2001). *Efficient Muscle Shape Deformation*. Paper presented at the Proceedings of the IFIP TC5/WG5.10 DEFORM'2000 Workshop and AVATARS'2000 Workshop on Deformable Avatars.
- Bartels, R., & Beatty, J. (1989). A technique for the direct manipulation of spline curves. *Graphics Interface*, 89, 33-39.
- Bruce, M., Patrick, M., & James, G. (2006). Animation space: A truly linear framework for character animation. *ACM Trans. Graph.*, 25(4), 1400-1423.
- Burtnyk, N., & Wein, M. (1971). Computer Generated Key Frame Animation. *Journal of the Society of Motion Picture and Television Engineers*, 80(3), 149-153.
- Burtnyk, N., & Wein, M. (1976). Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *Commun. ACM*, 19(10), 564-569.
- Cerebral palsy (n.d.) Retrieved April, 2008, from http://en.wikipedia.org/wiki/Cerebral_palsy
- Collins, G., & Hilton, A. (2001). Models for character animation. *Software Focus*, 2(2), 44-51.

- Coppens, P. Loading and displaying .X files without DirectX. Retrieved April, 2007, from <http://www.gamedev.net/reference/programming/features/xfilepc/>
- Digital Human Research Center, A. (n.d.). Shape transformation technique. Retrieved December, 2007, from <http://www.dh.aist.go.jp/research/centered/dressdummy/FFDexp.php.en>
- Doug, L. J., & Christopher, D. T. (2005). Skinning mesh animations. *ACM Trans. Graph.*, 24(3), 399-407.
- Dual Quaternion skinning in Maya (n.d.) Retrieved August, 2007, from <http://mayadqskinning.sourceforge.net/manual.txt>
- Eberly, D. H. (2006). *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*: Morgan Kaufmann.
- Ferdi, S., Richard, E. P., Wayne, E. C., & Stephen, F. M. (1997). *Anatomy-based modeling of the human musculature*. Paper presented at the Proceedings of the 24th annual conference on Computer graphics and interactive techniques.
- Forstmann, S., & Ohya, J. (2006). Fast skeletal animation by skinned arc-spline based deformation. *EG 2006 Short Papers*, 1--4.
- Gait abnormalities in minimally impaired multiple sclerosis patients (n.d.) Retrieved April, 2008, from <http://msj.sagepub.com/cgi/content/abstract/5/5/363>
- Gibson, S. F. F., & Mirtich, B. (1997). A Survey of Deformable Modeling in Computer Graphics November. Cambridge, MA, Mitsubishi Electric Research Lab
- Gjerde, E. (n.d.). Origami Tessellations. Retrieved December, 2007, from <http://www.origamitessellations.com/category/math/page/2/>
- Joseph, T., Eftychios, S., Geoffrey, I., & Ronald, F. (2005). *Robust quasistatic finite elements and flesh simulation*. Paper presented at the Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation.
- Kavan, L. (2007). *Real-time Skeletal Animation*. Czech Technical University, Prague.
- Ken, S. (1985). Animating rotation with quaternion curves. *SIGGRAPH Comput. Graph.*, 19(3), 245-254.
- Ladislav, K., Steven, C., Ji, ra, & Carol, O. S. (2007). *Skinning with dual quaternions*. Paper presented at the Proceedings of the 2007 symposium on Interactive 3D graphics and games.
- Ladislav, K., Steven, C., Zara, J., & Carol, O. S. (2007). *Skinning with dual quaternions*. Paper presented at the Proceedings of the 2007 symposium on Interactive 3D graphics and games.
- Ladislav, K., & Zara, J. (2005). *Spherical blend skinning: a real-time deformation of articulated models*. Paper presented at the Proceedings of the 2005 symposium on Interactive 3D graphics and games.

