Motion Capture Adaptation for Animating Characters with Varying Topology

by Iu Ka Chun Bartholomew

A Thesis submitted in Partial Fulfillment of the Requirements for the Degree of Master of Philosophy in Department of Computing

The Hong Kong Polytechnic University

March 2005

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

Iu Ka Chun Bartholomew

Abstract

Research on motion retargeting and synthesis for character animation has been mostly focused on the character scale variations. In our recent work we have addressed the motion retargeting problem for characters with different topologies. The purpose of this research project has been to devise a new method for retargeting captured motion data to an enhanced character skeleton having a topology that is different from that of the original captured motion. The new topology could include altered hierarchical structures, altered number of segments and scaled segments.

In this thesis, we propose a framework based on the concept of a *motion control net* (MCN), an external structure analogous to the convex hull of a set of control points defining a parametric curve or a surface patch. Retargeting is achieved as a generalized inverse kinematics problem using an external MCN that encapsulates the motion characteristics of the character embedded in the original motion data and decouples the strong dependency of the motion on the topology of the original motion skeleton. Convergence of the retargeting results requires dynamic modification of the MCN structure. This also allows to interactively edit the MCN and modify the conditions for the motion analysis. The new method can automatically synthesize new segment information and, by combining the segment motion in the MCN domain with a suitable displacement of control points embedded in the original motion capture sensor data, the method can also generate realistic new motions that resemble the motion patterns in the original data.

Acknowledgements

I would like to thank my supervisor Dr. George BACIU for providing valuable advice, kindly encouragement as well as technical support.

I would also like to thank all my friends from The Hong Kong Polytechnic University for their moral support.

Finally, I would like to thank my parents, brother, sister and all relatives for their support, patience and encouragement when I was in depression.

Contents

1 Introduction **1**

1.1 Problem 3

1.2 Motivation 3

1.3 Contribution 5

1.4 Organization 5

2 Background and Related Work **6**

2.1 Overview 6

2.2 Character Skeleton Definition 8

2.3 Inverse Kinematics 9

2.4 Inverse Rate Control 11

2.5 Jacobian Transpose Method 13

2.6 Cyclic-Coordinate Descent Method 15

2.7 Spacetime Method 17

2.8 Pragmatic Spacetime Method 19

2.9 Intermediate Skeleton 20

3 Motion Control Net Theory and Implementation **23**

3.1 Motion Control Net Construction 24

3.1.1 Motion Control Net Connection 25

3.1.2 Control Net Angles 27

3.1.2.1 Hierarchical Data 27

3.1.2.2 Segment-to-Segment Movement 30

3.1.2.3 Orientation Data 31
3.2 Motion Control Net Manipulation 32
3.3 Inverse Control Net Overview 33
3.4 The Inverse Control Net - Method 1 34
3.4.1 Segment-to-Segment Movement Recovering 36
3.4.2 Hierarchical Data Recovering 37
3.4.3 Orientation Data Recovering 39
3.4.4 A Topological Retargeting Example 40
3.5 The Inverse Control Net - Method 2 40
3.5.1 End Vector Prediction 45
3.5.2 Restoration of Segment-to-Segment Movement 47
3.5.3 Restoration of Hierarchy Data 48
3.5.4 Restoration of the End Node Position 52
3.5.5 Restoration of the Orientation Data 52
3.5.6 End Node Adjustment 53
3.5.7 A Topological Retargeting Example 53
3.5.8 Sensitivity Problem on the Hierarchical Data 54
3.5.9 Segment-to-Segment Movement Sign Changing Problem . . 57
3.6 The Inverse Control Net - Method 3 58
3.6.1 Node Position Restoration 59
3.6.1.1 Step 1 60
3.6.1.2 Step 2 63
3.6.1.3 Step 3 63
3.7 Singularity issue 64
3.8 Data Format 64
3.8.1 Internal Data Format. 65
3.8.2 Script File Format. 66
3.9 Data Flow 67
3.10 Interface Design 68

4 Results **71**

4.1 Retargeting of a Sword-Swinging Sequence 72

4.2 Retargeting of Dancing Sequence A 75

4.3 Retargeting of Dancing Sequence B 75

4.4 Retargeting Human Motion to a Spider Skeleton 76

4.5 Manipulation details 77

4.6 Performance Comparison 79

5 Discussion **80**

5.1 Comparison 80

5.2 Future Extension 82

6 Conclusion **83**

Bibliography **85**

List of Figures

2.1 (a) The source skeleton. (b) The destination skeleton. (c) The dotted line is the new direction of the elbow. (d) The intermediate skeleton has the same number of nodes and using the same local coordinate frame as the source skeleton, while having the same orientation as the destination skeleton. 21

3.1 A MCN configuration. The nodes of the left skeleton are the source nodes and the nodes of the right one are the destination nodes. The CNA of the node $\mathbf{D}_3^d$ are generated by blending the CNA of nodes $\mathbf{D}_2^s$ and $\mathbf{D}_3^s$. 25

3.2 The hierarchical data and segment-to-segment movement of the node $\mathbf{D}_2^s$ are named α_2 and β_2 respectively. Only the principle axes x and y are shown in the figure. 27

3.3 (a) The arrows represent the coordinate frame $\mathbf{M}_1$. (b) The arrows represent the auxiliary frame $\mathbf{B}$. (c) Reference frames of different nodes. (d) The hierarchical data. 28

3.4 The segment-to-segment movement. 30

3.5 (a) $\mathbf{T}_i$ is the translated coordinate frame of $\mathbf{D}_{i-1}^s$. (b) The auxiliary frame $\mathbf{G}_i$. (c) The coordinate frame $\mathbf{M}_i$ of $\mathbf{D}_i^s$. (d) γ_i is the angle between the x axis of $\mathbf{G}_i$ and the x axis of $\mathbf{M}_i$. Only the principle x axis and y axis are shown in the figure. 31

3.6 (a): Before applying the ICN method 1. The upper skeleton is the original arm. (b): After the segment-to-segment movement recovery. (c): After six iterations of applying algorithm 1. (d): After the orientation data recovery, and the whole method 1 finishes. 41

3.7 First iteration of algorithm 1. 41

3.8 Second iteration of algorithm 1. 41

3.9 Third iteration of algorithm 1. 42

3.10 Fourth iteration of algorithm 1. 42

3.11 Fifth iteration of algorithm 1. 43

3.12 Sixth iteration of algorithm 1. 43

3.13 The steps for predicting the eighth end vector in the first prediction iteration. Arrows show the principle axes of: (a) the coordinate frame $\mathbf{M}_1$, (b) the auxiliary frame $\mathbf{A}_1$, (c) the auxiliary frame $\mathbf{A}_1$ post-multiplied by a rotational transformation which is formed by an x-unit vector and ϕ. Other lines are the first seven predicted end vectors. 46

3.14 The position of $\mathbf{D}_3^d$ lies on the dotted ellipse if the magnitude of β_2^g is kept constant. $\mathbf{a}_2$ is the rotation axis for restoring the hierarchical data of the node $\mathbf{D}_3^d$. 48

3.15 Two possible motions restored by the ICN method 2. The position of the end node of limb 1 and limb 2 lies along the direction of the target line. 53

3.16 The steps of a topological retargeting example. The six lines, which are shown from (a) to (d), represent the end vectors predicted in the first iteration of the prediction algorithm. The lines are counted clock-wise. The lines between p and q in (f) are the end vectors predicted in the second iteration of the prediction algorithm. The lines around r in (g) are the end vectors in the third iteration, although the separation of the lines is so small such that the lines are packed together. The line s in (k) is the final end vector. The source arm is only shown in (a) for reference. 55

3.17 Another view of nodes $\mathbf{D}_1^d$ and $\mathbf{D}_2^d$ in figure 3.14(without showing $\mathbf{D}_3^d$ and $\mathbf{D}_4^d$). The viewpoint is at the right hand side of figure 3.14 and is point at the target line (viewing the target line as a point, and the point is in the same position as $\mathbf{D}_1^d$.). The dotted ellipse is the possible position of $\mathbf{D}_3^d$, and the restored α_3^r is bounded to the range $[\theta_i, \theta_j]$. 55

3.18 Left: The sign of β_3^g is negative. Middle: The sign of β_3^r is positive after scaling some segments. Right: The sign of β_3^r is positive after modifying β_2^g and β_4^g. 56

3.19 The geometric property of α_3 and β_2. X, Y, Z are the principle axes of the coordinate frame $\mathbf{M}_1$. 60

3.20 The two solution pairs $\{r_1, s_1\}$ and $\{r_2, s_2\}$. X, Y, Z are the principle axes of the coordinate frame $\mathbf{M}_1$. 62

3.21 An example of two valid solutions. The right skeleton is selected as final solution. 62

3.22 Data flow of the topological retargeting system. 68

3.23 Interface of the topological retargeting system 69

4.1 α and β of the additional segment. α and β of the skeleton III to V are generated by displacing a curve on the original data. 73

4.2 Retargeting of a sword-swinging motion. The leftmost skeleton I is the source motion. The second skeleton II from left is a simple retargeted motion. The arm movement of the third and fourth motion is created by shifting the CNA by a curve. The arms of the rightmost skeleton V are synthesized by using two MCNs for each arm. 74
4.3 Source dancing motion. 75
4.4 Retargeted dancing motion having an additional segment in the lower leg. 75
4.5 Source motion of dancing sequence B. The stick figure is a pair of crab legs which is produced by keyframing. 76
4.6 The arms are synthesized by blending different limbs of the source motion. The movement of legs is created by combining the α of the original leg and the β of the crab leg. 76
4.7 The source motion, a human walking sequence. 77
4.8 The retargeted spider motion. Each leg motion is created by blending the motion of human arm and leg in the MCN domain. 77

List of Algorithms

1 The phase two of the ICN method 1 36

2 The algorithm used to find a solution in an interval. 38

3 ICN method 2 with a given end vector. 45

4 ICN method 2 without a given end vector. 45

5 ICN method 3 with a given end vector. 59

6 ICN method 3 without an end vector. 59

List of Tables

3.1 Three phases of the ICN method 1 35

3.2 An example of boundary values. 39

3.3 The given α_i^g, the given β_i^g and the selected intervals. 39

3.7 A simple script controls the construction of the MCN and the running of the ICN. 70

Chapter 1

Introduction

Motion capture is a practical and reliable technique for generating realistic motion for animated characters and has a broad range of usage in medical and biomechanical application, 3D simulation, feature films and entertainment. In contrast to the traditional animation technique known as *keyframing*, which involves manually creating a set of keyframes along the motion and then interpolating with intermediate frames, motion capture for computer character animation commonly involves recording human motion and then mapping it onto the motion of computer characters.

Among different motion capture technologies, the most popular and accurate motion capture system is an optical system. An optical system captures motion by using at least three cameras to record the trajectory of the markers which are attached to the performer's body. The motion data is then calculated from the trajectory by a motion capture software and recorded for immediate or delayed use. Motion being captured in this way is usually represented and stored in skeleton-based format.

The second part of motion capture for computer animation, which is the research focus of this thesis, is mapping the captured data onto the motion of virtual characters. The mapping is direct and straightforward if only playback of the motion of the original performer's skeleton is involved. In practical animation production,

however, the captured motion needs to be adapted to new character skeletons with different type of variations. Such process is also known as *motion retargeting*. Various retargeting methods [36, 12, 35, 21, 38] have been developed for retargeting motion to a destination skeleton having *dimension variations*, for example, scaled segments. The proposed methods above, however, so far did not address the problem for retargeting to a skeleton with *topological variations* having altered number of segments and altered hierarchical structures. An intuitive way to accomplish *topological retargeting* might be to generate the movement of the additional segments by directly cloning the information from its neighbor segments [28], but the additional segments and its neighbor segments would then move in a similar way, resulting in an unnatural motion.

In order to create realistic motion, topological retargeting requires a data representation which abstracts animation data from the skeleton hierarchy and describes limb motion regardless of the character skeleton topology. *Motion control net* (MCN), the new data representation we proposed to actualize the decoupling of the strong dependency of the motion characteristics from the topology of the original captured skeleton data, is an external structure analogous to the convex hull of a set of control points defining a parametric curve or a surface patch. MCN uses three sets of data to represent animation data. The first data set stores the hierarchical data, the second data set stores the segment-to-segment movement and the third data set stores the orientation data. These three sets of data together are named *control net angles* (CNA). In order to effectively represent motion of the whole body, different parts of the body are usually represented by different MCNs and a particular segment in the skeleton can belong to more than one MCN.

The concept of retargeting through MCN is a process of specifying the mapping between the segments in the captured character skeleton and the destination character skeleton, and then creating the corresponding MCN(s). In case the destination contains additional segments that are not found in the source character skeleton, the movement of the additional segments can be produced by duplicating or interpo-

lating the CNA from its neighbors or other sources in the MCN domain. Existing segments might need to be adjusted for matching its movement with the movement of the additional segments, and the adjustment can be done by displacing a curve on the CNA of the segments.

1.1 Problem

Problem in motion capture is that the recorded motion adheres to the particular skeleton of the performer employed in the motion capture sessions. When the recorded motion is applied to other different character skeletons directly, the main characteristics of the motion may be lost and artifacts may be introduced. Various retargeting methods [36, 12, 35, 21, 38] have already addressed the problem of retargeting with dimension variations; however, these methods did not address the problem for retargeting to skeleton with topological variations. The objective of this thesis is to tackle the retargeting in the presence of the topological variations.

1.2 Motivation

Motion capture system for computer character animation, having begun development since the late 1970's and being improved over time, is now a proven way to create realistic character motion. While motion capture system can virtually record the motion of any moving object, the focus of many researches that have been done before was on the precision of capturing human motion and how to reproduce the captured motion to a virtual humanoid character. Driven by the widespread adoption of motion capture system and the high demand of synthesizing non-humanoid characters commonly seen in feature films and 3D games, the needs for using motion capture techniques to create various types of virtual animal and non-humanoid characters are increasing. In most cases, however, since the costs of performing motion capture is still relatively high and usually the characters being animated are

imaginary creatures that simply do not exist in the real world, capturing the motion from an object with similar size, shape and features is impractical. As a result, captured human motion is being used to provide baseline reference in creating the character motion, and high level of manual participation in the creation process is still required.

In order to solve the above problem, various retargeting methods were devised to map the captured motion to animated character having certain level of variations. The most significant differences between a captured object and the animated character can be classified into two categories: dimensional and topological variations. We found that existing retargeting methods mostly address the problem of dimensional retargeting only. The lack of a general topological retargeting method motivated us to attempt devising such a method.

Our topological retargeting method differentiates from the traditional retargeting by considering the modification of the skeleton topology. The idea of our topological retargeting comes from the physical operation of a marionette. A marionette can be very easily controlled by a cross-bar from which strings are extended to the main limbs of the marionette. The same cross-bar not only can control a human-like figure, but can also control other non-humanoid puppets with high precision. Being so simple but highly adaptive, the cross-bar mechanism gives us an intuition of how to achieve general topological retargeting in our research. The controlling operations of a cross-bar can be roughly classified into three categories: one category adjusts the body shape and the limbs of the marionette, another category adjusts the reachable range of the limbs, and the last category adjusts the orientation of the limbs. These categorized controls act as the major starting points for formulating our retargeting method.

1.3 Contribution

Our research contributes to the field of motion retargeting by proposing a new topological retargeting method that performs motion retargeting in the presence of topological variations on character design. Four major contributions that have been made by our research are:

1. motion control net (MCN), the structure that encapsulates the skeleton topology,
2. control net angles (CNA), the data set that represents the animation details,
3. inverse control net (ICN), the method that converts the MCN and CNA back to the motion data, and
4. a generalized retargeting method using the MCN framework that covers both dimensional and topological retargeting.

1.4 Organization

The remainder of the thesis is organized as follows. In chapter 2 we review the background information and other relevant works of motion retargeting. In Chapter 3 we present our work including how to construct the MCN from a recorded motion, how to calculate the CNA, how to manipulate the MCN and CNA to perform topological retargeting, and how to preform the ICN. We also present the singularity issue, data format, data flow and the interface design of our topological retargeting system. In Chapter 4 we present some motion samples created by using our topological retargeting system and compare the performance of the ICN algorithms. In Chapter 5 we compare our method with other retargeting methods and outline the future extension. Chapter 6 offers our conclusion.

Chapter 2

Background and Related Work

2.1 Overview

Retargeting is a process that adapts a captured motion to a destination character skeleton with dimension variations having scaled segments. The earliest research bases on inverse kinematics (IK). Badler [1] used a minimal number of markers to capture motion, and applied inverse kinematics to adjust the position of the limbs. Subsequently, several variations of inverse kinematics were developed. Witkin and Popovic [42] described a motion warping technique for generating smooth transition between modified frames. Choi and Ko [6] presented an on-line retargeting system using inverse rate control. Shin et al. [36] demonstrated a real-time system for retargeting using a dynamic importance measure and a hybrid inverse kinematics solver. Lee and Shin [21] presented a motion editing system using a hierarchical curve fitting technique.

In order to generate realistic motion, physical constraints are added to the retargeting method, and a method with such constraints is spacetime method. This method utilizes constraints both in time and space and casts the retargeting problem as a constrained optimization problem. Being an optimization based method, spacetime method takes a user specified function as an objective function which usually measures the total energy consumption of the generated motion. Motion is

then synthesized by satisfying the constraints and minimizing the objective function. Originally, the spacetime constraints approach was introduced by [41]. Cohen [8] extended this approach into a more comprehensive system by adding a spacetime windows. Liu et al. [25] made a further improvement by adding constraints, objective manipulations, and a representation for the degrees-of-freedom. Gleicher [11] used a simplified spacetime technique to implement an interactive motion editing system. Gleicher [12] presented a pragmatic spacetime method.

In general, the methods mentioned above generate a motion similar to the original motion. In animation production, however, an animator may wish to produce new motion or create exaggerated motion based on existing motion. A number of methods then have been developed for this purpose. Litwinowicz [23] first demonstrated an animation system that creates motion using the signal processing technique. Bruderline and Williams [3] showed that the signal processing technique can be applied to motion manipulation. Gleicher [14] described a method to modify existing animation to follow a different path using constraint-based techniques. Liu and Popović[24] demonstrated a system that transforms simple animation of a complex dynamic character into realistic animation by applying laws of physics and the bio-mechanical formulations. Kovar et al.[18] proposed a method for synthesizing motion by using a graph structure.

There were some recent proposed methods focusing on retargeting motion to a skeleton with altered number of segments. Monzani [28] described a method using an intermediate skeleton and inverse kinematics for motion retargeting. The intermediate skeleton, is constructed using the same local coordinate frame as the source skeleton, has the same number of nodes as the source skeleton but has the same orientation as the destination skeleton. The intermediate skeleton is used to convert the motion from the source skeleton to the destination skeleton. Inverse kinematics is then applied to the destination skeleton to establish spatial constraints. Dontcheva [10] described a layered approach for creating and editing motion. The approach focuses on mapping the motion of a real-time input device, such as reflective props,

widgets, or a mouse, to the motion of the skeleton. Spatial mapping between the motion of the input device and the motion of the destination skeleton is calculated by the Canonical Correlation Analysis (CCA).

In the following sections, we first give the character skeleton definition and then present the details of different techniques for motion retargeting.

2.2 Character Skeleton Definition

A character skeleton is a hierarchy of segments. Each segment in the skeleton comprises of a pair of nodes, which are linked by a parent-child relationship. An exception to this relationship is the first node (root node) of the skeleton, which has no parent. In a typical skeleton configuration, the relationship starts with the lower torso as the root node, the upper torso as the child of the lower torso, and connected all the way to the palm being the last child in the skeleton. Similar relationships are also applied to link the foot and other limbs to the root node. Each node in the segment chain has an unique number to identify them individually. The numbering starts from the first node and continues to the end node, and a child node always has an identity larger than its parent. In a skeleton, some nodes may have multiple child nodes, creating a fan of segments.

Some retargeting methods focus on a particular segment chain of the skeleton without regarding other segments in the skeleton. The first node of the segment chain can be any node of the skeleton. The end node of the segment chain can be any direct child node or any descendant node of the first node. In motion retargeting, the end node is commonly named as the *end-effector* which appears first in the robotics study. Originally, an end-effector is the description of a device or tool connected to the end of a robot arm. When the end-effector is used to describe the end node of a segment chain in a retargeting method, it is the node which the user selects to working with.

2.3 Inverse Kinematics

In motion retargeting, the objective is to use the source motion data to drive the movement of a target skeleton. The target skeleton usually has segment lengths different from those of the source. As the segment lengths are changed, geometric constraints, for instance, the foot must be placed above the ground, may be violated. Under those cases, the target skeleton needs to be adjusted to re-establish the constraints; however, if the position of a node other than the root node is adjusted, the position of all the descendant nodes will also be affected. Therefore, tools that can position a node directly without affecting the descendant nodes are necessary for the skeleton manipulation. One of these tools is *inverse kinematics.*

Kinematics is the study of the position, velocity and acceleration of the end-effector without regarding the force driven the skeleton. Kinematics is divided into two fields: *forward kinematics* and *inverse kinematics*. Prior to the discussion of inverse kinematics, we first give a brief introduction to the basic idea of forward kinematics.

Forward kinematics is the study of how the changes in the joint-space-parameters of the segment chain affect on the position, velocity and acceleration of the end-effector. Each segment in the chain contains joint-space-parameters[9]. The joint-space-parameters forms a transformation $\mathbf{T}$ which is a 4 by 4 homogeneous matrix comprised of translation and rotation. The transformation $\mathbf{T}$ transforms the coordinate frame at the parent end (the node $i-1$) of a segment to the coordinate frame at the child end (the node i). $\mathbf{T}$ is written as:

$$\mathbf{T} \;=\; \mathrm{T}_{\hat{\mathbf{Y}}}\,(d)\,\mathrm{R}_{\hat{\mathbf{Z}}}\,(\theta)\,\mathrm{T}_{\hat{\mathbf{X}}}\,(a)\,\mathrm{R}_{\hat{\mathbf{Z}}}\,(\alpha) \tag{2.1}$$

where $\mathrm{T}_{\hat{\mathbf{Y}}}\,(d)$ creates a translational transformation along a y unit vector $\{0,1,0\}^T$ with length d. $\mathrm{T}_{\hat{\mathbf{X}}}\,(a)$ creates a translational transformation along a x unit vector $\{1,0,0\}^T$ with length a. $\mathrm{R}_{\hat{\mathbf{Z}}}\,(\theta)$ creates a rotational transformation about a z unit vector $\{0,0,1\}^T$ with θ degrees. $\mathrm{R}_{\hat{\mathbf{Z}}}\,(\alpha)$ creates a rotational transformation about

a z unit vector $\{0, 0, 1\}^T$ with α degrees. The representation of the segment chain that is formed by a list of $\mathbf{T}_i^{i-1}$s is known as the Denavit-Hartenberg representation [9].

In robotic study, any one of the joint-space-parameters can be a variable because the segment of a robot has more degree-of-freedom [9] than the segment of a living creature such as human. When the study of kinematics applies on a human skeleton which contains only rotational joint, only the θ is a variable; other three parameters d, a and α are fixed value that are used to describe the segment configuration.

In order to find the position and rotation of the segment n relative to the segment i, all the transformation in between these segments are concatenated together:

$$\mathbf{T}_n^i = \mathbf{T}_{i+1}^i \mathbf{T}_{i+2}^{i+1} ... \mathbf{T}_n^{n-1} \tag{2.2}$$

If, for example, we want to find the global position and orientation of the segment n, we set i to 0. The resulting global position and orientation are the concatenating of all the transformation from node 0 to n:

$$\mathbf{T}_n^0 = \mathbf{T}_1^0 \mathbf{T}_2^1 \mathbf{T}_3^2 ... \mathbf{T}_n^{n-1} \tag{2.3}$$

The relationship of the joint-space-parameters and the position and the orientation in the Cartesian space can be written in this form:

$$\mathbf{x} = \mathrm{f}\left(\mathbf{q}\right) \tag{2.4}$$

where $\mathbf{q}$ is a vector of joint-space-parameters, and $\mathbf{x}$ is a vector of the position and the orientation in Cartesian space.

Inverse kinematics is the study of how the position, velocity and acceleration of the end-effector affect the joint-space-parameters of the segment chain. Given the position and the orientation of the end-effector in Cartesian space, the joint-space-parameters are calculated by the inversion of equation 2.4. The form of inverse

kinematics is:

$$q = \mathrm{f}^{-1}(\mathbf{x}) \tag{2.5}$$

which is the basis of inverse kinematics.

The major problem of inverse kinematics, however, is hard to find the inverse of the function f in equation 2.4 due to the highly non-linear nature of the function f. In addition, a closed-form solution does exist only for some specific design robot arms [40]. Therefore, inverse kinematics may only be applicable to part of a general human skeleton by solving the inverse kinematics equation directly. In order to apply inverse kinematics on an arbitrary skeleton, the inverse kinematics must be solved by additional numerical methods or optimization techniques.

2.4 Inverse Rate Control

Inverse rate control is an iterative motion retargeting method. The problem of inverse kinematics is hard to find the inverse of the highly non-linear function f in equation 2.4. Instead of finding the function inverse, inverse rate control takes the derivative of the function f for performing motion retargeting. The derivative of the function f is known as the Jacobian [40, 9]. The Jacobian of the function f relates the rate of changes in the end-effector position and orientation $\dot{\mathbf{x}}$ to the rate of changes in joint-space-parameters $\dot{\mathbf{q}}$:

$$\dot{\mathbf{x}} = \mathrm{J}(\mathbf{q})\,\dot{\mathbf{q}} \tag{2.6}$$

where $\mathbf{q}$ describes the current status of the segment configuration. J is the Jacobian of the function f:

$$\mathrm{J} = \frac{\delta \mathrm{f}}{\delta \mathbf{q}} \tag{2.7}$$

J is a $m * n$ matrix, where m is the dimension of $\mathbf{x}$. m is 3 when the equation calculates only the position, or m is 6 when the equation calculates both the position and the orientation. n is the number of segments involved in the calculation. The

inverse of equation 2.6 provides the basis for the inverse rate control:

$$\dot{\mathbf{q}} = \mathrm{J}^{-1}(\mathbf{q})\,\dot{\mathbf{x}} \tag{2.8}$$

Inverting the Jacobian J is easier than inverting the function f because the non-linear complexity of the function f is reduced after taking the derivative.

A simplified inverse rate control method [6, 40], which is based on equation 2.8, performs motion retargeting without considering the orientation. The method adjusts the skeleton by advancing the end-effector toward a given position in each iteration within a small time step. In each iteration, $\dot{\mathbf{x}}$ is calculated as the difference between the current end-effector position and the given end-effector position. The corresponding rate of changes of the joint-space-parameters $\dot{\mathbf{q}}$ is then given by equation 2.8. $\dot{\mathbf{q}}$ is next integrated to $\mathbf{q}$ to find the new status of the skeleton configuration. Since $\mathbf{q}$ is changed in each iteration, $\mathrm{J}^{-1}(\mathbf{q})$ is valid only for a small time step, and $\mathrm{J}^{-1}(\mathbf{q})$ must be recomputed at each iteration. The procedure ends when the difference between the current end-effector position and the given end-effector position is smaller than a threshold.

Although inverting the Jacobian J is easier than inverting the function f, some problems exist when inverting the Jacobian. There are two causes of the problem and the corresponding solutions:

Non-Square Matrix

The Jacobian matrix has a dimension of $m * n$. m is the number of variables in Cartesian space, and n is the number of segments involved in the calculation . If n is larger than m, the segment configuration is redundant in degrees-of-freedom. It means that the segment chain contains more degrees-of-freedoms than required to position the end-effector. If n is smaller than m, the segment chain cannot be moved in some direction in Cartesian space. In any case that m is not equal to n, the resulting Jacobian matrix is a non-square matrix which has a problem in finding

the inverse as standard inverting methods work only for a square matrix. In order to invert a non-square matrix, the pseudoinverse [5] method can be used.

Singularity

The Jacobian matrix is singular if it is not in full rank, or some rows are linear dependent, or the determinant is zero. A singular matrix is not invertible. In the physical operation, a segment chain having a singular Jacobian matrix cannot be moved in some direction in Cartesian space no matter how the joint-space-parameters are changed. This problem can be solved by first detecting the singular condition using the Singular Value Decomposition (SVD) [33], then using the damped least-squares formulation [27] to solve the problem.

2.5 Jacobian Transpose Method

The methods described in the previous sections include a matrix inversion. There may be problems in finding the inverse of a matrix. In contrast, the Jacobian transpose method [40, 9] requires no matrix inversion. This method suggests a simplified dynamic model for motion retargeting of which the calculation only needs to find the transposition of the Jacobian matrix; therefore, this method has no matrix inversion problems.

Dynamic is the study of the relation between external force and internal torque. A force $\mathbf{F}$ is a set of pulling forces f along the principle axes of a coordinate frame:

$$\mathbf{F} = \{\mathrm{f}_x, \mathrm{f}_y, \mathrm{f}_z\}^{\mathrm{T}} \tag{2.9}$$

where $\{\}^{\mathrm{T}}$ denotes matrix transpose. In the simplified dynamic model, the end-effector is moved to a given position by a driving force which is calculated as the difference between the given end-effector position $\mathrm{x}_{\mathrm{d}}(t)$ and the current end-

effector position $\mathrm{x_c}\,(t)$:

$$\mathbf{F} \;=\; \mathrm{x_d}\,(t) - \mathrm{x_c}\,(t) \tag{2.10}$$

The external force $\mathbf{F}$ is related to the internal segment torque τ by the Jacobian matrix that is calculated in equation 2.7:

$$\tau \;=\; \mathrm{J^T}\mathbf{F} \tag{2.11}$$

The simplified dynamic model suggests that the internal joint torque τ can be simplified to be the joint-space-parameters acceleration $\ddot{\mathbf{q}}$, and the acceleration can be further simplified to the joint-space-parameter velocity $\dot{\mathbf{q}}$. Equation 2.11 is simplified to:

$$\dot{\mathbf{q}} \;=\; \mathrm{J^T}\mathbf{F} \tag{2.12}$$

Jacobian transpose method performs motion retargeting by advancing the end-effector toward a given position $\mathrm{x_d}\,(t)$ in each iteration within a small time step. In each iteration, the $\mathbf{F}$ is calculated as the difference between the current end-effector position and the given end-effector position. The corresponding joint-space-parameters velocity are then given by equation 2.12. $\dot{\mathbf{q}}$ is next integrated to $\mathbf{q}$ to find the new status of the skeleton configuration. The method ends when the difference between the current end-effector position and the given end-effector position is smaller than a threshold.

Although this method avoids matrix inversion, there is still a problem. As the matrix is inherently ill-conditioned, the Jacobian matrix is not completely immune to singularity. In each iteration of the Jacobian transpose method, when a segment chain approaches a fully extended configuration, the joint-space-parameters that is calculated from equation 2.12 oscillate around this fully extended configuration. Fortunately, this problem can be solved by using a smaller integration step-size, or by using an integration method with an adaptive step-size, or by clamping the joint-space-parameters velocity to a bounded value before integrating it into the

new status of the skeleton configuration.

2.6 Cyclic-Coordinate Descent Method

The Cyclic-Coordinate Descent (CCD) [39] method is an iterative heuristic search method, having an approach different from the previous three methods for performing motion retargeting. The most important advantage of this method is that it is completely immune to the matrix singularity and scales up to any number of segments. CCD starts motion retargeting from the end-effector up to the n^{th} parent segment in the segment chain. In each iteration, CCD focuses on only one segment and only one of the joint-space-parameters of the segment is allowed to change in order to minimize the position error and the orientation error of that segment. After adjusting the n^{th} parent segment of the end-effector, CCD goes back to adjust the end-effector if the current end-effector is not matched to the given position and orientation. CCD stops when the position error and the orientation error are smaller than a threshold. Since there is only one changing variable at each CCD iteration, a simple analytic solution can be formulated, and the execution speed of this solution is fast enough to be a real time algorithm.

The basic element of CCD is the error measure. The position error is defined as the square of the difference between the given position P_d and the current position P_c of the end-effector:

$$\mathrm{E}_p\left(q\right) = \left\|\mathrm{P}_d - \mathrm{P}_c\right\|^2 \tag{2.13}$$

Let the current orientation be $\mathrm{O}_c = \left\{\mathrm{u}_{1c}, \mathrm{u}_{2c}, \mathrm{u}_{3c}\right\}^{\mathrm{T}}$, and the given orientation be $\mathrm{O}_c = \left\{\mathrm{u}_{1d}, \mathrm{u}_{2d}, \mathrm{u}_{3d}\right\}^{\mathrm{T}}$. The orientation error is defined by:

$$\mathrm{E}_o\left(q\right) = \sum_{j=1}^{3}\left(\left(\mathrm{u}_{jd} \cdot \mathrm{u}_{jc}\right) - 1\right)^2 \tag{2.14}$$

In each iteration, CCD minimizes the error function:

$$\mathrm{E}\left(q\right) = \mathrm{E}_p\left(q\right) + \mathrm{E}_o\left(q\right) \tag{2.15}$$

A simplified CCD method [20] focuses on the position only. The method takes equation 2.13 as the error function. The detailed step of the method is the same as the original CCD method except for the error function. In each iteration, error of each node is minimized by rotating segment i about an axis a_i with a degree ϕ_i. a_i is calculated by the cross product of two vectors: one vector $\overrightarrow{P_{id}}$ is the position of the child end of segment i to the given position d, another vector $\overrightarrow{P_{ie}}$ is the position of the child end of segment i to the current end-effector position e. The equation of a_i is written as:

$$a_i = \overrightarrow{P_{id}} \times \overrightarrow{P_{ie}} \tag{2.16}$$

The rotation degree ϕ_i is found by the dot product of the vectors used in equation 2.16:

$$\phi_i = \overrightarrow{P_{id}} \cdot \overrightarrow{P_{ie}} \tag{2.17}$$

There is a behavioural difference in the interactive dragging operation when comparing CCD and other methods. The inverse rate control and the Jacobian transpose method adjust all segments in each iteration, and the manipulation of a segment chain is similar to playing a flexible elastic rod. In contrast, CCD prefers making changes to the distal segments, and the manipulation of a segment chain is similar to pulling on a loosely connected chain. The dragging behavior of CCD, however, can be changed by adding a damping factor d [20]. In each iteration, the rotation degree ϕ_i is clamped to the damping factor by:

$$\phi_i = \max\left(\phi_i,\, d\right) \tag{2.18}$$

After introducing the damping factor, the dragging behavior of CCD is the same as other methods.

2.7 Spacetime Method

Spacetime method is fundamentally different from the previously mentioned methods. Spacetime method focuses on the whole animation rather than an individual frame; therefore, the generated motion is guaranteed to be smooth. This method casts the retargeting problem as a constraint optimization problem. When an animator uses spacetime method to perform motion retargeting, the animator needs to specify an objective function to measure the pose, and a set of spacetime constraints to control the motion. The objective function is usually the total power consumption of all segments. The objective function and the spacetime constraints are then passed to an optimization algorithm which will find a pose that is best fitting the spacetime constraints and having minimum power consumption.

The moving particle presented in [41] is a simple example of spacetime method. The motion of the particle is governed by the Newton's law of motion:

$$\mathrm{f} = m\ddot{x} + mg \tag{2.19}$$

where $\ddot{x}$ is the acceleration. m is the mass and g is the gravity. In order to simplify the calculation of energy consumption, the function f is taken as energy consumption in a unit time. The objective function R is the total energy of the motion which is defined by the integration of function f over time:

$$R = \int_{t_0}^{t_n} |\mathrm{f}\,(t)|^2\, dt \tag{2.20}$$

In order to position the starting point in the x dimension to the position a and the ending point in the x dimension to the position b, two boundary spacetime constraints are added:

$$c_1 = x_0 - a = 0$$

and

$$c_2 = x_n - b = 0$$

Before solving the objective function R with an optimization algorithm, the $\ddot{x}$ in equation 2.19 has to be changed to a function $\mathrm{x}(t)$, depending on x only. A finite difference formulas is used to approximate the acceleration:

$$\ddot{x}_i = \frac{x_{i+1} - 2x_i + x_{i-1}}{h^2} \tag{2.21}$$

where h is the time interval between samples.

The above steps are used to generate motion of a single particle. For motion retargeting, a similar technique is used, but spacetime method does not work on the joint-space-parameters directly. The method works on the B-spline coefficient of a motion curve of which the minimization will not cause rapid oscillations. An example of using spacetime method to perform motion retargeting is given by [35]. In the example, kinematics constraint is enforced on the foot by:

$$r_k(t) = \|\mathrm{p}_a(t) - \mathrm{p}_d(t)\|^2 \tag{2.22}$$

where $\mathrm{p}_a(t)$ is the actual position of the foot and $\mathrm{p}_d(t)$ is the given position of the foot. The objective function R is defined by the integration of all K constrained frames across the constrained time interval from t_1 to t_2:

$$R = \int_{t_1}^{t_2} \sum_{k=1}^{K} r_k(t)\, dt \tag{2.23}$$

Although spacetime method outputs smooth motion, the method has two problems. The first problem is the slow execution speed. The optimization algorithm that is used in the spacetime method is usually a complex and non-linear algorithm which has a slow execution speed. In addition, spacetime method involves applying such algorithm to all frames of the motion; therefore, spacetime method may not able to provide a real time manipulation ability. Another problem is that the spacetime constraints are highly sensitive to the initial state. If the initial state is far away from the solution, the optimization algorithm may not be able to converge to

the solution.

2.8 Pragmatic Spacetime Method

While the constraints are easily specified for a moving particle, the constraints are much more difficult to be specified in retargeting a human motion if high level qualities of the motion must be preserved. Gleicher [12] presented in his work a more pragmatic approach for retargeting using the spacetime method. His approach has the following characteristics. It avoids encoding the high level qualities mathematically. It avoids using too many different types of constraint and objective in the system. His approach also tackles the problem without solving optimization problems which are costly to solve.

The pragmatic spacetime method adapts the motion from a character to another character having identical structure but different segment lengths while preserves desirable qualities of the source motion. The qualities are preserved by maintaining the identified features of the motion as constraints in the optimization process. The constraints can be violated if the character segment lengths are changed. The violated constraints are re-established by a spacetime constraints method, which works on the entire motion instead of individual frames.

The method consists of five steps:

1. Identify the constraints of the initial motion. Some of the constraints provided by Gleicher's system are: a parameter is in a range, a point follows a specific location, a point is in a region, a point is in the same position at different time, a point follows a path, two points are a specified distance apart and the vector between two points has a specified orientation.

2. Calculate an initial estimate motion $\mathrm{m}_1(t)$ which is comprised of a scaling of the skeleton and a translation to recenter the scaled skeleton. The translation is necessary for establishing the constraints such as a footprint constraint. This translation is then interpolated to frames that has no any displacements,

and the translation is filtered with a low-pass filtering to remove high frequencies.

3. Choose a representation for the displacement curve $\mathrm{d}\,(t)$ in order to constrain the curve not to have high frequencies. A cubic B-splines is chosen in the method. The control point of the curve can be placed with different spacing for representing different frequencies in the motion.

4. Calculate a displacement curve $\mathrm{d}\,(t)$ by solving the constraints. When $\mathrm{d}\,(t)$ is added to the $\mathrm{m}_1\,(t)$ calculated in step 2, the constraints should be satisfied.

5. (Optional) If the above step cannot satisfy the constraints sufficiently, use $(\mathrm{m}_1\,(t) + \mathrm{d}\,(t))$ as the initial estimate motion and a denser set of control points for the displacement curve to start a new iteration of calculation.

Since this pragmatic approach simplifies the original spacetime method, some artifacts appear in the resulting motion. In addition, unrealistic motion may appear because of the choosing of only geometric constraints in the system.

2.9 Intermediate Skeleton

Intermediate Skeleton [28] takes a different approach to perform motion retargeting. This method differentiates from other works by considering retargeting motion to a skeleton with different number of segments. The method consists of two steps. The first step is to construct an intermediate skeleton which has the same number of nodes and has the same local coordinate frame as the source skeleton, while having the same orientation as the destination skeleton. The second step is to copy the local value of the intermediate skeleton node to the destination skeleton node. IK is then used to adjust the nodes for satisfying the geometric constraints.

Figure 2.1 shows how to construct the intermediate skeleton. Figure 2.1 a shows the source skeleton, and 2.1 b shows the destination skeleton. Figure 2.1 c shows the initial intermediate skeleton, which has the same number of nodes and same

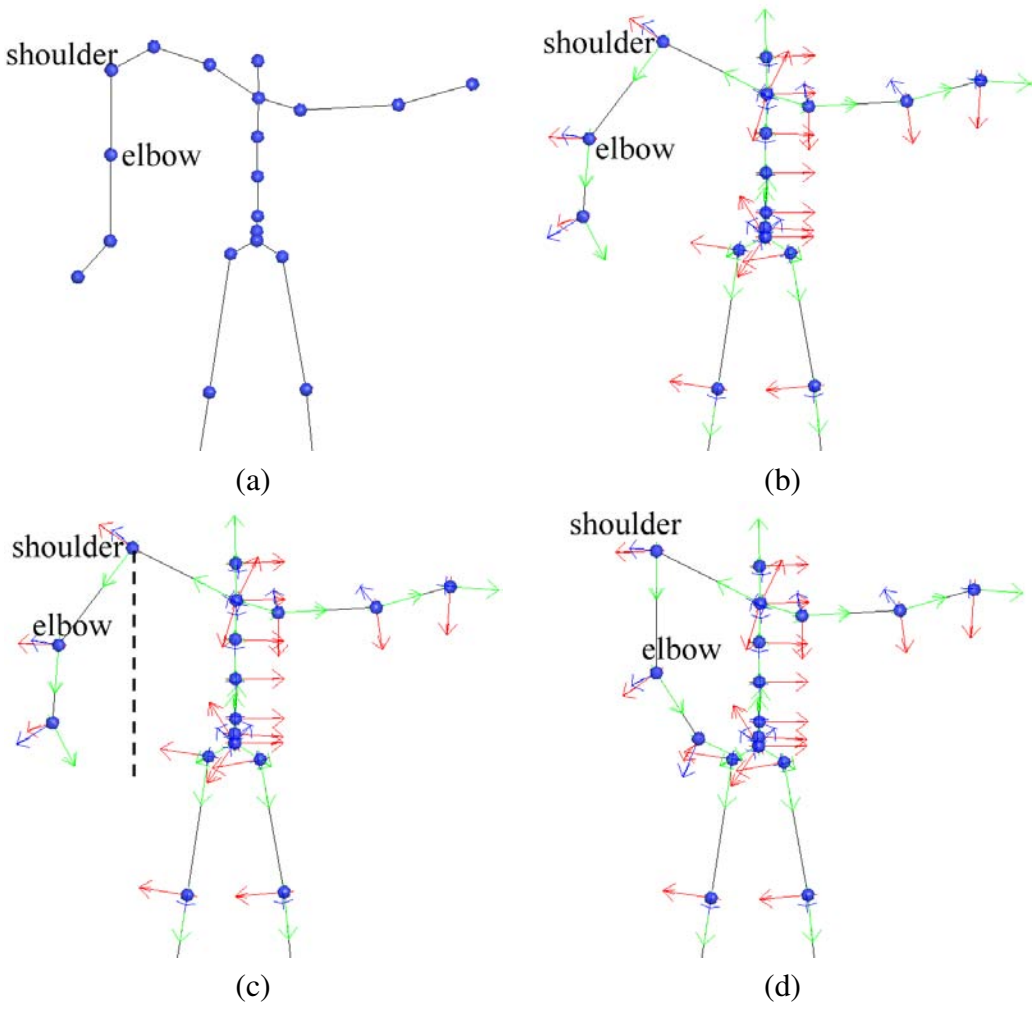


Figure 2.1: (a) The source skeleton. (b) The destination skeleton. (c) The dotted line is the new direction of the elbow. (d) The intermediate skeleton has the same number of nodes and using the same local coordinate frame as the source skeleton, while having the same orientation as the destination skeleton.

local coordinate frame as the source one. The dotted line of the figure is the new orientation of the elbow in which the orientation is the same as the elbow in the source skeleton. The elbow after rotating to the new orientation is shown in figure 2.1 d. After rotating the elbow to the dotted line, all the descendant nodes are rotated in a similar way and the construction of intermediate skeleton is finished.