- Lewis, J. P., Matt, C., & Nickson, F. (2000). *Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation*. Paper presented at the Proceedings of the 27th annual conference on Computer graphics and interactive techniques.
- Magenat-Thalmann, N., Laperri, R., re, & Thalmann, D. (1988). *Joint-dependent local deformations for hand animation and object grasping*. Paper presented at the Proceedings on Graphics interface '88.
- Marc, A. (2002). Linear combination of transformations. *ACM Trans. Graph.*, 21(3), 380-387.
- Maths - Dual Quaternions (n.d.) Retrieved August, 2007, from <http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/other/dualQuaternion/>
- McKinley, M. (n.d.). Wire Tool. Retrieved December, 2007, from http://www.getatutorial.com/tutorials/Maya/Modeling/Wire-Tool-in-maya_1067.html
- Meredith, M., & Maddock, S. (2000). *Motion Capture File Formats Explained*. University of Sheffield: Technical Report.
- Michael, P., Patrick, C., Joe, L., & Karan, S. (2005). *Outside-in anatomy based character rigging*. Paper presented at the Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation.
- Multiple sclerosis (n.d.) Retrieved April, 2008, from http://en.wikipedia.org/wiki/Multiple_sclerosis
- Nadia Magnenat, T., Frederic, C., Hyewon, S., & George, P. (2004). *Modeling of Bodies and Clothes for Virtual Environments*. Paper presented at the Proceedings of the 2004 International Conference on Cyberworlds.
- Nedel, L., & Thalmann, D. (1998). *Modeling and Deformation of the Human Body using an Anatomically-Based Approach*. Paper presented at the Proceedings of the Computer Animation.
- Observational gait analysis (n.d.) Retrieved May, 2008, from <http://moon.ouhsc.edu/dthomps/gait/knematics/oga.htm>
- Osteoarthritis (n.d.) Retrieved April, 2008, from <http://en.wikipedia.org/wiki/Osteoarthritis>
- Parent, R. (2002). *Computer Animation: Algorithms and Techniques*: Morgan Kaufmann.
- Paul, G. K., Doug, L. J., & Dinesh, K. P. (2002). *EigenSkin: real time large deformation character skinning in hardware*. Paper presented at the Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation.
- Peter-Pike, J. S., Charles F. Rose, III, & Michael, F. C. (2001). *Shape by example*. Paper presented at the Proceedings of the 2001 symposium on Interactive 3D graphics.
- Platt, J., Terzopoulos, D., Fleischer, K., & Barr, A. (1987). Elastically Deformable Models. *Siggraph Proceedings*, 205-214.

- Poliomyelitis (n.d.) Retrieved April, 2008, from <http://en.wikipedia.org/wiki/Polio>
- Porcher-Nedel, L. *Anatomic modeling of human bodies using physically-based muscle simulation*. Ph. D. dissertation, Swiss Federal Institute of Technology, 1998.
- Provot, X. (1995). Deformation constraints in a mass-spring model to describe rigid cloth behavior. *Graphics Interface*, 95, 147--154.
- Richard, E. P. (1977). *A system for sculpting 3-D data*. Paper presented at the Proceedings of the 4th annual conference on Computer graphics and interactive techniques.
- Singh, K., & Fiume, E. (1998). Wires: a geometric deformation technique. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 405-414.
- Sven, F., Jun, O., Artus, K.-G., & Ryan, M. (2007). *Deformation styles for spline-based skeletal animation*. Paper presented at the Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation.
- Terzopoulos, D., & Waters, K. (1990). Physically-based facial modeling, analysis, and animation. *Journal of Visualization and Computer Animation*, 1(2), 73-80.
- Thomas, W. S., & Scott, R. P. (1986). Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph.*, 20(4), 151-160.
- Xiaohuan Corina, W., & Cary, P. (2002). *Multi-weight enveloping: least-squares approximation techniques for skin animation*. Paper presented at the Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation.
- Xiaosong, Y., Arun, S., & Jian, J. Z. (2006). Curve skeleton skinning for human and creature characters: Research Articles. *Comput. Animat. Virtual Worlds*, 17(3-4), 281-292.
- Zordan, V. B., & Van Der Horst, N. C. (2003). *Mapping optical motion capture data to skeletal motion using a physical model*. Paper presented at the Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation.

APPENDICES

APPENDIX A: DEFINITIONS OF GAIT DISORDERS

A.1 Multiple Sclerosis (MS)

It is also known as “disseminated sclerosis” or “encephalomyelitis disseminate”. Immune system attacks the central nervous system (CNS) which is an autoimmune case which leads to demyelination. It may result with numerous physical and mental symptoms, where patient often ends up with physical and cognitive disability ("Multiple sclerosis,").