This method can handle altered number of nodes in the skeleton. For the case that the source skeleton has additional nodes, the nodes are regarded as one single nodes. For the case that the source skeleton has fewer nodes, the value of the

existing nodes is distributed to multiple destination nodes. However, if the initial posture of the source skeleton and the destination skeleton is extremely different, problems may happen if the value of intermediate skeleton is copied directly to the destination skeleton.

Chapter 3

Motion Control Net Theory and Implementation

The flow of our topological retargeting method is divided into three steps. The first step is the conversion from the original motion data into our data representation the motion control net (MCN) and the control net angles (CNA). The second step is the manipulation using our data representation, where the topological retargeting is performed. The final step is the conversion from our data representation back to the original motion data format through the process inverse control net (ICN).

The first step involves the MCN construction which is divided into two parts. The MCN is first constructed by connecting the source character skeleton to the destination character skeleton. After the connection, each parameter of the CNA is calculated.

The task in the second step is the manipulation of the structure of the MCN and the parameters of the CNA. Since the manipulation is done by an animator, the implementation of the topological retargeting method only needs to give the animator the flexibility for adjusting the topology of the character skeleton and controlling the character motion.

The final step is the conversion from the MCN and the CNA back to the original motion data format by using the ICN. We have developed three different methods

for performing the ICN. Each method has a different characteristics.

3.1 Motion Control Net Construction

A MCN is an interconnection between different segments of character skeletons, representing by a graph structure. In this graph, a node represents the child end of a segment. An edge identifies the connection between nodes. A node can either serve as a *source node* to provide data or a *destination node* to receive new data. In the destination node, additional parameters are used to provide extra information to perform the ICN.

Each node of a MCN contains CNA which encapsulate animation data. Each CNA consists of three parameters: the hierarchical data α, the segment-to-segment movement β and the orientation data γ. The first two parameters, α and β, store the position information of the node. The third parameter γ stores the orientation information of the node. All three parameters are rotational angles. The α parameter of all nodes is relative to a common reference point while parameters β and γ of each node are relative to its parent node.

For the rest of the discussion, we have made two assumptions about the original motion data. The first assumption is that the original motion is stored in the Hierarchical Translation Rotation (HTR) format, a commonly used format for storing motion captured data, in which the position and the orientation of each node are relative to its parent. In order to calculate the global position and orientation of each node from the HTR format, the relative translation and the relative rotation of each node are post-multiplied on its parent node. The second assumption is that the translation direction of the relative translation is the y axis of the coordinate frame of its parent node. We make this assumption because our method takes this direction as a reference for the formulation. If the translation direction is changed, certain constants and the choice of the reference axes in the CNA calculation must also be changed too.

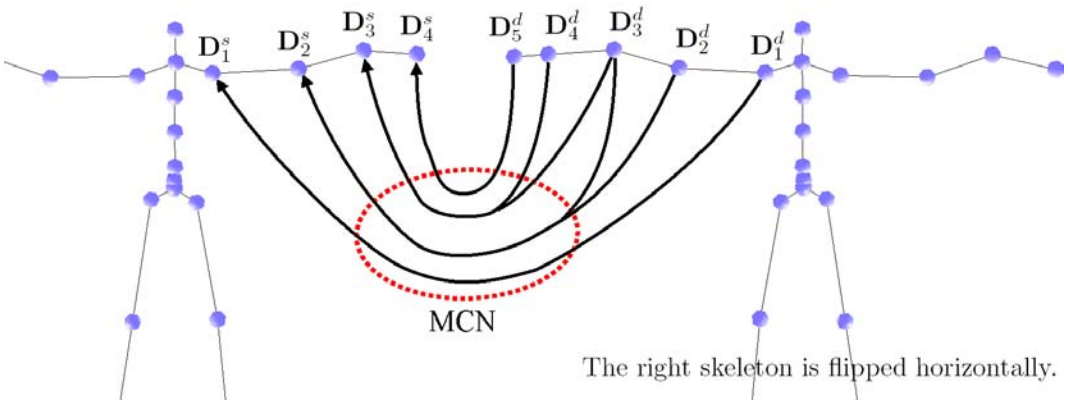


Figure 3.1: A MCN configuration. The nodes of the left skeleton are the source nodes and the nodes of the right one are the destination nodes. The CNA of the node $\mathbf{D}_3^d$ are generated by blending the CNA of nodes $\mathbf{D}_2^s$ and $\mathbf{D}_3^s$.

3.1.1 Motion Control Net Connection

A basic MCN connection requires two character skeletons. The first skeleton is the source character skeleton accompany by a motion clip. The second skeleton is a character skeleton having a topology different (for topological retargeting) from the source character skeleton. The first step of the connection is to set up a *base node* and an *end node* on the source character. These two nodes are the reference nodes for the system to calculate the CNA between them. The next step is to set up a base node and an end node on the destination character. These two nodes notify the system that the nodes between them are the place holder for receiving retargeting information. The final step is to link between the nodes of the source character and the destination character.

The method that we have developed uses the position and the orientation information of the first two nodes of the MCN as reference nodes for calculating the CNA and performing the ICN; therefore, the reference nodes will be kept stationary when the ICN is performed. This property of the reference nodes may lead to some confusions when choosing the base node and end node. The followings are some guidelines for choosing a suitable base node and end node on the source and destination character and setting the connections between them.

Figure 3.1 shows an example of a MCN configuration which is used for retargeting a human arm to another arm having an additional segment in the forearm. First, the base node and the end node are set on the source character skeleton which is the left skeleton in the figure. The end node is the palm, $\mathbf{D}_4^s$. The base node is the node prior to the elbow, the shoulder $\mathbf{D}_1^s$. An identifier number $(\mathbf{D}_i^s,\, i = 1 \cdots n)$ will then be given to each node starting from the base node and continuing to the end node. In this configuration, n is being equal to 4. Since the first two nodes of the MCN are the reference nodes, the base node must be one node prior to the segment chain which is going to perform retargeting.

Second, the base node and the end node are set on the destination character skeleton which is the right skeleton in the figure. The base node is the shoulder, $\mathbf{D}_1^d$, of which the setting allows the ICN to perform motion retargeting starting from the elbow. The end node is the palm, $\mathbf{D}_5^d$. Similarly, the numbering of the destination nodes $\left(\mathbf{D}_i^d,\, i = 1 \cdots n\right)$ starts from the base node $\mathbf{D}_1^d$ and continues to the end node $\mathbf{D}_5^d$, with n being equal to 5.

The final step of connecting the MCN is to specify the linkage. The linking criterion is that one destination node should be linked to one or many source nodes, and different source nodes can be combined to form a new synthetic source node. More importantly, source nodes from different source skeletons can be linked to the nodes of a particular destination character. In figure 3.1, the source nodes $\mathbf{D}_1^s$, $\mathbf{D}_2^s$, $\mathbf{D}_3^s$ and $\mathbf{D}_4^s$ link to the destination nodes $\mathbf{D}_1^d$, $\mathbf{D}_2^d$, $\mathbf{D}_4^d$ and $\mathbf{D}_5^d$ respectively. For the synthesized node, the connection is the the source nodes $\mathbf{D}_2^s$ and $\mathbf{D}_3^s$ linking to the destination node $\mathbf{D}_3^d$, and the CNA of $\mathbf{D}_3^d$ is generated by blending the CNA of the source nodes $\mathbf{D}_2^s$ and $\mathbf{D}_3^s$. The general equations for synthesizing the data of a new source node $\mathbf{D}_{src}^s$ are the linear combinations of the parameters of the CNA: $\alpha_{src} = \sum_i k_i\alpha_i$, $\beta_{src} = \sum_i j_i\beta_i$ and $\gamma_{src} = \sum_i l_i\gamma_i$ where k_i, j_i and l_i are some factor values, and the subscript i is the identifier of the source nodes. The source nodes can come from different source skeletons, and the sum of the factor values would not necessarily be equal to 1.

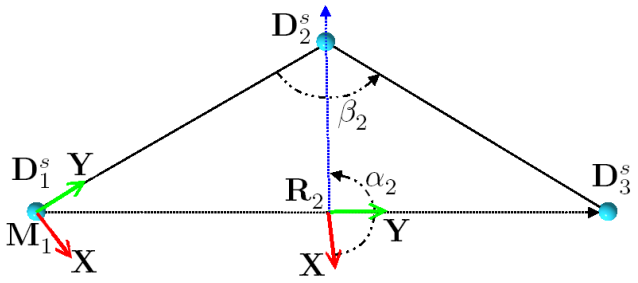


Figure 3.2: The hierarchical data and segment-to-segment movement of the node $\mathbf{D}_2^s$ are named α_2 and β_2 respectively. Only the principle axes x and y are shown in the figure.

3.1.2 Control Net Angles

Each node of a MCN contains CNA, which comprises of three parameters: the hierarchical data α, the segment-to-segment movement β and the orientation data γ. The equations which calculate the parameters are described in the following sections. There are two usages of the equations and two choices of the nodes, depending on the context: when the equations are used for calculating the CNA of a MCN, the nodes are the source nodes; when the equations are used for validating the CNA of a node after performing retargeting through the ICN, the nodes are the destination nodes. In the calculations of the following sections, the nodes default to the source nodes.

3.1.2.1 Hierarchical Data

The hierarchical data α (shown in figure 3.2 and 3.3d) stores the hierarchical information of the MCN. Each α_i of $\mathbf{D}_i^s$ in the MCN is relative to a reference frame $\mathbf{R}_i$. Before calculating the reference frame, we construct an auxiliary frame $\mathbf{B}$ which aligns one of the principle axes of the coordinate frame of the base node to the center line. The center line $\hat{\mathbf{r}^{n/1}}$ is defined as a normalized vector which subtracts the position of $\mathbf{D}_n^s$ from $\mathbf{D}_1^s$. $\mathbf{B}$ is the coordinate frame of the base node post-multiplied by a rotational transformation which aligns $\hat{\mathbf{r}^{2/1}}$ to $\hat{\mathbf{r}^{n/1}}$. With the second assumption of the original motion data, $\hat{\mathbf{r}^{2/1}}$ is the same as the y axis of $\mathbf{M}_1$ which is the

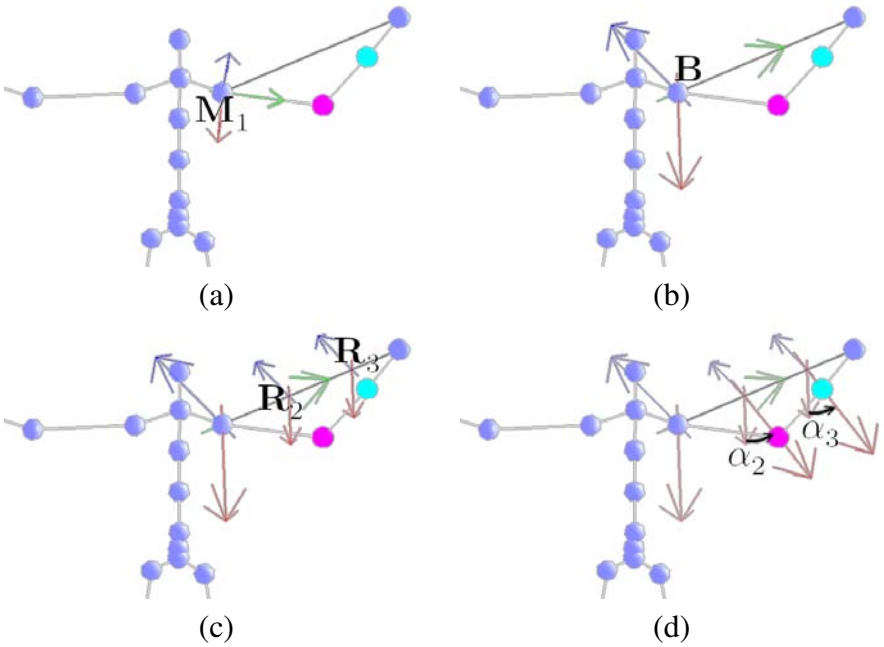


Figure 3.3: (a) The arrows represent the coordinate frame $\mathbf{M}_1$. (b) The arrows represent the auxiliary frame $\mathbf{B}$. (c) Reference frames of different nodes. (d) The hierarchical data.

coordinate frame of $\mathbf{D}_1^s$. The auxiliary frame $\mathbf{B}$ is written as:

$$\mathbf{B} = \mathbf{M}_1 \mathrm{R}_{\hat{\mathbf{r}^{2/1}} \times \hat{\mathbf{r}^{n/1}}} \left(\arccos \left(\hat{\mathbf{r}^{2/1}} \cdot \hat{\mathbf{r}^{n/1}} \right) \right) \tag{3.1}$$

where $\times$ is the cross product operator. $\cdot$ is the scalar product operator. $\mathrm{R}_{\mathbf{x}}(\theta)$ is a rotation function, and its result is a rotational transformation. The parameter $\mathbf{x}$ of the rotation function is the rotation axis, and the parameter θ is the rotation angle. The auxiliary frame $\mathbf{B}$ is then post-multiplied by a translational transformation to form the reference frame $\mathbf{R}_i$. The transformation is formed by the center line and the projection of $\mathbf{r}^{i/1}$ on the center line as the distance. The $\mathbf{R}_i$ is written as:

$$\mathbf{R}_i = \mathbf{B} \mathrm{T}_{\hat{\mathbf{Y}}} \left(\left(\hat{\mathbf{r}^{i/1}} \cdot \hat{\mathbf{r}^{n/1}} \right) \left\| \mathbf{r}^{i/1} \right\| \right) \tag{3.2}$$

where $\mathrm{T}_{\mathbf{x}}(d)$ is a translation function, and its result is a translational transformation. The parameter $\mathbf{x}$ of the translation function is the translation axis, and the parameter

d is the length of the translation. $\hat{\mathbf{Y}}$ is a unit vector $\{0, 1, 0\}^{\mathrm{T}}$. $\mathbf{r}^{i/1}$ is a vector which subtracts the position of $\mathbf{D}_i^s$ from $\mathbf{D}_1^s$. $\|.\|$ returns the length of the input vector.

The magnitude of the hierarchical data is the angle between two vectors about the center line. The first vector is one of the principle axes of the reference frame $\mathbf{R}_i$. With the second assumption of the original motion data, we take the x axis of the reference frame to be the first vector. Another vector $\mathbf{r}^{i/\mathrm{E}_4(\mathbf{R}_i)}$ is from the position of the node $\mathbf{D}_i^s$ to the position of the reference frame $\mathbf{R}_i$. The function $\mathrm{E}_c^r(\mathbf{M})$ is used to extract a vector or a scalar from the input, where c is the column number, r is the row number and $\mathbf{M}$ is the input. In order to extract the position vector from the reference frame, the column number c is set to 4 and the row number r is ignored. The magnitude of α_i is written as:

$$\alpha_i^{'} = \arccos\left(\mathrm{E}_1(\mathbf{R}_i) \cdot \mathbf{r}^{i/\mathrm{E}_4(\mathbf{R}_i)}\right) \tag{3.3}$$

where the x axis (first column) of the input reference frame is extracted by setting c to 1 in the extraction function $\mathrm{E}_c^r(\mathbf{M})$. $\alpha_i^{'}$ is in the range [0, 180] degree. The sign of $\alpha_i^{'}$ is determined by the cross product of the vectors $\mathrm{E}_1(\mathbf{R}_i)$ and $\mathbf{r}^{i/\mathrm{E}_4(\mathbf{R}_i)}$. If the result of the cross product is in the opposite direction of the center line, the sign of α_i is negative. α_i is written as:

$$\alpha_i = \alpha_i^{'} \upsilon\left(\mathrm{E}_1(\mathbf{R}_i) \times \mathbf{r}^{i/\mathrm{E}_4(\mathbf{R}_i)},\ \hat{\mathbf{r}^{n/1}}\right) \tag{3.4}$$

where

$$\upsilon(.,\ .) = \begin{cases} 1 & \text{if both inputs are in the same direction} \\ \text{-1} & \text{otherwise.} \end{cases}$$

From the above equations, α depends on the center line; therefore, if the MCN is re-configured with a different base node and end node, the value of α for a particular node may be different, but the magnitude differences of α with that of its neighbour nodes will remain the same.

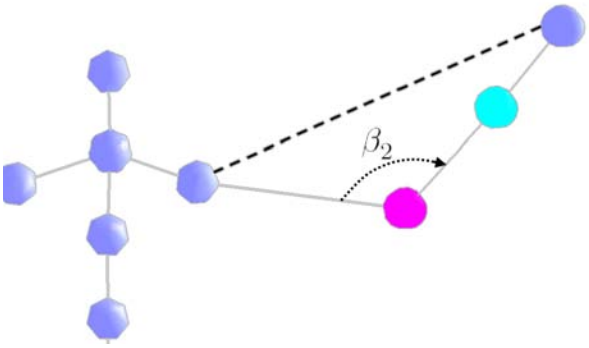


Figure 3.4: The segment-to-segment movement.

3.1.2.2 Segment-to-Segment Movement

The segment-to-Segment Movement β (shown in figure 3.2 and 3.4) stores the angle between two segments. In order to calculate β_i for the node $\mathbf{D}_i^s$, it is necessary to have the position information of the nodes $\mathbf{D}_{i-1}^s$, $\mathbf{D}_i^s$ and $\mathbf{D}_{i+1}^s$. First we write the magnitude of β_i as:

$$\beta_i^{'} \ = \ \arccos\left(\mathbf{r}^{i/\hat{i-1}} \cdot \mathbf{r}^{i\hat{+}1/i}\right) \tag{3.5}$$

Similar to the calculation of α, we also need to determine the sign of β_i by using a reference frame which is the $\mathbf{R}_i$ from equation 3.2 post-multiplied by a rotational transformation. The rotational transformation is formed by the center line and α_i. Since the y axis of $\mathbf{R}_i$ is in the direction of the center line, the rotation axis of the rotational transformation is a y unit vector. The reference frame $\mathbf{R}_i$ becomes:

$$\mathbf{R}_i \ = \ \mathbf{R}_i \mathrm{R}_{\hat{\mathbf{Y}}}\left(\alpha_i\right) \tag{3.6}$$

The sign of β_i is determined by the dot product of the z axis of $\mathbf{R}_i$ and the result of the cross product of the vectors used in equation 3.5:

$$\beta_i \ = \ \beta_i^{'}\sigma\left(\mathrm{E}_3\left(\mathbf{R}_i\right) \cdot \left(\mathbf{r}^{i/\hat{i-1}} \times \mathbf{r}^{i\hat{+}1/i}\right)\right) \tag{3.7}$$

(a) (b) (c) (d)

Figure 3.5: (a) $\mathbf{T}_i$ is the translated coordinate frame of $\mathbf{D}^s_{i-1}$. (b) The auxiliary frame $\mathbf{G}_i$. (c) The coordinate frame $\mathbf{M}_i$ of $\mathbf{D}^s_i$. (d) γ_i is the angle between the x axis of $\mathbf{G}_i$ and the x axis of $\mathbf{M}_i$. Only the principle x axis and y axis are shown in the figure.

where

$$\sigma(x) = \begin{cases} 1 & x \geq 0 \\ \text{-1} & \text{otherwise.} \end{cases}$$

After inspecting the equations, the magnitude of β is a constant, and the sign of β will be changed if the choice of the base node and end node are changed.

3.1.2.3 Orientation Data

The orientation data γ_i stores the differences in rotation between the node $\mathbf{D}^s_i$ and its parent node $\mathbf{D}^s_{i-1}$ about an axis which is in the same direction as the translation direction of the second assumption of the original motion data. In order to find the orientation data, we first define the translated coordinate frame $\mathbf{T}_i$ (shown in figure 3.5a). $\mathbf{T}_i$ is the coordinate frame $\mathbf{M}_{i-1}$ post-multiplied by a translational transformation which is formed by a y unit vector and the translation value of the HTR data of $\mathbf{D}^s_i$. We then create an auxiliary frame $\mathbf{G}_i$ (shown in figure 3.5b) by post-multiplying $\mathbf{T}_i$ with a rotational transformation which is formed by the result of the cross product of the vectors used in equation 3.5 and β'_i:

$$\mathbf{G}_i = \mathbf{T}_i \mathrm{R}_{\mathbf{r}^{i}/\hat{i}_{-1} \times \mathbf{r}^{i+\hat{1}}/i} \left(\beta'_i\right) \tag{3.8}$$

From figure 3.5b and 3.5c, the y axis of $\mathbf{G}_i$ is the same as the y axis of $\mathbf{M}_i$. The magnitude of the orientation data is calculated by the dot product of these two axes:

$$\gamma'_i = \arccos\left(\mathrm{E}_1(\mathbf{G}_i) \cdot \mathrm{E}_1(\mathbf{M}_i)\right) \tag{3.9}$$

Similarly, we have to check the sign of this magnitude. The sign of γ_i is determined by the dot product of the y axis of $\mathbf{M}_i$ and the result of the cross product of the vectors $\mathrm{E}_1(\mathbf{G}_i)$ and $\mathrm{E}_1(\mathbf{M}_i)$. γ_i is written as:

$$\gamma_i = \gamma_i^{'}\sigma\left[\left(\mathrm{E}_1(\mathbf{G}_i) \times \mathrm{E}_1(\mathbf{M}_i)\right) \cdot \mathrm{E}_2(\mathbf{M}_i)\right] \tag{3.10}$$

Since the calculation of γ does not depend on the reference frame, this variable is a constant under different MCN configurations.

3.2 Motion Control Net Manipulation

The MCN manipulation is a user editing process where the topology and animation details modifications are performed. In our implementation, the MCN manipulation includes four major types of editing: displacing a curve to the CNA, setting keyframes on the CNA, re-ordering the nodes of the MCN, and making different combination on the MCN and the CNA. The first two types provide animation details modifications, and the last two types provide topology modifications. One should note that the MCN manipulation mentioned above is just one particular implementation of the possible MCN editing features. There is generally no limitation on how a user performs the MCN manipulation provided that the connection of the resulting MCN is valid to perform the ICN. More details on the manipulation will be given in section 4.5.

The manipulation functions are provided by a set of graphical user interface in our implementation; however, some of the modifications are too complicate to be controlled with the graphical user interface alone, and so they are controlled by a script instead. More details on the graphical user interface and the script will be given in later sections.

3.3 Inverse Control Net Overview

The inverse control net (ICN) is a process that converts a given MCN as well as the related CNA back to a motion format which stores the position and the orientation of each node with reference to the world coordinate. This format is named global format, and it can be converted back to the HTR format, which is the original motion data format, with some existing methods, but they are not mentioned in this thesis.

A major difference between the ICN and other retargeting methods is the end-effector positioning. Most of the retargeting methods require only the position of the end-effector which is given by an animator in order to calculate the joint-space-parameters of the segment chain. In contrast, our method would not necessary need the end-effector position. Instead, we use the end node position information which is embedded in the CNA. Although end-effector position is not a necessary information to perform the ICN, our method can use a given end-effector position for a direct positioning purpose.

The ICN algorithm is the second important part of the MCN theory because the MCN itself is a set of linkages and values, providing no visual feedback for an animator to work with, and the MCN is useful if it can be converted back to the original motion data format. In addition, the response time of the ICN algorithm should be short because a user may need to edit the motion interactively. However, the steps of the ICN which are used to find the embedded end node position information are complicated because the information cannot be restored by simple matrix multiplication, computation time may be long. In order to develop a fast ICN algorithm, we have tried different strategies to restore the end node position information.

The ICN method 1 is a preliminary algorithm. The major purpose of this method is to verify the feasibility of the conversion from the MCN back to the original motion data format. The disadvantage of method 1 is the slowest execution speed because the method is a recursive algorithm. Then the ICN method 2 has been developed as an iterative algorithm with a prediction algorithm which predicts the end node position. With the predicted end node position, the execution speed of

method 2 is faster than method 1. There are two advantages of method 2. The first advantage is that method 2 can find the closet solution if there is no exact solution. The second advantage is that method 2 can partially handle the sensitivity problem which is discussed in section 3.5.8.

To further addressing the sensitivity problem, we have developed the ICN method 3. Method 3 is similar to method 2 as method 3 also utilizes the end node prediction algorithm. The main difference is that method 3 uses a closed-form solution to restore the motion, and this difference makes method 3 the fastest method among the 3 methods. However, method 3 itself still cannot solve the sensitivity problem. A fully automatic sensitivity problem solver is required to solve the problem completely, while method 3 only implemented a preliminary algorithm of the solver. Further development is required to extend and complete the solver in the future.

In the following three sections, we are going to present the three ICN methods. In the discussion, some procedures work on a portion of the segment chain which is indicated by the term *working list*. In addition, we use the superscript to distinguish two types of representation of the CNA. The first one is the CNA of the source nodes, or the modified CNA, represented by α^g, β^g and γ^g. The second one is the restored/recovered CNA of the destination nodes, represented by α^r, β^r and γ^r. In order to calculate the restored/recovered CNA, equations in section 3.1.2.1 are used, but the formulation of the vector $\mathbf{r}^{\hat{i}/j}$ is different from the equations described in the section in which the position of the source nodes is used to form the vector. The vector $\mathbf{r}^{\hat{i}/j}$ used to calculate the restored/recovered CNA is formed by the position of the destination nodes.

3.4 The Inverse Control Net - Method 1

Method 1 takes a brute-force approach for converting the MCN back to the original motion data format. A major disadvantage of the method is that the solution of the method may be trapped in a local minimum. In addition to the trapping problem,

1 Recover the β of the nodes.
2 Call algorithm 1 with $\mathbf{D}_2^d$ and $\mathbf{D}_n^d$ as input parameters to recover the α of the nodes.
3 Recover the γ of the nodes.

Table 3.1: Three phases of the ICN method 1

the execution speed of the method is slow because of the recursive nature of the method. Although there are some disadvantages, we still include the method in this thesis because the method can: verify the feasibility of the conversion from the MCN to the original motion data format, provide a better understanding of the idea of the ICN, and provide an introduction to the other two ICN methods.

Method 1 consists of three phases which are listed in table 3.1. The first phase recovers the segment-to-segment movement β. The second phase recovers the hierarchical data α by calling algorithm 1 with $\mathbf{D}_2^d$ and $\mathbf{D}_n^d$ as input parameters. The position of the nodes are found as β and α are recovered. The final phase recovers the orientation data γ. The most complicated part of method 1 is the second phase, which is a recursive algorithm.

Algorithm 1 shows the steps of the hierarchical data recovery. The algorithm uses an error function to measure the accuracy of the hierarchical data of the segment chain. The function calculates the total sum of differences of the given hierarchical data α_i^g and the recovered hierarchical data α_i^r that is calculated after the recovery. The error function is defined as:

$$err\left(\mathbf{D}_j^d,\, \mathbf{D}_n^d\right) = \sum_{i=j}^{n} \left(\|\alpha_i^g - \alpha_i^r\|\right) \tag{3.11}$$

In line 2 of the algorithm, a for loop with limited iteration is used to prevent the algorithm from running into an infinite loop in case the error function always gives a value larger than a predefined threshold. The case occurs when an iteration adjusts α_i of $\mathbf{D}_i^d$ in order to achieve a minimum error value, and this adjustment affects all α_js of other nodes, making the error value in their next iterations larger than those of their previous iterations. This case is a local minimum trapping problem that

should be avoided. From our experiments, setting the maximum iteration of the for loop to 10 is enough to successfully recover the hierarchical data while preventing the algorithm from running into an infinite loop if no solution can be found.

Algorithm 1 The phase two of the ICN method 1

Input parameters: A *from* destination node $\mathbf{D}_f^d$. A *to* destination node $\mathbf{D}_t^d$.

```
1  void hierar_recover(Node D_f^d, Node D_n^d){
2     for(i=0; i<10; i++){ //avoid infinity looping.
3        if (D_f^d == D_t^d)
4           return;
5        call hierar_recover(D_{f+1}^d, D_t^d);
6        set err_val = err(D_f^d, D_t^d); // call the equation 3.11.
7        if (err_val > threshold){
8           recover the hierarchical data of D_f^d by the method described in
                section 3.4.2.
9           set err_val = err(D_f^d, D_t^d); // call the equation 3.11.
10          if (err_val < threshold)
11             break;
12       }
13       else
14          break;
15       }
16    }
17 }
```

In the following sections, we present the element operations of each phase of the ICN method 1, then present an example of using method 1 to perform a topological retargeting.

3.4.1 Segment-to-Segment Movement Recovering

Segment-to-segment movement recovering is the first phase of the ICN method 1. The working list of this phase is the nodes from $\mathbf{D}_2^d$ to $\mathbf{D}_{n-1}^d$. Before starting any calculations, the rotation data of the working list is reset. The next step is to match the β^rs of the working list to the given β^gs. In order to do this, we need to find a rotation axis which is an arbitrary axis, and we use a x-axis $\{1, 0, 0\}^{\mathrm{T}}$ for simplicity. The rotation axis is then made relative to the coordinate frame of the node by adding the position of the node and pre-multiplying by the inverse of the coordinate frame

of the node. β_i^r of the node $\mathbf{D}_i^d$ is then recovered by post-multiplying $\mathbf{M}_i$ by the rotational transformation which is formed by the rotation axis and β_i^g:

$$\mathbf{M}_i \;=\; \mathbf{M}_i \mathrm{R}_{\mathbf{M}_i^{-1}\left(\{1,0,0\}^{\mathrm{T}}+\mathrm{E}_4\left(\mathbf{M}_i\right)\right)}\left(\beta_i^g\right) \tag{3.12}$$

3.4.2 Hierarchical Data Recovering

The hierarchical data recovering is a step of the phase 2 of the ICN method 1. The idea of recovering hierarchical data is to rotate the node $\mathbf{D}_i^d$ about an axis to minimize the error of α_i^r while preserving β_{i-1}^r. First we find the axis for the rotation. The rotation axis $\mathrm{E}_2\left(\mathbf{M}_{i-1}\right)$ is the y axis of the coordinate frame of $\mathbf{D}_{i-1}^d$, which is equal to $\mathrm{E}_2\left(\mathbf{T}_i\right)$ ($\mathbf{T}_i$is given in section 3.1.2.3). When $\mathbf{M}_i$ is post-multiplied by the rotational transformation which is formed by this axis to recover α_i^r, β_{i-1}^r remains unchanged; however, the sign of β_i^r will be changed. Then the axis is made relative to the node $\mathbf{D}_i^d$ by adding the position of the node and pre-multiplying by the inverse of $\mathbf{M}_i$. The axis is written as:

$$\mathbf{a}_i \;=\; \mathbf{M}_i^{-1}\left(\mathrm{E}_2\left(\mathbf{T}_i\right)+\mathrm{E}_4\left(\mathbf{M}_i\right)\right) \tag{3.13}$$

In order to rotate the node $\mathbf{D}_i^d$, the coordinate frame $\mathbf{M}_i$ of the node is post-multiplied by the rotational transformation which is formed by the axis $\mathbf{a}_i$ and θ_i(given in later steps) :

$$\mathbf{M}_i \;=\; \mathbf{M}_i \mathrm{R}_{\mathbf{a}_i}\left(\theta_i\right) \tag{3.14}$$

The possible range of θ_i is [0, 360] degree. We use a method that is similar to the bisection to find the suitable value of θ_i. However, there exists a problem that the method may find two solutions in the range, one is correct and one is fake. If the method finds θ_i starting with the interval [0, 360], one of the solutions may be missed. In order to solve the problem, the possible range is divided into several sub-intervals. From our experiments, we find that dividing the range into 10 sub-intervals is enough to find the two solutions. The 10 sub-intervals are then passed

to algorithm 2 one by one to find the solution. Since α_i^r and the sign of β_i^r will be changed after applying equation 3.14, these two values are used as the criteria to guide the bisection.

Algorithm 2 lists the steps which find the solution from an interval. The function same_sign that is used in line 9 returns true if the parameter 1 has the same sign as the parameter 2 or the parameter 1 has the same sign as the parameter 3. The algorithm returns no solution if the input interval contains a fake solution or has no solution.

Algorithm 2 The algorithm used to find a solution in an interval.

Input parameters: A given α_i^g and a given β_i^g. An interval information: the left boundary θ_i^L and the right boundary θ_i^R.
Note: α_i^L and β_i^L are calculated by using the $\mathbf{M}_i$ which is calculated by equation 3.14 using the input θ_i^L. α_i^R and β_i^R are calculated similarly.

```
1  float interval_solver(float α_i^g, float β_i^g, float θ_i^L, float θ_i^R){
2     if (β_i^g == β_i^L && |α_i^g − α_i^L| < threshold)
3        return θ_i^L; // check if the left boundary is the solution.
4     if (β_i^g == β_i^R && |α_i^g − α_i^R| < threshold)
5        return θ_i^R; // check if the right boundary is the solution.
6     set θ_i^M = (θ_i^L + θ_i^R)/2; // the θ value in the middle of the interval.
7     set M_i^M by using equation 3.14 with θ_i^M; // update the coordinate
         frame with θ_i^M
8     calculate α_i^M and β_i^M by equation 3.4 and 3.7 using M_i^M;
9     if (α_i^M <= α_i^g && α_i^g <= α_i^R && same_sign (α_i^g, α_i^L, α_i^R))
10       return interval_solver(α_i^g, β_i^g, θ_i^M, θ_i^R);
11    else if (α_i^L <= α_i^g && α_i^g <= α_i^M && same_sign (α_i^g, α_i^L, α_i^R))
12       return interval_solver(α_i^g, β_i^g, θ_i^L, θ_i^M);
13    else
14       return no_solution;
15 }
```

Here is an example of using algorithm 2 to find a solution from a given interval, and the data of this example is gather from an experiment. Initially, the interval which is [0, 360] degree is divided into 10 sub-intervals which are listed in table 3.2. The sub-intervals are then passed to algorithm 2 one by one to find the solution. Table 3.3 lists the given α_i^g, the given β_i^g and the selected intervals which may contain a solution. From the experiment, the selected interval 1 contains no

solution because α_4^r and α_5^r are increasing (according to table 3.2). Without testing the existence of a turning point in between the interval, we cannot guarantee that the interval really contains no solution. Although a turning point may exist in the interval, dividing the initial intervals into 10 sub-intervals can avoid the existence of the turning point; therefore, we do not have to derive equations for testing the turning point.

i^{th} boundary	θ_i	α_i^r	β_i^r
0	0	259.34	19.17
1	36	281.42	19.17
2	72	303.68	19.17
3	108	326.91	19.17
4	144	354.04	19.17
5	180	41.37	19.17
6	216	141.25	19.17
7	252	187.06	19.17
8	288	213.93	19.17
9	324	237.10	19.17

Table 3.2: An example of boundary values.

The given values:		α_i^g	β_i^g
		138.84	19.17
Selected interval 1:	i	θ_i	α_i^r	β_i^r
	4	144	354.04	19.17
	5	180	41.37	19.17
Selected interval 2:	i	θ_i	α_i^r	β_i^r
	5	180	41.37	19.17
	6	216	141.25	19.17

Table 3.3: The given α_i^g, the given β_i^g and the selected intervals.

3.4.3 Orientation Data Recovering

The orientation data recovering is the last phase of the ICN method 1. The working list of this phase is the nodes from $\mathbf{D}_2^d$ to $\mathbf{D}_{n-1}^d$. In order to recover the orientation data, the first step is to calculate the global position of each node and store it in a

variable $\mathbf{g}_i$. The next step is to reset the rotation data of the working list. Instead of actually resetting the rotation data of the list, the same effect can be achieved by using $\mathbf{T}_i$ in the calculation instead of using $\mathbf{M}_i$. Each node $\mathbf{D}_i^d$ is then rotated such that the node $\mathbf{D}_{i+1}^d$ is collided with the position $\mathbf{g}_{i+1}$. The coordinate frame of the node $\mathbf{D}_i^d$ becomes:

$$\mathbf{M}_i \;=\; \mathbf{T}_i \mathrm{R}_{\hat{\mathbf{r}^{i+1}/i} \times \hat{\mathbf{r}^{\mathbf{g}_{i+1}/i}}} \left(\arccos \left(\hat{\mathbf{r}^{i+1}/i} \cdot \hat{\mathbf{r}^{\mathbf{g}_{i+1}/i}} \right) \right) \tag{3.15}$$

Now, the orientation data of each node $\mathbf{D}_i^d$ is zero. The orientation data of the node $\mathbf{D}_i^d$ is then recovered by post-multiplying $\mathrm{M_i}$ with a rotational transformation which is formed by a y unit vector and the givenγ_i^g:

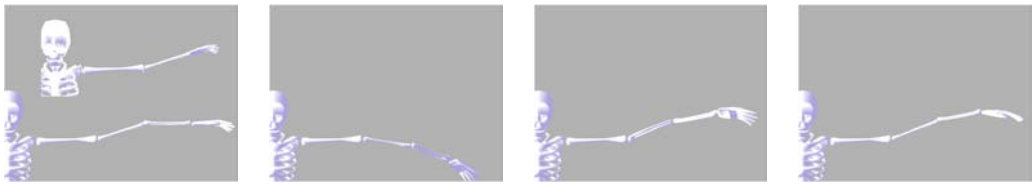
$$\mathbf{M}_i \;=\; \mathbf{M}_i \mathrm{R}_{\hat{\mathbf{Y}}} \left(\gamma_i^g \right) \tag{3.16}$$

3.4.4 A Topological Retargeting Example

Here is an example of using ICN method 1 to perform topological retargeting on the left arm which is shown in figure 3.1. Figure 3.6a shows the arm before applying method 1. Figure 3.6b shows the resulting arm after the segment-to-segment movement recovery. Figure 3.6c shows the resulting arm after calling to algorithm 1, and the algorithm returns at the sixth loop. The iterations of each calling to algorithm 1 are shown in figure 3.7 to figure 3.12. After the position of all nodes is recovered, the last phase to apply is the orientation data recovery. Figure 3.6d shows the final arm after running the whole method 1.

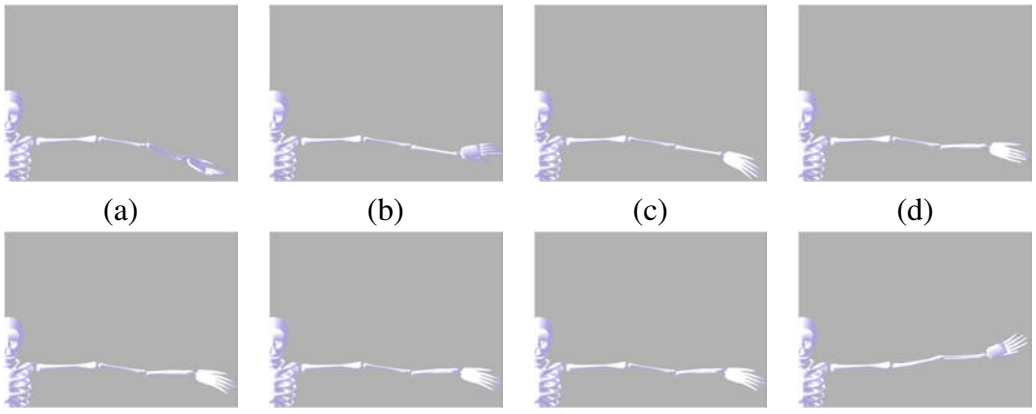
3.5 The Inverse Control Net - Method 2

The ICN method 1 requires a MCN as well as the related CNA to perform the ICN while the ICN method 2 requires one more piece of information, the target direction of the end node, to perform the ICN. The target direction is stored in the



(a) (b) (c) (d)

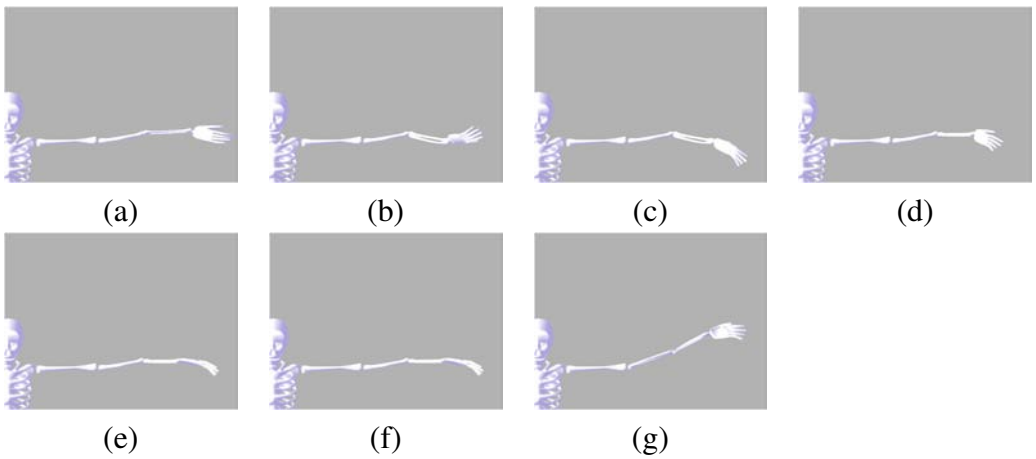
Figure 3.6: (a): Before applying the ICN method 1. The upper skeleton is the original arm. (b): After the segment-to-segment movement recovery. (c): After six iterations of applying algorithm 1. (d): After the orientation data recovery, and the whole method 1 finishes.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 3.7: First iteration of algorithm 1.