Its potential negative effects on the gait pattern of the patient ("Gait abnormalities in minimally impaired multiple sclerosis patients,") are

1. Reduced Speed of Progression
 - Shorter Strides
 - Prolonged Double Support Phase
2. Muscular Function
 - Premature Recruitment of Gastrocnemius
 - Late Relaxation of Tibialis Anterior during Stance Phase

A.2 Polio

Polio stands for poliomyelitis or infantile paralysis. It is an acute viral infectious disease which can spread from person to person via the fecal-oral route, etc. In fewer than 1% of cases, the virus enters and affects the central nervous system, infecting and destroying motor neurons. This results with muscle weakness, reduced muscle tone or even muscle paralysis ("Poliomyelitis,").

Depending on the nerves involved, different types of paralysis may occur. Effects of the polio are diverse. It can cause stiffness and muscle weakness at the legs or arms. The individual may even lose the ability to use of one or both legs, arms ("Poliomyelitis,").

A.3 Cerebral palsy

Palsy is a medical term derived from the word paralysis. Cerebral palsy (CP) is a general term which comprises a group of non-progressive and non-contagious conditions that cause physical disability in human development. Damage to the young developing brain at the motor control centers

is the cause of CP. It mostly occurs during pregnancy, seldom during childbirth or sometimes after birth up to about age three ("Cerebral palsy,").

All CP types are characterized by abnormal motor development and coordination, reflexes, muscle tone or posture such as slouching. Joint and bone deformities and contractures can also occur. Spasticity, spasms, other involuntary movements are the classical symptoms. Unsteady gait, problems with balance, and/or soft tissue findings such as decreased muscle mass can be the results. Toe walking and scissor walking where the knees come in and cross are the most common indicators for patients with CP. Similar to polio, symptoms are very diverse ("Cerebral palsy,").

A.4 Osteoarthritis

Osteoarthritis (OA) also known as degenerative arthritis, is a degenerative joint disease. Destruction or decrease of synovial fluid and eventually erosion of the cartilage tissue at the joints occurs as a result of low-grade inflammation. This causes decreased movement because of the pain. Besides, regional muscle loss may occur, and ligaments may become more lax.

Chronic pain, causing loss of mobility and often stiffness are the main symptoms. Although there are other symptoms, the prominent symptom associated with abnormal gait is the limited joint motion ("Osteoarthritis,").

APPENDIX B: PROGRAM CODES

B.1 OpenGL Mouse Selection Code with C++

```
void AbstractMouseHandler::Selection(unsigned short mouse_x, unsigned short mouse_y)
{
    // Create a selection buffer
    GLuint buffer[512];

    // The number of objects will be stored in hits
    GLint hits;

    // The Size Of The Viewport. [0] Is <x>, [1] Is <y>, [2] Is <length>, [3] Is
    <width>
    GLint viewport[4];

    // This Sets The Array <viewport> To The Size And
    // Location Of The Screen Relative To The Window
    glGetIntegerv(GL_VIEWPORT, viewport);
    // Tell OpenGL to use the buffer to fill selected object IDs
    glSelectBuffer(512, buffer);

    // Puts OpenGL In Selection Mode. Nothing Will Be Drawn to
    // screen. Object ID's and Extents Are Stored In The Buffer.
    (void) glRenderMode(GL_SELECT);

    // Initializes The Name Stack
    glInitNames();
    // Push 0 (At Least One Entry) Onto The Stack
    glPushName(0);

    // Switch to Projection Matrix Mode
    glMatrixMode(GL_PROJECTION);

    // Push The Projection Matrix so restore it later
    glPushMatrix();

    // Resets The Matrix to Identity
    glLoadIdentity();

    // This Creates A Matrix That Will Zoom
    // Up To A Small Portion Of The Screen, Where The Mouse Is.
    gluPickMatrix((GLdouble) mouse_x, (GLdouble) (viewport[3]-mouse_y), 1.0f,
    1.0f, viewport);

    // Apply The Perspective Matrix
    gluPerspective(45.0f, (GLfloat) (viewport[2]-viewport[0])/(GLfloat)
    (viewport[3]-viewport[1]), 0.1f, 10000.0f);

    // switch to the Modelview Matrix
    glMatrixMode(GL_MODELVIEW);

    // Resets The Matrix to Identity
    glLoadIdentity();

    // camera class sets the perspective matrix
    // due to current camera orientation
    Cam->SetPrespective();
    glPushMatrix();

    // render objects under selection mode
    SelectionDraw();

    // Select The Projection Matrix
    glMatrixMode(GL_PROJECTION);
    // Pop The Projection Matrix
```

```

    glPopMatrix();
    // Select The Modelview Matrix
    glMatrixMode(GL_MODELVIEW);
    // Switch To Render Mode, Find Out How Many objects are under // mouse
    hits=glRenderMode(GL_RENDER);

    // at last OpenGL api filled the buffer, so process the
    // results at will
    processHits(hits, buffer);
}