(a) (b) (c) (d)

(e) (f) (g)

Figure 3.8: Second iteration of algorithm 1.

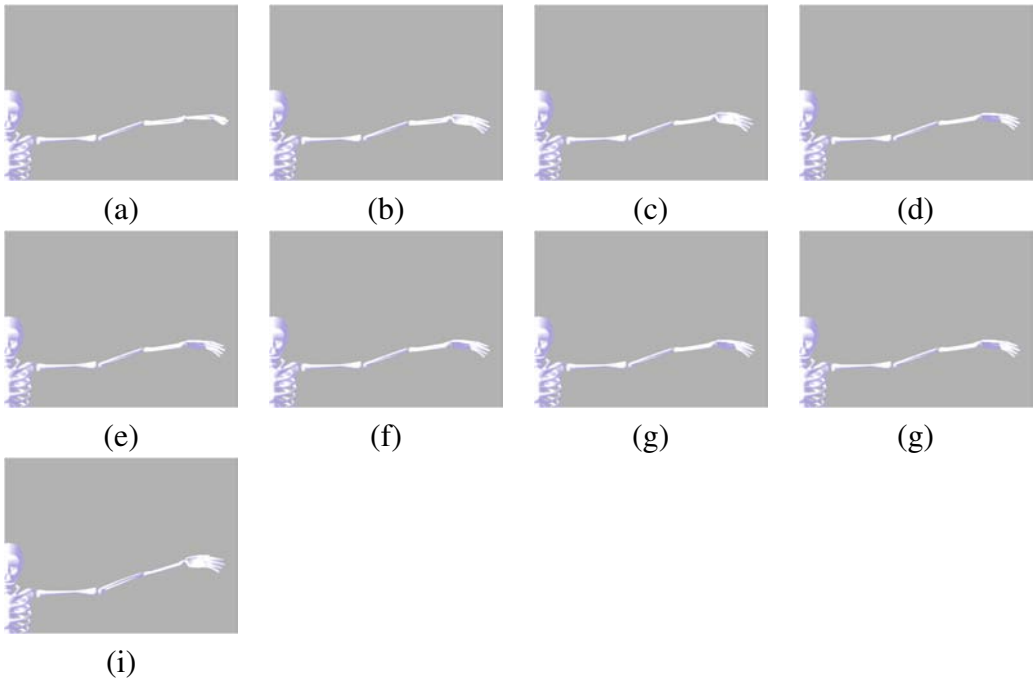


Figure 3.9: Third iteration of algorithm 1.

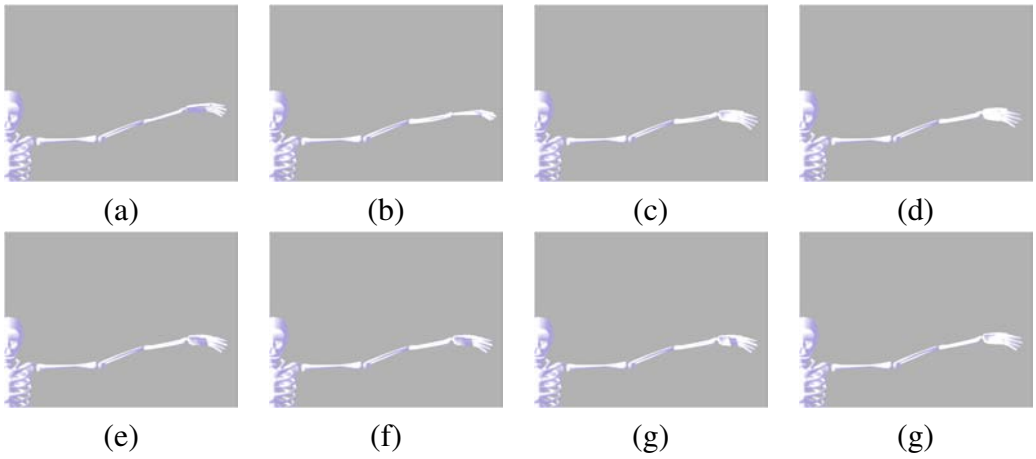


Figure 3.10: Fourth iteration of algorithm 1.

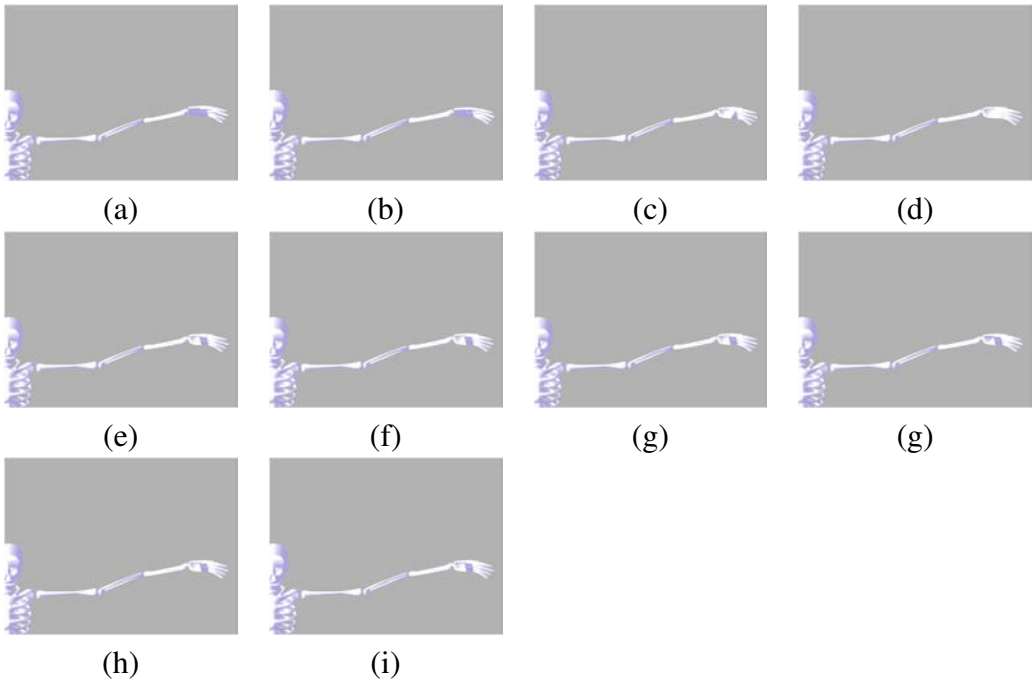


Figure 3.11: Fifth iteration of algorithm 1.

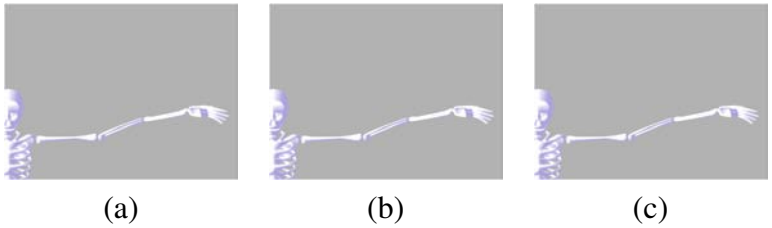


Figure 3.12: Sixth iteration of algorithm 1.

end vector $\mathbf{e}$. The end vector is necessary for method 2 because the hierarchical data depends on the end vector. In fact, method 1 also needs the end vector for calculating the hierarchical data, but method 1 implicitly finds the end vector by the recursive algorithm. In method 2, however, the end vector comes either from a user input or from a prediction algorithm which is similar to the bisection method. With the using of the end vector in the algorithm, the execution speed of method 2 is faster than method 1.

With a given end vector $\mathbf{e}$, method 2 starts performing the ICN. The whole method 2 can be divided into five phases. The first three phases recover the position of the nodes. Position recovery begins with restoring the segment-to-segment movement β, then restoring the hierarchical data α, and followed by restoring the position of the end node. After restoring the position of all nodes, method 2 tests the error of the restored hierarchical data of the restored motion. However, the second phase may not have succeeded in restoring the hierarchical data in certain cases, and these cases will give a large error. If the error is larger than a threshold value, the user may need to adjust the CNA and perform method 2 again. The adjustment is further discussed in section 3.5.8. If the restored nodes have a error smaller than a threshold value, the phase four is processed. In the phase four, method 2 restores the orientation data γ of the nodes. After restoring all parameters of the CNA, an optional phase, phase five, is applied to adjust the position of the end node. The steps of method 2 with a given end vector are listed in algorithm 3. The steps of method 2 without a given end vector are listed in algorithm 4.

In the following sections, we first present each phase of method 2, next present an example of using method 2 to perform topological retargeting, then discuss some problems of method 2. In the sections, we use the word “restore” to describe each phase rather than using the work “recover”. This naming distinction is to avoid the confusion making with the name of the phases of method 1.

Algorithm 3 ICN method 2 with a given end vector.

Input parameter: An end vector e.

```
1  restore the segment–to–segment movement of the nodes;
2  restore the hierarchical data of the nodes;
3  restore the end node position;
4  set error = result of the error function 3.19;
5  if (error < threshold){
6     restores the orientation data of the nodes;
7     optional phase: end node adjustment;
8     return true;
9  }
10 else
11    return false;
```

Algorithm 4 ICN method 2 without a given end vector.

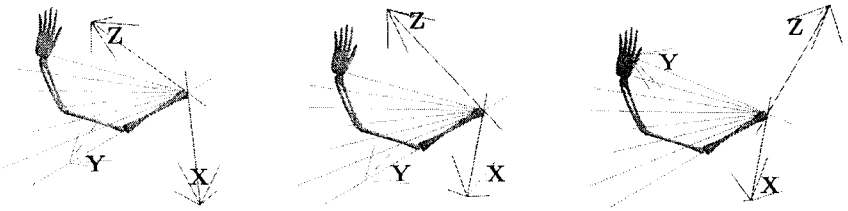
Note: No end vector e is given.

```
1  while(true){
2     predict a new end vector by the method described in section 3.5.1;
3     if (no more predicted end vector)
4        break;
5     set success = call algorithm 3 with the newly predicted end vector as
          the input parameter;
6     if (success == true)
7        break;
8     else
9        calculate an error value by the error function 3.19 for the next
             prediction;
10 }
```

3.5.1 End Vector Prediction

There are two modes for specifying the end vector. The first mode is the user-specifying mode in which the user configures the end node to follow the trajectory of another object or the locus of other nodes of the character. The second mode is the prediction mode which uses the end vector embedded in the MCN. The end vector in the user-specifying mode is formed by normalizing the vector which is formed by subtracting the position of the user positioned end node from the position of the base node.

The end vector in the prediction mode is not found by simple matrix multiplica-



(a) (b) (c)

Figure 3.13: The steps for predicting the eighth end vector in the first prediction iteration. Arrows show the principle axes of: (a) the coordinate frame $\mathbf{M}_1$, (b) the auxiliary frame $\mathbf{A}_1$, (c) the auxiliary frame $\mathbf{A}_1$ post-multiplied by a rotational transformation which is formed by an x-unit vector and ϕ. Other lines are the first seven predicted end vectors.

tion, but by an iterative algorithm. In order to predict the end vector e, we first form an auxiliary frame $\mathbf{A}_1$ which is shown in figure 3.13b. This frame is calculated by post-multiplying the coordinate frame $\mathbf{M}_1$ with the rotational transformation which is formed by a y-unit vector and the rotation angle α_2^g - $\pi/2$. By using this rotation angle in the subsequent calculations, the restored motion always has a correctly restored α_2^r. The auxiliary frame $\mathbf{A}_1$ is written as:

$$\mathbf{A}_1 = \mathbf{M}_1 \mathrm{R}_{\hat{\mathbf{Y}}} \left(\alpha_2^g - \pi/2\right) \tag{3.17}$$

The end vector e is the y axis of the auxiliary frame after being post-multiplied by a rotational transformation which is formed by a x-unit vector and an angle ϕ (given in later steps). Figure 3.13c shows the auxiliary frame after the post-multiplication. The end vector is written as:

$$\mathbf{e} = \mathrm{E}_2 \left(\mathbf{A}_1 \mathrm{R}_{\hat{\mathbf{X}}} \left(\phi\right)\right) \tag{3.18}$$

The rotation angle ϕ is in the range [0, 180] degree, and ϕ is found by a modified bisection method (the modifications are given later). In each bisection iteration, an error value that is calculated from an error function is used as a criterion for further bisection. The error function calculates the error at line 9 of algorithm 4. The error

function is defined as:

$$err \;=\; \sum_{i=2}^{n-1} \left(\exp\left(n-i\right) . \left\| \mathbf{r}^{i/i-1} \right\| . \left| \alpha_i^g - \alpha_i^r \right| \right) \tag{3.19}$$

where $\exp\left(.\right)$ is an exponential function, and $\left\| \mathbf{r}^{i/i-1} \right\|$ is the length of the segment between the nodes $\mathbf{D}_i^d$ and $\mathbf{D}_{i-1}^d$. The purpose of the exponential term is to increase the importance of the error at the nodes closer to the base node because these nodes have a larger impact on the position of the end node. The multiplication of the segment length also magnifies the importance. Although this error function is an approximate measure for the segment chain, it works well in our bisection process as it is our goal to find either a minimal error posture or a correct posture. Should the case arise in which no predicted end vector is able to generate an error-free motion, the motion with the smallest error will be chosen as the final solution.

There is an important modification to the bisection iterations. In each iteration, the bisection interval is divided into several fixed-size sub-intervals, and the sub-interval having the smallest error value is selected for the next bisection iteration. The error value of the sub-intervals is calculated by summing the left and the right boundary error, which are calculated by equation 3.19, of the sub-intervals. We make this modification because there may be more than one local minimum in each iteration, especially in the first iteration. This modification can avoid selecting a local minimum which contains no solution. Typically, dividing the division into twenty intervals suffices to find the solution. Therefore, a single iteration of the modified bisection gives twenty predicted end vectors.

3.5.2 Restoration of Segment-to-Segment Movement

This is the first phase of the ICN method 2. This phase restores the segment-to-segment movement. The working list of this phase is the nodes from $\mathbf{D}_2^d$ to $\mathbf{D}_{n-2}^d$. Before starting any calculations, the rotation data of the working list is reset. This step is equal to the step which is presented in section 3.4.1. The next step is to

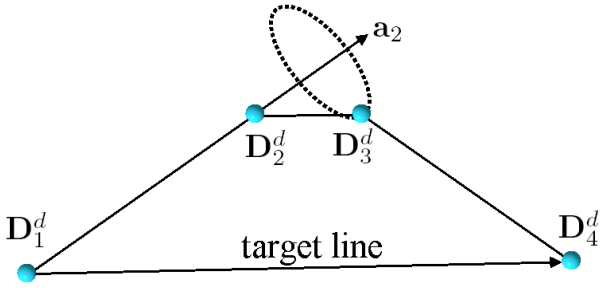


Figure 3.14: The position of $\mathbf{D}_3^d$ lies on the dotted ellipse if the magnitude of β_2^g is kept constant. $\mathbf{a}_2$ is the rotation axis for restoring the hierarchical data of the node $\mathbf{D}_3^d$.

adjust the angle between the nodes until β^r is matched to β^g. Equation 3.12 is then applied to the working list to restore the segment-to-segment movement.

3.5.3 Restoration of Hierarchy Data

This is the second phase of the ICN method 2. The idea of restoring the hierarchical data is to rotate the nodes about an axis $\mathbf{a}_i$ to minimize the error of the hierarchical data while preserving the segment-to-segment movement of their parent nodes. We do not use an analytical solution for this phase because an exact solution does not always exist. For example, consider restoring the hierarchical data for the node $\mathbf{D}_3^d$ in figure 3.14. The segment between $\mathbf{D}_2^d$ and $\mathbf{D}_3^d$ is short with compared to its neighbors. If the magnitude of β_2^g is kept constant, the possible location of $\mathbf{D}_3^d$ lies on the dotted ellipse which is shown in the figure and the restored α_3^r is limited inside a range. Even if β_2^g is a variable, α_3^r still lies on an ellipse which has a larger diameter. In order to find a closest solution, a numerical method is used instead.

Before restoring the hierarchical data, we first define a target line which is the end vector $\mathbf{e}$ multiplied by the distance between the node $\mathbf{D}_1^d$ and the node $\mathbf{D}_n^d$. This target line replaces the center line when the equations which are described in section 3.1.2.2 are used to validate the restored β^r. We next compute the axis $\mathbf{a}_{i-1}$ which is used to rotate the node $\mathbf{D}_i^d$. When $\mathbf{M}_{i-1}$ is post-multiplied by the rotational transformation which is formed by $\mathbf{a}_{i-1}$ and an arbitrary rotation degree,

the magnitude of β_{i-1}^{r} remains unchanged. The axis $\mathbf{a}_{i-1}$ is the extension of the y axis of $\mathbf{T}_{i-1}$. $\mathbf{a}_{i-1}$ is equal to the vector $\mathbf{r}^{i-1/i-2}$ added the position of $\mathbf{D}_{i-1}^{d}$ and then made relative to the coordinate frame of $\mathbf{D}_{i-1}^{d}$.

$$\mathbf{a}_{i-1} \;=\; \mathbf{M}_{i-1}^{-1}\left(\mathbf{r}^{i-1/i-2} + \mathbf{r}^{i-1/1}\right) \tag{3.20}$$

However, $\mathbf{a}_{i-1}$ is not a good choice for the subsequent calculations because the rotational transformation which is formed by this axis involves several sine and cosine functions, the numerical errors in the calculation can be large and the formulation of derivatives is more complicated. A better formulation of the axis is formed by shifting the origin of the axis from $\mathbf{M}_{i-1}$ to $\mathbf{T}_{i-1}$. After the shifting, the axis becomes the y axis of coordinate frame $\mathbf{T}_{i-1}$. The new position of $\mathbf{D}_{i}^{d}$ is then calculated by rotating $\mathbf{r}^{i/i-1}$ about a y unit vector with θ_i (given in later steps):

$$\mathbf{p}_i \;=\; \mathrm{R}_{\hat{\mathbf{Y}}}\left(\theta_i\right)\mathbf{r}_i \tag{3.21}$$

where

$$\mathbf{r}_i \;=\; \mathbf{T}_{i-1}^{-1}\mathbf{r}^{i/i-1}$$

The next step is to determine α_i^r after the rotation. The calculations are similar to the equations described in section 3.1.2.1 but having the origin shifted. First we calculate the reference frame of which the formulation is similar to equation 3.2. The reference frame is written as:

$$\mathbf{R}_i \;=\; \mathbf{B}\mathrm{T}_{\hat{\mathbf{Y}}}\left(\left(\mathbf{p}_i - \mathbf{q}_1\right)\cdot \mathrm{E}_2\left(\mathbf{B}\right)\right) \tag{3.22}$$

where

$$\mathbf{B} \;=\; \mathbf{T}_{i-1}^{-1}\mathbf{M}_1\mathrm{R}_{\mathbf{r}^{\hat{2}/1}\times\mathbf{r}^{\hat{e}/1}}\left(\mathbf{r}^{\hat{2}/1}\cdot\mathbf{r}^{\hat{\mathbf{e}}/1}\right) \tag{3.23}$$

$$\mathbf{q}_1 \;=\mathbf{T}_{i-1}^{-1}\mathrm{E}_4\left(\mathbf{M}_1\right) \tag{3.24}$$

Then, the magnitude of α_i^r with arbitrary rotation angle is written as:

$$\alpha_i^{r'} = \arccos(t_i) \tag{3.25}$$

where

$$t_i = \mathbf{s}_i \cdot \mathrm{E}_1(\mathbf{R}_i) / \|\mathbf{s}_i\| \tag{3.26}$$

$$\mathbf{s}_i = \mathbf{p}_i - \mathrm{E}_4(\mathbf{R}_i) \tag{3.27}$$

Since the rotation part of $\mathbf{R}_i$ is the same as $\mathbf{B}$, equation 3.26 can be written as:

$$t_i = \mathbf{s}_i \cdot \mathrm{E}_1(\mathbf{B}) / \|\mathbf{s}_i\| \tag{3.28}$$

The sign determination is similar to equation 3.4:

$$\alpha_i^r = \alpha_i^{r'} \upsilon(\mathbf{s}_i \times \mathrm{E}_1(\mathbf{B}), \mathrm{E}_2(\mathbf{B})) \tag{3.29}$$

Since we do not know the rotation angle θ_i in advance, hierarchical data of the node $\mathbf{D}_i^d$ is restored by bisecting θ_i from 0 to 2π. In each bisection, α_i^r of $\mathbf{D}_i^d$ is checked with equation 3.29. In some cases, such as the configuration shown in figure 3.14, there will be two solutions, and one solution is a fake solution. The fake solution can be eliminated by checking the sign of β_{i-1}^r using the equations described in section 3.1.2.2 in which the center line is replaced by the target line.

If we plot θ_i against α_i^r, the graph will show that there are two turning points for certain configurations, such as the one shown in figure 3.14. If we consider only the value of α_i^r and β_{i-1}^r, these turning points will cause problems in the bisection because if the solution is closer to the turning point and the solution lies in the bisection interval, the system will determine that this interval contains no solution. This problem can be solved by checking the derivative of equation 3.29. Since we need only the sign of the derivative, we take the derivative of equation 3.28

instead, and there is no possibility of division by zero when calculates the arc cosine derivative. The derivative of equation 3.28 is written as:

$$\frac{d\,t_i}{d\,\theta_i} = \frac{\frac{d\,\mathrm{E}^1(\mathbf{s}_i)}{d\,\theta_i}\cdot\mathrm{E}_1^1(\mathbf{B})+\frac{d\,\mathrm{E}^2(\mathbf{s}_i)}{d\,\theta_i}\mathrm{E}_1^2(\mathbf{B})+\frac{d\,\mathrm{E}^3(\mathbf{s}_i)}{d\,\theta_i}\mathrm{E}_1^3(\mathbf{B})}{\|\mathbf{s}_i\|} - \frac{\left[\mathrm{E}^1(\mathbf{s}_i)\mathrm{E}_1^1(\mathbf{B})+\mathrm{E}^2(\mathbf{s}_i)\mathrm{E}_1^2(\mathbf{B})+\mathrm{E}^3(\mathbf{s}_i)\mathrm{E}_1^3(\mathbf{B})\right]\frac{d\,\|\mathbf{s}_i\|}{d\,\theta_i}}{\mathbf{s}_i\cdot\mathbf{s}_i} \tag{3.30}$$

where

$$\frac{d\,\mathrm{E}^1(\mathbf{s}_i)}{d\,\theta_i} = -s\mathrm{E}^1(\mathbf{r}_i) + c(\theta_i)\,\mathrm{E}^3(\mathbf{r}_i) - \mathrm{E}_2^1(\mathbf{B})\frac{d\,g_i}{d\,\theta_i} \tag{3.31}$$

$$\frac{d\,\mathrm{E}^2(\mathbf{s}_i)}{d\,\theta_i} = -\mathrm{E}_2^2(\mathbf{B})\frac{d\,g_i}{d\,\theta_i} \tag{3.32}$$

$$\frac{d\,\mathrm{E}^3(\mathbf{s}_i)}{d\,\theta_i} = -c\mathrm{E}^1(\mathbf{r}_i) - s\mathrm{E}^3(\mathbf{r}_i) - \mathrm{E}_2^3(\mathbf{B})\frac{d\,g_i}{d\,\theta_i} \tag{3.33}$$

$$\frac{d\,g_i}{d\,\theta_i} = \left(-s\mathrm{E}^1(\mathbf{r}_i) + c\mathrm{E}^3(\mathbf{r}_i)\right)\mathrm{E}_2^1(\mathbf{B}) + \left(-c\mathrm{E}^1(\mathbf{r}_i) - s\mathrm{E}^3(\mathbf{r}_i)\right)\mathrm{E}_2^3(\mathbf{B}) \tag{3.34}$$

$$\frac{d\,\|\mathbf{s}_i\|}{d\,\theta_i} = \frac{\frac{d\,\mathrm{E}^1(\mathbf{s}_i)}{d\,\theta_i}\mathrm{E}^1(\mathbf{s}_i) + \frac{d\,\mathrm{E}^2(\mathbf{s}_i)}{d\,\theta_i}\mathrm{E}^2(\mathbf{s}_i) + \frac{d\,\mathrm{E}^3(\mathbf{s}_i)}{d\,\theta_i}\mathrm{E}^3(\mathbf{s}_i)}{\|\mathbf{s}_i\|} \tag{3.35}$$

$$s = \sin(\theta_i) \tag{3.36}$$

$$c = \cos(\theta_i) \tag{3.37}$$

When the extraction function $\mathrm{E}_c^r(\mathbf{V})$ is applied to a vector $\mathbf{V}$ without using the parameter c, the function returns a scalar value from the row r of the input vector $\mathbf{V}$.

Bisection is applied on the nodes from $\mathbf{D}_3^d$ to $\mathbf{D}_{n-1}^d$. The node $\mathbf{D}_2^d$ is not included in the calculation because the node is a part of the reference. Although there may be some errors introduced in each node in the bisection iteration, these errors will not propagate to their child node because the calculation of all hierarchical data depends on the same reference.

3.5.4 Restoration of the End Node Position

Restoration of the end node position is the third phase of the ICN method 2. This phase restores the position of the end node $\mathbf{D}_n^d$, at the same time preserving β_{n-1}^r and α_{n-1}^r. In order to achieve this, the y axis of the coordinate frame $\mathbf{M}_{n-1}$ is aligned to the plane $\mathbf{p}$ which is formed by the vector $\mathbf{r}^{n-1/1}$ and $\mathbf{e}$. This step ensures that α_{n-1}^r is equal to α_{n-1}^g. The y axis of the coordinate frame $\mathbf{M}_{n-1}$ is aligned to the plane $\mathbf{p}$ by:

$$\mathbf{M}_{n-1} \;=\; \mathbf{M}_{n-1}\mathrm{R}_{\hat{\mathbf{n}}\times\left(\mathbf{M}_{n-1}^{-1}\hat{\mathbf{r}^{n/1}}\right)}\left(\pi/2 - \arccos\left(\hat{\mathbf{n}}\cdot\left(\mathbf{M}_{n-1}^{-1}\hat{\mathbf{r}^{n/1}}\right)\right)\right) \tag{3.38}$$

where

$$\hat{\mathbf{n}} \;=\; \mathbf{M}_{n-1}^{-1}\left(\hat{\mathbf{r}^{n-1/1}}\times\mathbf{e} + \mathbf{r}^{n-1/1}\right) \tag{3.39}$$

Now, the y axis of the coordinate frame $\mathbf{M}_{n-1}$ is in the plane $\mathbf{p}$. We then apply a rotation on coordinate frame $\mathbf{M}_{n-1}$ about the vector $\hat{\mathbf{n}}$ which is perpendicular to plane $\mathbf{p}$ in order to match β_{n-1}^r to β_{n-1}^g:

$$\mathbf{M}_{n-1} \;=\; \mathbf{M}_{n-1}\mathrm{R}_{\hat{\mathbf{n}}}\left(\beta_{n-1}^g\right) \tag{3.40}$$

One should note that $\mathbf{M}_{n-1}$ on the right hand side of equation 3.40 is the resulting frame given by equation 3.38, and equation 3.39 must be applied again for calculating $\hat{\mathbf{n}}$ in equation 3.40.

3.5.5 Restoration of the Orientation Data

This is the last phase of the ICN method 2. The procedures for restoring the orientation data is the same as the procedures described in section 3.4.3.

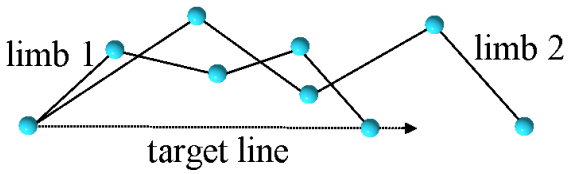


Figure 3.15: Two possible motions restored by the ICN method 2. The position of the end node of limb 1 and limb 2 lies along the direction of the target line.

3.5.6 End Node Adjustment

This phase is an optional phase. The ICN method 2 restores the motion based on the end vector. Since the ICN method 2 utilizes only the directional information of the end vector for the restoration, the end node of the restored motion lies along the direction of the end vector. This situation is shown in figure 3.15. If the skeleton of the destination character is similar to that of the source character, the end node of the resulting motion is in the given position. However, most of the topological retargeting works on an altered skeleton. It is necessary to adjust the position of the end node if positional constraints are imposed. Therefore, this phase is required only if there is a need to adjust the end node to satisfy the constraint.

In order to fix the position of the end node, we can adjust the β of each node of the segment chain, but it is more time-consuming. Instead, we can simply use an IK algorithm to fix the position of the end node.

3.5.7 A Topological Retargeting Example

Here is a simple example of performing topological retargeting on the left arm which is shown in figure 3.1. In this example, no end vector is given, and it is predicted by the prediction algorithm described in section 3.5.1. Figures 3.16a to 3.16d show the resulting arm after performing the first three phases of the ICN method 2 using the seventh end vector predicted in the first iteration of the prediction algorithm. The first step of the first phase is to reset the rotation data of nodes from $\mathbf{D}_2^d$ to $\mathbf{D}_4^d$. The resulting arm is the lower arm shown in figure 3.16a, and the

upper arm is the source arm. Figure 3.16b shows the result of the second step of the first phase which restores the segment-to-segment movement for nodes $\mathbf{D}_2^d$ and $\mathbf{D}_3^d$. Figure 3.16c shows the result of the second phase which restores the hierarchical data of nodes $\mathbf{D}_2^d$ and $\mathbf{D}_3^d$. Figure 3.16d shows the result of the third phase which restores the position of the end node $\mathbf{D}_5^d$. After restoring the position of the nodes from $\mathbf{D}_2^d$ to $\mathbf{D}_4^d$, the system tests the error of the pose and determines that the error of this pose is larger then a predefined threshold value. A new end vector is then predicted to perform another iteration of the ICN. Figure 3.16e shows the resulting arm after performing a single ICN iteration using the last end vector predicted in the first iteration of the prediction algorithm. Figure 3.16f shows the resulting arm after performing a single ICN iteration using the last end vector predicted in the second iteration of the prediction algorithm. Since the error of the restored motion is still larger than the threshold, the end vector prediction algorithm continues. Figures 3.16g to figure 3.16j show the steps of the last ICN iteration using the end vector predicted in the third iteration of the prediction algorithm. Since the error of the restored motion is smaller than the threshold, the prediction algorithm ends. Finally, the fourth phase restores the orientation, and the whole retargeting ends. The final result is shown in figure 3.16k.

3.5.8 Sensitivity Problem on the Hierarchical Data

The ICN method 2 can find a closet solution with smallest error, and the accuracy of the solution is highly dependent on the end vector prediction algorithm which is in turn dependent on the error function 3.19. The sensitivity problem occurs when α^g is set to a value that method 2 can only find a closet solution, and this solution has a bad continuity with its neighbor frames in the restored motion although the solution has the smallest error. In the restoration of the hierarchical data, every restored α_i^r is bounded to a range. An example of the range is shown in figure 3.14 in which the restored α_3^r is bounded to the range $[\theta_i, \theta_j]$ given that the β_2^g is a constant and the segment between $\mathbf{D}_2^d$ and $\mathbf{D}_3^d$ is short with compared to its neighbors. If β_2^g is a

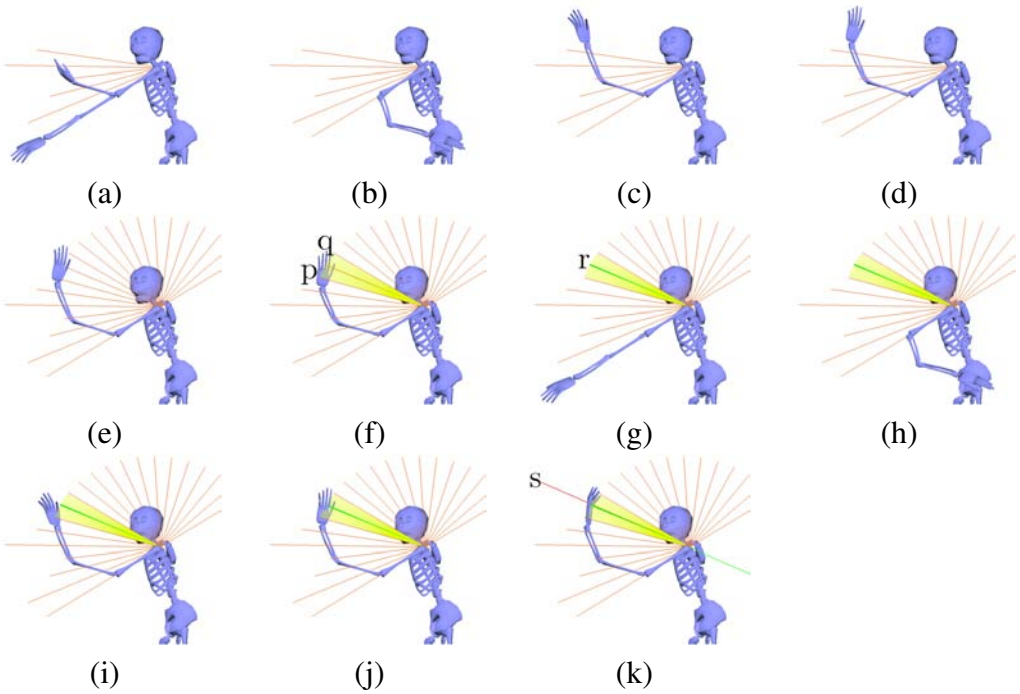


Figure 3.16: The steps of a topological retargeting example. The six lines, which are shown from (a) to (d), represent the end vectors predicted in the first iteration of the prediction algorithm. The lines are counted clock-wise. The lines between p and q in (f) are the end vectors predicted in the second iteration of the prediction algorithm. The lines around r in (g) are the end vectors in the third iteration, although the separation of the lines is so small such that the lines are packed together. The line s in (k) is the final end vector. The source arm is only shown in (a) for reference.

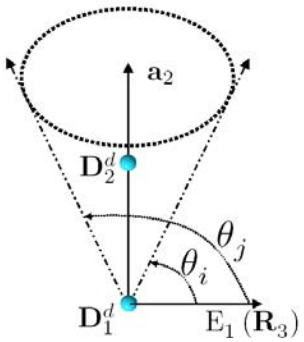


Figure 3.17: Another view of nodes $\mathbf{D}_1^d$ and $\mathbf{D}_2^d$ in figure 3.14(without showing $\mathbf{D}_3^d$ and $\mathbf{D}_4^d$). The viewpoint is at the right hand side of figure 3.14 and is point at the target line (viewing the target line as a point, and the point is in the same position as $\mathbf{D}_1^d$.). The dotted ellipse is the possible position of $\mathbf{D}_3^d$, and the restored α_3^r is bounded to the range $[\theta_i,\ \theta_j]$.

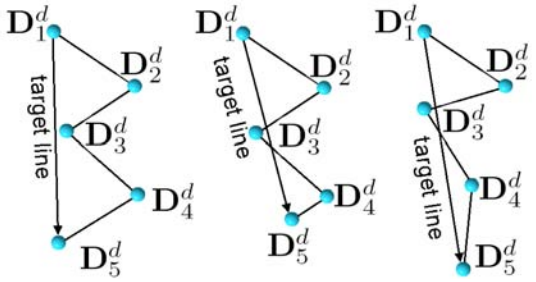


Figure 3.18: Left: The sign of β_3^g is negative. Middle: The sign of β_3^r is positive after scaling some segments. Right: The sign of β_3^r is positive after modifying β_2^g and β_4^g.

variable allowing α_3^r to have a wider range, however, the range of α_3^r is still limited.

When the sensitivity problem occurs, a user can solve the problem by adjusting the hierarchical data and performing the ICN again; however, our implementation of the topological retargeting system can only provide limited hints for the user to adjust the parameters of the CNA. Moreover, there may be a side effect on adjusting the parameters. For example, assuming the user adjusts α_i to solve the sensitivity problem, the sensitivity problem may then shift to the parameter α_{i+1}. In this case, the user does not know which parameter can be adjusted in order to solve the problem. In the worst case, the problem may not be solved by any adjustments on the hierarchical data, or can be solved by adjusting all hierarchical data.

In our implementation of the topological retargeting method, the user can adjust the alpha parameter with the visual feedback from the system user interface, but there are no other hints providing to the user for the adjustment. Although the response time of the ICN method 2 is nearly real time and the user needs no waiting, editing is still a tedious job. There is another problem occurs after adjusting the hierarchical data. Since the shape of the pose highly depends on the hierarchical data. If we make heavy changes to the hierarchical data, the resulting motion will lose its original shape.

3.5.9 Segment-to-Segment Movement Sign Changing Problem

In the MCN manipulation, there are four types of editing can be applied to the MCN. During the manipulation, when the segment lengths of the destination character skeleton are changed, or some segment-to-segment movements are changed, there may exist a sign changing problem of the restored segment-to-segment movement. Figure 3.18 shows an example of the problem. In the figure, there are three skeletons which have different signs of the restored β_3^r. The sign of β_3^r can be simply determined by the relative position of the node $\mathbf{D}_3^d$ to the target line. If the node $\mathbf{D}_3^d$ is on the left hand side of the target line, then the sign of β_3^r is positive. The differences of the three skeletons in the figure are: the left most skeleton I in the figure is the source skeleton, and the sign of β_3^g is negative; the middle skeleton II is the destination skeleton having some segment lengths modified, and the sign of β_3^r is positive; the right most skeleton III is the destination skeleton which has β_2^g and β_4^g modified, and the sign of β_3^r is positive. However, the restored β_3^r of skeleton II and skeleton III should have a negative sign because β_3^g of the source skeleton is negative. In case the sign of the given β_i^g is different from the restored β_i^r, α_i^g should be changed to $\alpha_i^g + \pi$ in order for the ICN method 2 to restore the motion correctly.

In case the problem occurs when performing the ICN, we can adjust the sign of β^g and shift α^g by π to solve the problem; however, making change in one node may affect other nodes. In the worst case, changing the sign of β_i^g will shift the problem to nodes other than $\mathbf{D}_i^d$. A better solution is to perform matching on the position of the nodes of the current frame with those of the neighbor frames. However, the node position matching is not suitable to be applied to the ICN method 2 because the system needs to generate different combinations of adjusted CNA for matching, and this will increase the complexity of the ICN method 2 by ${}_nC_2$. In contrast, the ICN method 3 does not need to generate different combinations of adjusted CNA for matching because the steps of method 3 give multiple solutions which are equal to the combinations of adjusted CNA, and the node position matching is an essential step of the ICN method 3.

3.6 The Inverse Control Net - Method 3

The ICN method 3 takes another approach to restore the hierarchical data. Method 3 restores the position of the nodes in a single phase by solving a set of equations which is based on the geometrical property of α and β. Unlike method 2 which uses a numerical method to restore the position of the nodes, method 3 uses an analytical method; therefore, the execution speed of method 3 is much faster than that of method 2. However, the analytical solution of method 3 cannot find a closet solution, hence the method cannot solve the sensitivity problem. Actually, method 3 only implemented a preliminary algorithm of the sensitivity solver. Further development is required to extend and complete the solver in the future.

The ICN method 3 is similar to the ICN method 2, as method 3 also uses the end vector prediction algorithm. The only difference is the step for restoring α and β. Method 3 does not restore α and β in two different phases but in a single phase. We make this change by considering the geometrical property of α and β. Three equations are then set for relating the parameters and the geometrical property, and the position of the nodes is found by solving these three equations. The steps of method 3 with a given end vector are listed in algorithm 5. The steps of method 3 without a given end vector are listed in algorithm 6.

In the following section, we present the phase that is used to restore the node position. The rest of the phases which are mentioned in algorithm 5 and algorithm 6 will not be discussed in this section because these phases are already presented in section 3.5.

Algorithm 5 ICN method 3 with a given end vector.

Input parameter: An end vector $\mathbf{e}$.

```
1  restore the position of the nodes from D_3^d to D_{n-1}^d by the method
       described in section 3.6.1;
2  if (the position of any node cannot be restored by line 1){
3     return false;
4  }
5  restore the end node position;
6  set error = result of the error function 3.19;
7  if (error < threshold){
8     restores the orientation data of the nodes;
9     optional phase: adjusting the end node;
10    return true;
11 }
12 else
13    return false;
```

Algorithm 6 ICN method 3 without an end vector.

Note: No end vector $\mathbf{e}$ is given.

```
1  while(true){
2     predict a new end vector by the method described in section 3.5.1;
3     if (no more predicted end vector)
4        break;
5     set success = call algorithm 5 with the newly predicted end vector as
          the input parameter;
6     if (success == true)
7        break;
8     else
9        calculate an error value by the error function 3.19 for the next
             prediction;
10 }
```

3.6.1 Node Position Restoration

There are three steps for restoring the position of the nodes. The first step is to calculate the variables r and s which are shown in figure 3.19. There are maximum four pair of solutions of r and s. The second step is to prune the solutions, and only maximum 2 solutions left after the pruning. The third step is to select the final solution pair by comparing the position of the nodes of the current frame with that

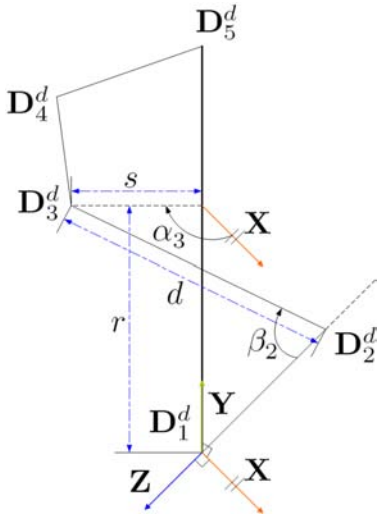


Figure 3.19: The geometric property of α_3 and β_2. X, Y, Z are the principle axes of the coordinate frame $\mathbf{M}_1$.

of the neighbor frames.