```

B.2 Mouse Drag Code with C++

```

// "linePoint": is the projection of mouse pointer on near plane and will be updated
// due to screen coordinates of mouse and current perspective projection matrix
void retrieveProjectionOfMousePointerOnNearPlane(math3d::Vector *linePoint, GLint
mouse_x, GLint mouse_y)
{
    GLint viewport[4];
    glGetIntegerv(GL_VIEWPORT, viewport);

    GLfloat half_height = 1.0f * tan( 45.0f * 0.5f * TWOPI_OVER_360 );
    GLfloat aspect = (GLfloat) (viewport[2]-viewport[0])/(GLfloat) (viewport[3]-
viewport[1]);
    GLfloat half_width = half_height * aspect;

    GLfloat eye_screen_ratio_x = (2 * half_width) / ((GLfloat) (viewport[2]-
viewport[0]));
    GLfloat eye_screen_ratio_y = (2 * half_height) / ((GLfloat) (viewport[3]-
viewport[1]));

    linePoint->x = -half_width + ((GLfloat)mouse_x) * eye_screen_ratio_x;
    linePoint->y = +half_height - ((GLfloat)mouse_y) * eye_screen_ratio_y;
    linePoint->z = -1.0f;
}

// line function paramters should be adjusted due to current model view matrix
void retrieveLineFuntionDataFromInverseOfViewMatrix(math3d::Vector *linePoint,
math3d::Vector *lineVector, GLfloat *m)
{
    GLfloat xp2 = linePoint->x*m[0] + linePoint->y*m[4] + linePoint->z*m[8] + m[12];
    GLfloat yp2 = linePoint->x*m[1] + linePoint->y*m[5] + linePoint->z*m[9] + m[13];
    GLfloat zp2 = linePoint->x*m[2] + linePoint->y*m[6] + linePoint->z*m[10] + m[14];

    lineVector->x = xp2 - m[12];
    lineVector->y = yp2 - m[13];
    lineVector->z = zp2 - m[14];

    linePoint->x = xp2;
    linePoint->y = yp2;
    linePoint->z = zp2;
}

```

B.3 Calculation of Line Plane Intersection with C++

```

// "linePoint": is a point on the mouse ray, "lineVector": is the direction of
// mouse ray, "planePoint": is a point on the plane of mouse ray. "planeNormal":
// is the normal of the plane
// returns: "glPoint": funtion returns the intersection point of the mouse ray
// and plane defined
math3d::Vector* retrieveLinePlaneIntersection(math3d::Vector *linePoint,
math3d::Vector *lineVector, math3d::Vector *planePoint, math3d::Vector *planeNormal)
{
    math3d::Vector *intersectionPoint = new math3d::Vector();

    GLfloat t =
        ((planePoint->x*planeNormal->x + planePoint->y*planeNormal->y + planePoint-
>z*planeNormal->z) - (linePoint->x*planeNormal->x + linePoint->y*planeNormal->y +
linePoint->z*planeNormal->z)) / (lineVector->x*planeNormal->x + lineVector -
y*planeNormal->y + lineVector->z*planeNormal->z);

    // intersection of plane and line is
    intersectionPoint->x = lineVector->x*t + linePoint->x;

```



```
intersectionPoint->y = lineVector->y*t + linePoint->y;  
intersectionPoint->z = lineVector->z*t + linePoint->z;  
  
return intersectionPoint;  
}
```

B.4 Calculation of Rotation Quaternion between Two Vectors with Matlab Script

```
function [dq] = dualQuaternionBetweenTwoVectors(vec1, vec2)  
    rotationAxisVec = cross(vec1, vec2);  
    rotationAxisVec = rotationAxisVec / norm(rotationAxisVec);  
    tetha = acos( dot(vec1, vec2) / (norm(vec1) * norm(vec2)) );  
    dq = [cos(tetha/2) sin(tetha/2)*rotationAxisVec 0 0 0 0];  
end
```