3.6.1.1 Step 1

The step 1 is to find out the four pair of solutions of r and s. Figure 3.19 shows the geometric property of α_3 and β_2. The position of the node $\mathbf{D}_3^d$ is found by $\mathbf{M}_1$, α_3, r and s:

$$\mathbf{P}_3 \;=\; \mathrm{E}_4\left(\mathbf{M}_1 \mathrm{R}_{\hat{\mathbf{Y}}}\left(\alpha_3\right) \mathrm{T}_{\hat{\mathbf{Y}}}\left(r\right) \mathrm{T}_{\hat{\mathbf{X}}}\left(s\right)\right) \tag{3.41}$$

where $\mathbf{P}_3$ denotes the position of the node $\mathbf{D}_3^d$. The length d is found by the position of the nodes $\mathbf{D}_2^d$ and $\mathbf{D}_3^d$:

$$d \;=\; \left[\left(\mathbf{P}_3 - \mathbf{P}_2\right) \cdot \left(\mathbf{P}_3 - \mathbf{P}_2\right)\right]^{\frac{1}{2}} \tag{3.42}$$

β_2 is found by the position of the nodes $\mathbf{D}_1^d$, $\mathbf{D}_2^d$ and $\mathbf{D}_3^d$:

$$cos\left(\beta_2\right) \;=\; \left(\mathbf{P}_2 - \mathbf{P}_1\right) \cdot \left(\mathbf{P}_3 - \mathbf{P}_2\right) / \left\|\left(\mathbf{P}_2 - \mathbf{P}_1\right)\right\| \cdot \left\|\left(\mathbf{P}_3 - \mathbf{P}_2\right)\right\| \tag{3.43}$$

Then the above equations are generalized by substituting the node number with i, the equations become:

$$\mathbf{P}_i \;=\; \mathrm{E}_4\left(\mathbf{M}_1 \mathrm{R}_{\hat{\mathbf{Y}}}\left(\alpha_i\right) \mathrm{T}_{\hat{\mathbf{Y}}}\left(r\right) \mathrm{T}_{\hat{\mathbf{X}}}\left(s\right)\right) \tag{3.44}$$

$$d \;=\; \left[\left(\mathbf{P}_i - \mathbf{P}_{i-1}\right) \cdot \left(\mathbf{P}_i - \mathbf{P}_{i-1}\right)\right]^{\frac{1}{2}} \tag{3.45}$$

$$\cos\left(\beta_{i-1}\right) \;=\; \left(\mathbf{P}_{i-1} - \mathbf{P}_{i-2}\right) \cdot \left(\mathbf{P}_i - \mathbf{P}_{i-1}\right) / \left\|\left(\mathbf{P}_{i-1} - \mathbf{P}_{i-2}\right)\right\| \cdot \left\|\left(\mathbf{P}_i - \mathbf{P}_{i-1}\right)\right\| \tag{3.46}$$

In order to find r and s, we first substitute equation 3.44 into equation 3.45 and break down the rotation and translation into sin and cos operation. Then we extract r from the substituted equation, there will be two solutions for r, r_1 and r_2:

$$r_1 \;=\; \mathrm{E}^2\left(\mathbf{P}_{i-1}\right) + \sqrt{a+b} \tag{3.47}$$

$$r_2 \;=\; \mathrm{E}^2\left(\mathbf{P}_{i-1}\right) - \sqrt{a+b} \tag{3.48}$$

where

$$a \;=\; 2\,\cos\left(\alpha_i\right) s\mathrm{E}^1\left(\mathbf{P}_{i-1}\right) - \left[\mathrm{E}^1\left(\mathbf{P}_{i-1}\right)\right]^2 - \left[\mathrm{E}^3\left(\mathbf{P}_{i-1}\right)\right]^2 \tag{3.49}$$

$$b \;=\; -s^2 - 2\,\sin\left(\alpha_i\right) s\mathrm{E}^3\left(\mathbf{P}_{i-1}\right) + d^2 \tag{3.50}$$

We next substitute r_1 into equation 3.46 and extract s, and there will be two s, s_1 and s_2. Similarly, we substitute r_2 into equation 3.46 and extract s, and there will be two s, s_3 and s_4. The s solutions are very long which include more than hundreds of arithmetic operation; therefore, the s solutions are not listed here. After simplifying the s solutions, s_1 is equal to s_3, and s_2 is equal to s_4. The final step is to substitute the s solutions back into the r solutions to find out r_1 and r_2.

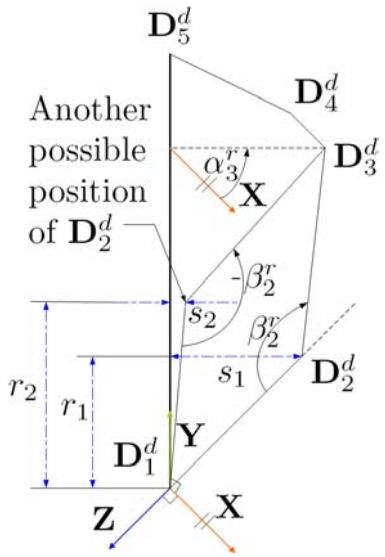


Figure 3.20: The two solution pairs $\{r_1,\ s_1\}$ and $\{r_2,\ s_2\}$. X, Y, Z are the principle axes of the coordinate frame $\mathbf{M}_1$.

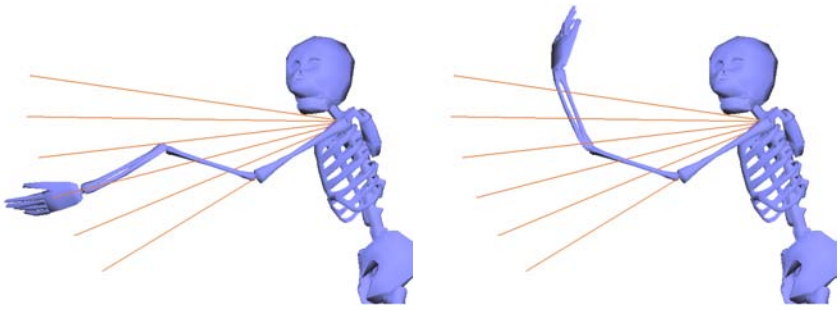


Figure 3.21: An example of two valid solutions. The right skeleton is selected as final solution.

3.6.1.2 Step 2

Step 1 gives out four solution pairs: $\{r_1,\ s_1\}$, $\{r_1,\ s_2\}$, $\{r_2,\ s_1\}$ and $\{r_2,\ s_2\}$ but not all solution pairs are valid. Figure 3.20 shows two solution pairs. From the figure, the valid solution pair should only be $\{r_1,\ s_1\}$, however, the solution pair $\{r_2,\ s_2\}$ is also valid because the magnitude of $-\beta_2^r$ is equal to β_2^r, although the sign is different. The valid solution pairs are then pruned to eliminate the false valid solution pair, $\{r_2,\ s_2\}$. The pruning is performed by substituting each solution pair into equation 3.46. The valid solution is the one which fulfils the equations. After the substitution, the final valid solution pair is found. Another illustration of two valid solutions is shown in figure 3.21.

However, if the problem described in section 3.5.9 occurs when performing the ICN method 3, we cannot prune the solution pairs because all the solution pairs will be regarded as invalid in the next step as the sign of β^r is changed. Therefore, pruning the solutions or not is controlled by the user.

3.6.1.3 Step 3

After the pruning, there are maximum 2 solution pairs left. However, if the pruning is disabled, maximum four solution pairs reach this step. The final solution is found by comparing the position of the nodes of the current frame with those of the neighbor frames. Here we make an assumption that the nodes in nearby frames should have a small position difference. This assumption applies if the motion is relatively smooth and has no sharp changes. The position difference is calculated as the summation of the absolute value of all the node position $\mathbf{P}_i^f$ in the frame f minus the node position $\mathbf{P}_i^{f-1}$ in the previous frame $f-1$. We compare the position in the previous frame only because the ICN is applied on the motion in ascending frame order. The position difference is written as:

$$d = \sum_{i=1}^{n} \left(\left| \mathbf{P}_i^f - \mathbf{P}_i^{f-1} \right| \right)$$

The final solution pair is that which has the smallest position difference.

3.7 Singularity issue

Singularity occurs in solving the inverse rate control equation when the segment chain involved in the retargeting is fully stretched. A similar problem also arises in calculating the control net angles and performing the inverse control net when the involved segment chain is fully stretched. Fortunately, solving the singularity in our method is simpler than solving the singularity of inverse rate control.

There are two types of singularity occurred in the calculation. The first type is the calculation of the cross product which is used as a rotation axis to rotate a skeleton node or a vector, for example, the calculation of the rotation axis in equation 3.1. When the angle between two vectors used to calculate the cross product is close to 0 degree of 180 degree, the cross product cannot be determined. The solutions for 0 degree or 180 degree are respectively: the node or the vector needs not to be rotated, the node or the vector translates in the opposite direction of its direction with the node's translation value or the vector's length. The second singularity type is the calculation of the cross product which is used as a reference vector to determine the sign of a scalar, for example, the sign determination in equation 3.4. We can simply ignore this type of singularity because the error in the sign determination is small when the scalar value approaching 0 or 180 degree.

3.8 Data Format

This section describes the data format that is used in the topological retargeting system which is the implementation of the topological retargeting method. First we present the data members that are used internally in the system, only the important data members are listed. Then we present the script file format which is used for controlling the MCN and the ICN.

3.8.1 Internal Data Format.

The topological retargeting system uses several classes and structures to encapsulate the data members and operations. The class MCN is used internally by the system to store the base nodes and the end nodes of both the source and the destination skeleton, and the linkages between the nodes of the skeletons. The information and the numbering of the source nodes are stored in an array. The source base node is the first element in the array, and the source end node is the last element in the array. The identifier number of each node can be find by traversing the array or by calling some array functions. The linkages between the skeletons are store by a map where the key of the map is the source node and the value is the destination node.

Class: MCN	Usage
Array: source node	store the information and the numbering of the source nodes
Map: map the source node to the destination node	store the linkage information

Each node, either a source node or a destination node, contains an array of node data. Each element of the array stores the information of a single frame in the motion.

Structure: Node	Usage
Array: node data	each array element stores the information of a single frame in the motion

The node data each contains three variables, they are the hierarchical data α, the segment-to-segment movement β and the orientation data γ.

Structure: Node data	Usage
float: α	store hierarchical data

float: β	store segment-to-segment movement
float: γ	store orientation data

The internal structure representing the ICN is the class ICN. The class ICN contains a invert function which performs the ICN. The ICN method 1 to the ICN method 3 are the subclass of the class ICN, and each subclass has their own overridden invert function.

Class: ICN	Usage
function: invert	perform the ICN

3.8.2 Script File Format.

The operations of the topological retargeting system are quite complex and the functionality of the system keeps changing; therefore, we use a script file to control the construction of the MCN and the running of the ICN. The modification to the MCN and the CNA is also stored in the script file. Table 3.7 lists a simple script which performs a topological retargeting that is similar to the one shown in figure 3.16. The script file is divided into sections, and the sections must be in order. Here is the explanation of each section:

- Line 1 begins the script file.
- Line 2 to line 5 open a motion file and give it a name "fighter".
- Line 6 to line 11 construct the source skeleton of the MCN and give it a name "fighter_left_hand" . The source base node is "LUpArm_A", and the source end node is "L_Palm". The "Create" keyword in line 10 tells the system to perform the operation that is described in the section, but not all sections need to have the keyword "Create". If a section must has the keyword "Create" but is missed, then the operation in that section will not be performed.
- Line 12 to line 14 create a synthesis node by copying half of the CNA from $\mathbf{D}_1^s$ and half of the CNA from $\mathbf{D}_2^s$.

- Line 15 to line 17 modify the CNA. The first word in line 16 is the name of the source node. The second word "A" tells the system to modify α. The other choices of the second word are: "B" tells the system to modify β and "G" tells the system to modify γ. The third word "1" is the number of the source node. The fourth word "113" is the frame going to be modified. The fifth word is the value which is superimposed on the data. The whole line means that 30 degree is superimposed on α_1 of the node $\mathbf{D}_1^s$ in frame 113. Each line in "BEditNet" section modifies the CNA in a particular frame and this frame is a keyframe for generating a curve of data for modifying other frames.

- Line 18 to line 21 open another motion and give it a name "target".

- Line 22 to line 35 control the ICN. Line 23 specifies the ICN method type, and the possible value are 1 to 3 which represents the ICN method 1, method 2 or method 3 respectively. Line 24 specifies the destination motion. Line 25 specifies the source motion. Line 26 names this ICN "mappingA".

  - Line 27 to line 33 specify the linkage between the source nodes and the destination nodes. For example, line 28 specifies the linkage for the first destination node. The left hand side of the line is the name of the destination node, and the right hand side of the line is the number of the source node.

- Line 36 ends the script file.

3.9 Data Flow

Our system can be roughly divided into three different modules. Data flows from one module to another module to perform different operations. The left hand side of figure 3.22 is the first module, which is the conversion code that converts the

original motion data to the CNA. This module is controlled by the “CreateNet” section inside the script file that is described in table 3.7. The middle part of figure 3.22 is the second module, which manipulates the MCN and the CNA. The CNA manipulations include blending, interpolation and various other editing. The second module is controlled by the “SynNet”, “BEditNet” section inside the script file. The right hand side of figure 3.22 is the third module, which performs the ICN. This module is controlled by the “RetargetByNet” section inside the script file.

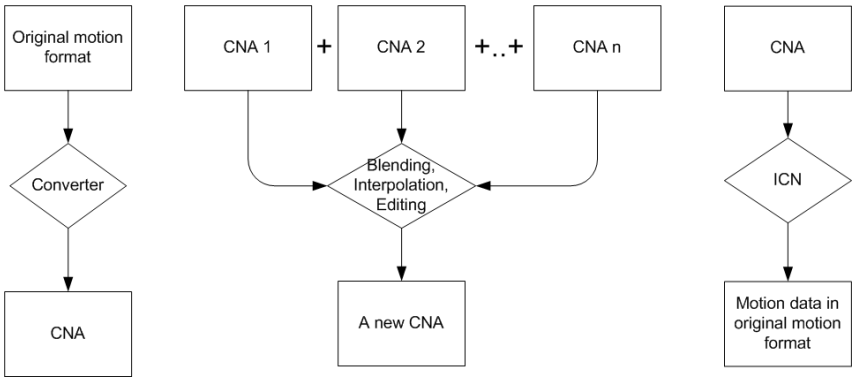


Figure 3.22: Data flow of the topological retargeting system.

3.10 Interface Design

The core part of the topological retargeting system is implemented by using C and C++. The interface part is implemented by using the GUI library wxWindows and OpenGL. We choose the wxWindows library mainly for the cross-platform compatibility. An alternative choice for writing cross-platform OpenGL program is GLUT, but GLUT does not have a good GUI system. Another reason of using wxWindows library is the license which is free for commercial and academic usage.

Figure 3.23 shows the main window of the topological retargeting system. The main window can be divided into six windows from I to VI. Window I is the main display of the system, contains a four view-port display and a set of tools on the bottom of the window controlling the view-port. Window II is mainly used for controlling the MCN, the ICN and the modification of the CNA. Window III contains

a set of tools to control an inverse kinematic algorithm. Window IV contains a set of buttons to control the view-port and to manipulate the skeleton and other objects which are displayed on the screen. Window V contains a set of tools which is used to modify the skeleton. Window VI displays various messages.

Certain steps of the ICN are not only controlled by the script, but also controlled by the GUI of the system. We do not use the script for performing all tasks because the script format and syntax are hard to remember than using the GUI. However, we cannot use the GUI for performing all tasks because this makes the GUI too complicated. A balanced implementation is to combine the script and the GUI to perform the tasks.

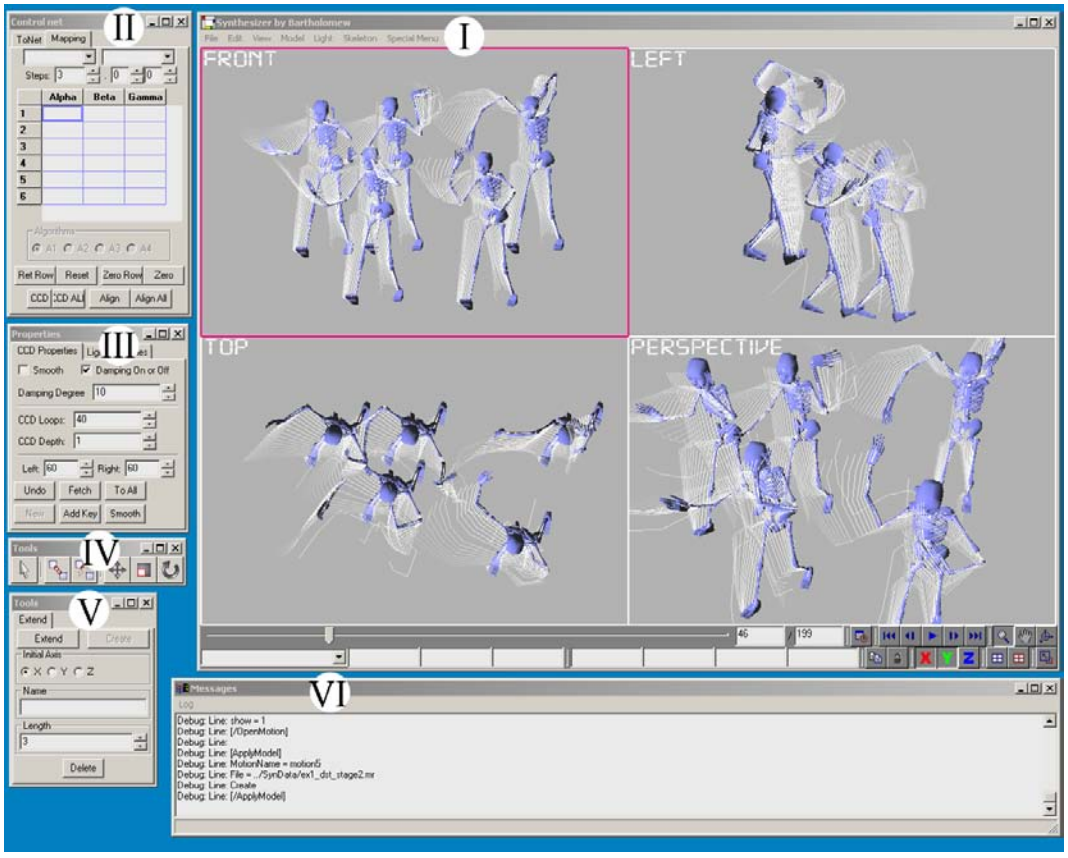


Figure 3.23: Interface of the topological retargeting system

```
[Script]
    [OpenMotion]
        file = ../SynData/both_src.gfa
        name = fighter
    [/OpenMotion]
    [CreateNet]
        base = LUpArm_A
        end = L_Palm
        name = fighter_left_hand
        Create
    [/CreateNet]
    [SynNet]
        new = fighter_left_hand 1 * 0.5 + fighter_left_hand 2 * 0.5
    [/SynNet]
    [BEditNet]
        fighter_left_hand    A    1    113    30
    [/BEditNet]
    [OpenMotion]
        file = ../SynData/both_dst.gfa
        name = target
    [/OpenMotion]
    [RetargetByNet]
        RetargetType = 3
        TargetMotionName = target
        SourceNet = fighter_left_hand
        name = mappingA
        [RetargetNetMapping]
            LUpArm_A = 0
            LLowArm_A = 1
            LHand_A = 4
            L_Palm = 2
            LFinger = 3
        [/RetargetNetMapping]
        Create
    [/RetargetByNet]
[/Script]
```

Table 3.7: A simple script controls the construction of the MCN and the running of the ICN.

Chapter 4

Results

This section describes the results of four experiments performed by the topological retargeting system. The first experiment shows a topological retargeting of a sword-swinging sequence. In order to illustrate the features of the MCN, we show two stages of this experiment. In the first stage, the motion is retargeted to a new skeleton having an additional segment added in the forearm. The first stage also demonstrates interactive editing through the modification of a set of key frames in the MCN. In the second stage, the structure of the forearm is changed and positional constraints are imposed on the retargeted skeleton. In the second experiment, we retarget a dancing motion to a skeleton having a modified leg. In the third experiment, we demonstrate another ability of the MCN which generates new motion based on the motion of an internal node or an external node. In the fourth example, we retarget a human walking motion to a spider skeleton.

The topological retargeting examples are created by using only the ICN method 2; however, there is no example of retargeting which is performed by method 1 and method 2 because method 1 is an algorithm which is used to test the feasibility of the conversion, and method 3 is a preliminary algorithm for an automatic sensitivity problem solver. In addition, method 1 and method 3 perform the same function as method 2 does, the differences are the accuracy and the execution speed. Therefore, only the examples which are created by using method 2 is shown in this section.

After presenting the results, we present the performance comparison of the three ICN algorithms.

4.1 Retargeting of a Sword-Swinging Sequence

Figure 4.2 illustrates an experiment in which five skeleton figures are engaged in a retargeted sword-swinging motion. The experiment is in two stages. The leftmost skeleton I is the original, skeletons II, III, IV are the stage 1 results, and the rightmost skeleton V is the stage 2 result.

In the first stage of retargeting, a segment has been added between the elbow and the wrist on both hands. The movement of the additional segment is generated by combining the CNA of the elbow and the wrist. At this stage, there is no other modification made on the CNA. Skeleton II shows the result of this stage. In order to illustrate the differences between the source and the retargeted skeleton, the source motion, skeleton I is provided as a reference.

Interactive editing is made possible by the dynamical adjustment of the CNA through a user interface in the retargeting system. An animator adjusts the CNA and views the result through the system interface. Since a single frame ICN operation requires only few milliseconds, an animator receives the result immediately and can immediately move to the next editing decision. After the editing, the modified frames act as key frames and the data between them are generated by a cubic curve. Skeletons III and IV show the result of interactive editing. In these two motion, we modify the values of the CNA of all nodes. The actual value of the CNA of the additional segment is shown in figure 4.1.

In the second stage, the motion is retargeted to a skeleton with a different topology and under a positional constraint. We modify the number of segments, their lengths, and trajectories of the skeleton. We reduce the length of hand segments so that they are slightly shorter than the original hand because the additional segments lengthen the hand, and change the trajectory of the elbow so that the elbow is al-

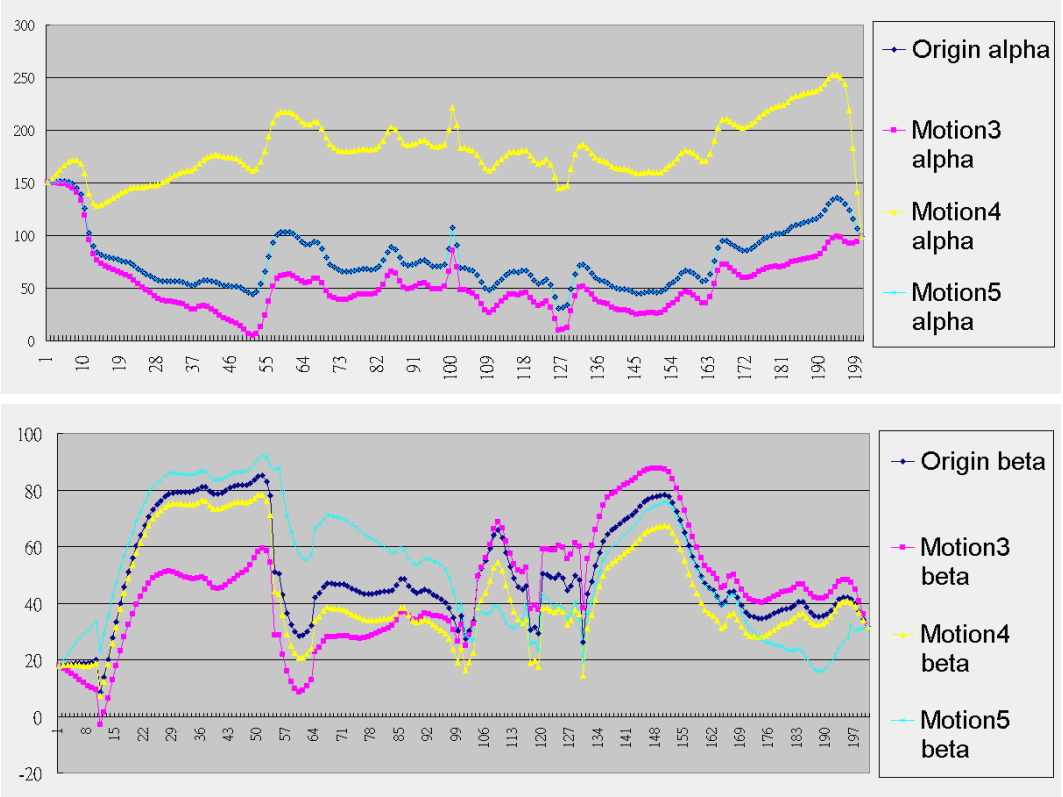


Figure 4.1: α and β of the additional segment. α and β of the skeleton III to V are generated by displacing a curve on the original data.

ways higher than the head. Trajectory of the elbow is changed using the first MCN, modifying the shoulder node. The arm motion is then modified using the second MCN, altering the upper arm and lower arm. Two MCNs are used for a single limb because this makes it easier to carry out the desired modification. The positional constraint is that the end node of the new hand must follow the end node of the original motion. Since adding a new segment to the forearm changes the reachable range of the palm, we adjust the end node position by modifying the β parameter. Finer adjustment of the end node is done through an IK algorithm. Skeleton V shows the result of this stage. The actual values of the CNA of the additional segment are shown in figure 4.1. The hierarchical data of the additional segment of skeleton V is the same as in skeleton I, while the segment-to-segment movement of the additional segment is modified.

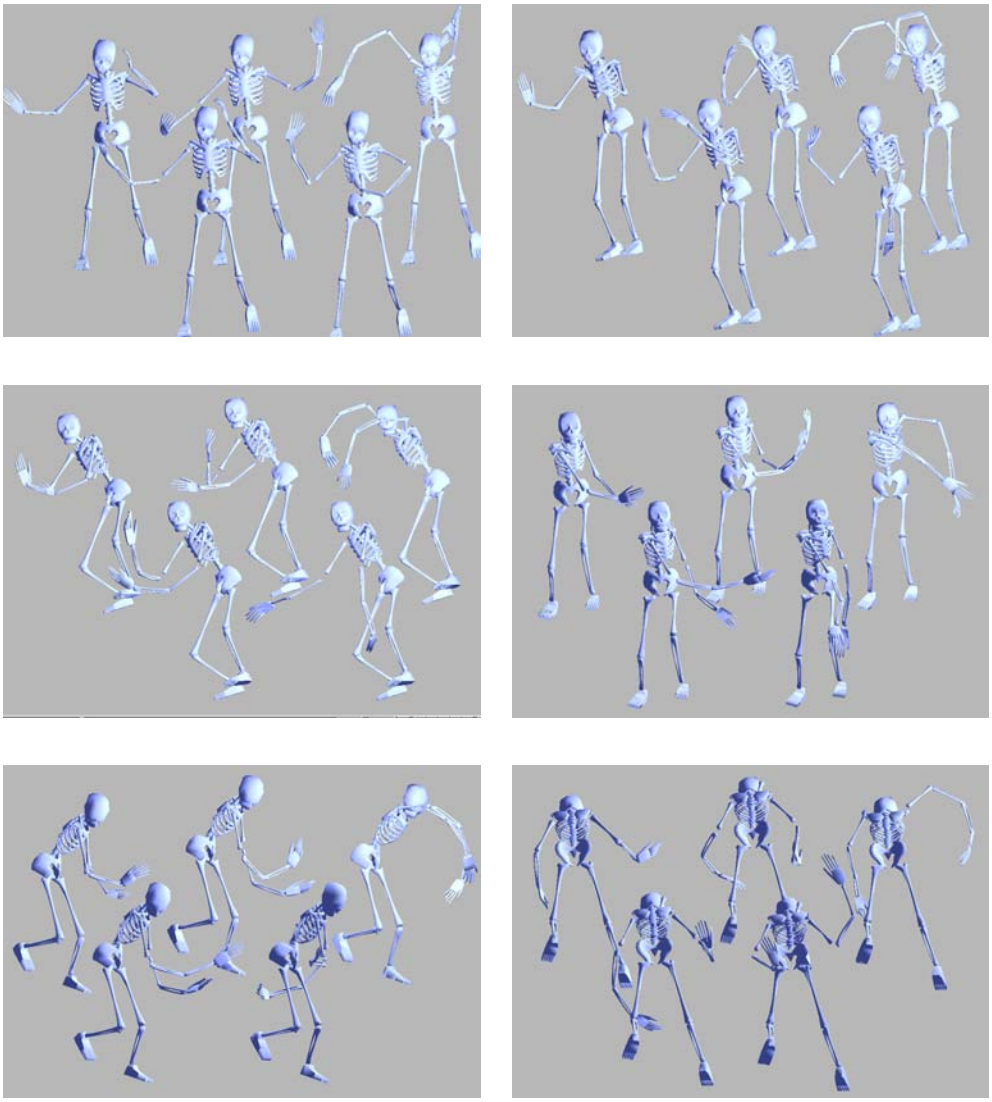


Figure 4.2: Retargeting of a sword-swinging motion. The leftmost skeleton I is the source motion. The second skeleton II from left is a simple retargeted motion. The arm movement of the third and fourth motion is created by shifting the CNA by a curve. The arms of the rightmost skeleton V are synthesized by using two MCNs for each arm.

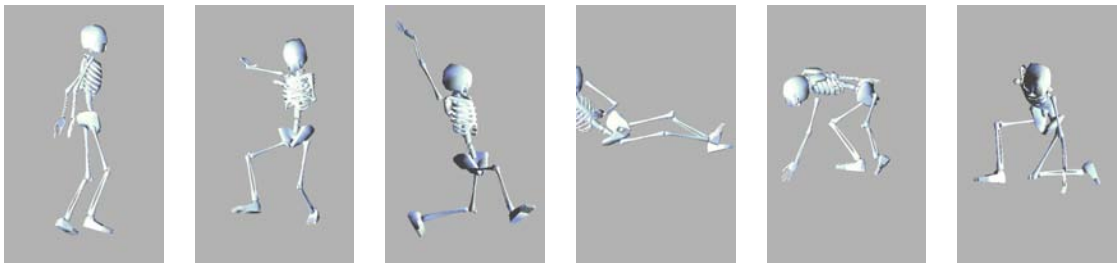


Figure 4.3: Source dancing motion.

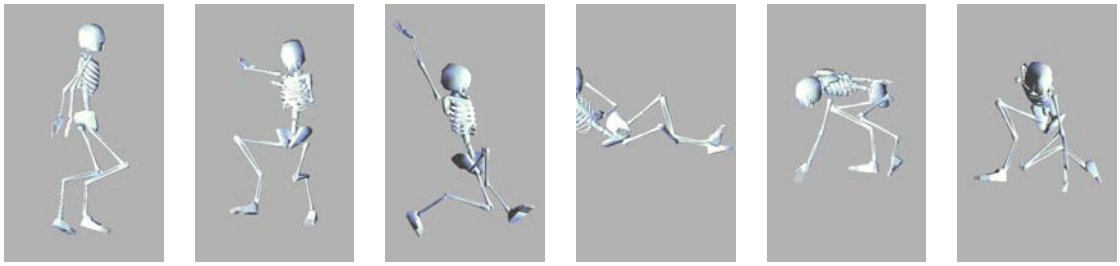


Figure 4.4: Retargeted dancing motion having an additional segment in the lower leg.

4.2 Retargeting of Dancing Sequence A

In this experiment, we retarget a dancing motion having a topological change on the leg. We change the topology of the leg by adding a new segment to the lower leg, and the movement of the new segment is copied from the original lower leg. In order to fit the new segment to the skeleton, the CNA of the original segments are changed slightly and the α parameter of the new segment is shifted by a π degree in certain frames, while the β parameter of the new segment is always changed to a negative value. The shifting and the negating of the CNA allow the new segment to look like an inversion of the original lower leg. Figure 4.3 shows the original dancing chips. Figure 4.4 shows the retargeted dance.

4.3 Retargeting of Dancing Sequence B

This experiment demonstrates the usage of the MCN to synthesize new motion. The motion contains a synthesized arm and a synthesized leg. We synthesize the

arm motion by blending the CNA of different limbs. The hierarchical data of the new arm is mainly copied from the original arm, with a portion from the leg. The segment-to-segment movement of the new arm is mainly copied from leg. Motion blending is performed on the leg. The hierarchical data of the leg is copied from the original leg. The segment-to-segment movement of the leg comes from a crab leg motion which is produced by keyframing. This setting demonstrates that the MCN can generate new motion that is affected by either an internal or external motion. This is useful in synthesizing a group of characters with similar motion. Figure 4.5 shows the original dancing clips. Figure 4.6 shows the retargeted dancing clips.

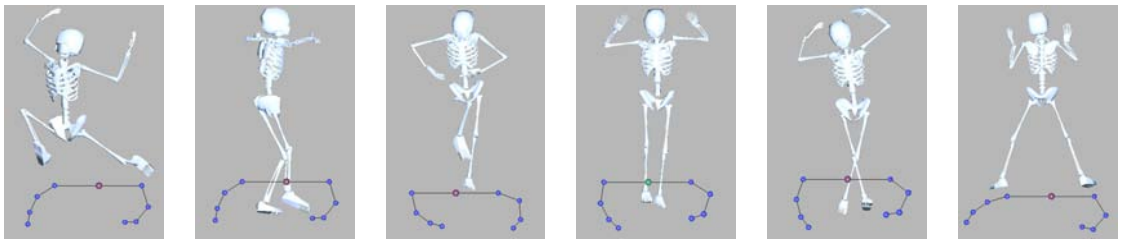


Figure 4.5: Source motion of dancing sequence B. The stick figure is a pair of crab legs which is produced by keyframing.

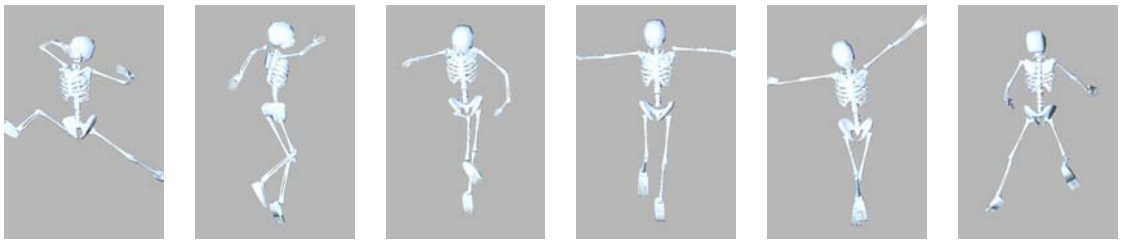


Figure 4.6: The arms are synthesized by blending different limbs of the source motion. The movement of legs is created by combining the α of the original leg and the β of the crab leg.

4.4 Retargeting Human Motion to a Spider Skeleton

This experiment demonstrates the example of retargeting a human motion to a 6 legs spider skeleton, showing that the MCN can perform retargeting of a skeleton to

another skeleton with big difference in the topology. Each spider leg is synthesised by blending the CNA of the human arm and leg. The hierarchical data of the spider leg is copied from the human arm, creating the forward crawling part of the spider leg movement. The segment-to-segment movement of the spider leg is copied from the human leg, creating the stretching part of the spider leg movement. As the spider contains 2 more pair of legs than a human, the motion data of the extra legs is copied from different time frame of the same human motion.

Figure 4.7: The source motion, a human walking sequence.

Figure 4.8: The retargeted spider motion. Each leg motion is created by blending the motion of human arm and leg in the MCN domain.

4.5 Manipulation details

In this section, we will describe the details of the manipulation of the MCN using the example of sword-swinging sequence stage 2 and dancing sequence B.

In the sword-swinging sequence stage 2 example, there is an additional node inserted in the forearm, and the data of the new node comes from its neighbor nodes, the original elbow $\mathbf{D}_2^s$ and wrist $\mathbf{D}_3^s$. The blending equations of the CNA of the additional node, $\mathbf{D}_3^d$ are: $\alpha_3^d = 0.5 * \alpha_2^s + 0.5 * \alpha_3^s$, $\beta_3^d = 0.5 * \beta_2^s + 0.5 * \beta_3^s$ and $\gamma_3^d = 0.5 * \gamma_2^s + 0.5 * \gamma_3^s$, where the superscript "s" of the CNA indicates the CNA from the source node and "d" means the CNA of the destination node.

After combining the CNA, some frames of the resulting motion may not satisfy the requirement and need to be adjusted. There are two modes of editing provided by our implementation. The first one is modifying a particular frame in the motion, the second one is displacing a curve on the CNA data. Modifying a particular frame is achieved by adding a line in the "[EditNet]" section in the script file, for example, a sample line: "fighter_left_hand BB 2 101 22.39". The line has the following meaning: the first token "fighter_left_hand" is the name of the MCN, the second token "BB" means the absolute β parameter (the value of this line replaces the original value of the node), the third token "2" means the node which is working on is the destination node $\mathbf{D}_3^d$ (the system is zero based, so 2 means the third node), the forth token is the frame number, and the fifth token is the new value of the node. The second mode is displacing a curve on the data, for example, the trajectory of the elbow can be modified by this mode. The curve is specified by a series of line which acts as control point of a B-Spline curve in the "[BEditNet]" section in the script file. A sample line is "fighter_left_hand B 3 18 8.97". The line has similar meaning as the line in the "[EditNet]" section except the second token. The second token "B" means the β parameter is relative, and the value "8.97" is added to the original value of the node. The result of adding a curve to the CNA is illustrated by the figure 4.1.

The advantage of adding a curve to the motion is that the motion characteristic can be preserved in the high frequency content of the curve and having the motion modified by the low frequency curve, the B-Spline curve with sparse control point. This approach is similar to the method proposed by [12].

In the dancing sequence B example, the elbow and wrist motion of the right arm is synthesised by blending the CNA of different limbs. The blending equations of the elbow, $\mathbf{D}_2^d$ are: $\alpha_2^d = 0.5 * \alpha_2^s$ (from right hand) $+ 0.5 * \alpha_2^s$ (from left hand), $\beta_2^d = \beta_2^s$ (from right foot) and $\gamma_2^d = \gamma_2^s$ (from right hand). The other modification of the CNA is similar to the sword-swinging sequence stage 2 example and is skipped here.

4.6 Performance Comparison

We compare the performance using the example of sword-swinging sequence stage 1 from section 4.1. In the example, the motion sequence contains 200 frames, and the number of nodes of each arm involved in the retargeting is 5, hence n is being equal to 5. ICN method 1 is a recursive algorithm, the complexity closes to O (n^n). The accuracy of retargeting the left hand is 83%, and the accuracy of retargeting the right hand is 97%. The total running time for both hands is 4.2 seconds. ICN method 2 is a iterative algorithm, the complexity is to O $(n\log n)$. The accuracy of retargeting the left hand is 77%, and the accuracy of retargeting the right hand is 83%. The total running time for both hands is 2.2 seconds. ICN method 3 is an analytical algorithm, the complexity is to O (n). The accuracy of retargeting the left hand is 79%, and the accuracy of retargeting the right hand is 88%. The total running time for both hands is 1.8 seconds.

The running time of method 1 is about the twice of method 2. The running time is faster than expected, because the number of maximum looping in each iteration is limited as listed in algorithm 1 line 2, although method 1 is a recursive algorithm. The accuracy of method 1 is higher than other two because the recursive nature lets the algorithm considers more alternative solutions (nearly all possible solutions) and finds out the correct solution from nearest solutions. The accuracy of method 2 and the accuracy of method 3 is closed together because method 2 can partially solve the sensitivity problem, and method 3 can solve sign changing problem; therefore, the accuracy of these two methods are closed together.

Chapter 5

Discussion

In this chapter, we first compare the functionality of our method with other methods, then discuss the future extension of our method.

5.1 Comparison

In the sword-swinging retargeting experiment presented in section 4.1, the MCN facilitates a number of functions beyond simple motion interpolation. Firstly, MCN allows the displacement of the CNA of the synthesized node with a curve to generate a new locus having a similar movement pattern but distinct motion. Secondly, the fact that the MCN can separate the hierarchical data from the motion means that we can create new motion from different source motion. For example, the hierarchical data of motion A and the segment-to-segment movement of motion B can be combined into new motion which is similar to motion A yet retains certain desired characteristics of motion B. This functionality is illustrated in the dancing sequence retargeting presented in section 4.3, which combines the motion of a crab leg and the motion of a human leg into the motion of the destination skeleton. If this operation is done using interpolation, the resulting motion may look quite different from the original motion.

Inverse kinematics, inverse rate control, jacobian transpose method and CCD

work directly on the end-effector, and the end-effector position suffices for the purpose of retargeting. The drawback of these methods is that they exercise only limited or no control over the nodes other than the end-effector. The MCN offers the advantage of controlling the nodes other than the end-effector. One disadvantage of inverse kinematics and inverse rate control is that, in the presence of a redundant degree of freedom, they give multiple solutions, and the final solution is selected by the user or by some predetermined rules. The MCN answers this by producing at most one solution in the redundant case. The dancing sequence retargeting in section 4.2 demonstrates this feature of which the end-effector follows the original end-effector and the redundant segments are well controlled. In the presence of singularity, MCN has less problem in handling the singularity, while special measure must be taken in inverse kinematics.

Spacetime method considers the whole sequence of motion in the retargeting process; therefore, the execution speed is relatively slow and may not be able to provide real time performance. In contrast, MCN considers one neighbor frame in the retargeting process and the ICN can be switched to using an analytical algorithm, execution speed is faster and can be performed in a real time manner. The advantage of spacetime method is that the generated motion is smooth. The smoothness is also ensured using MCN. Actually, the smoothness of the generated motion by MCN depends on the CNA data, not the method itself. The major drawback of spacetime method is that it is sensitive to the skeleton initial configuration. In contrast, MCN does not has the initial configuration problem.

Intermediate skeleton can solve a part of the topological retargeting problem as this method can work on altered number of nodes in the skeleton. However, Intermediate skeleton can only work for similar source and destination skeleton. If there are large differences in the skeletons, the method produces strange result. In contrast, MCN works for not just altered number of nodes in the skeleton, and MCN works for skeletons with big differences in hierarchy. Moreover, MCN can combine different motions from different skeletons and can re-order the nodes in

the skeleton, but the intermediate skeleton cannot.

5.2 Future Extension

A possible future extension to our topological retargeting method is the extension to the ICN in the presence of the sensitivity problem which is presented in section 3.5.8. Since the problem is mainly the bad continuity between the frames of the retargeted motion, a possible direction to solve this problem is using an optimization algorithm to minimize the discontinuity between the frames. The idea of solving this problem comes from spacetime method which considers all frames in the motion rather than an individual frame.

A possible extension is to extend the ICN method 2 to an automatic sensitivity problem solver. There are, however, some problems exist in converting the equations of method 2 to be used in the optimization algorithm. The first problem is the highly non-linear property of the equations. The equations in method 2 are multi-variable functions, and some variables depend on other variables and depend on several equations. For example, α_i depends on α_{i-1} and depends on several equations. The dependency combining with the iterative nature of method 2 is the second problem which makes the equations not suitable to be used in the optimization algorithm.

The ICN method 3 is developed in a direction that the equations of the method are suitable to be used in the optimization algorithm. The equations in method 3 are analytical formulations, finding a solution in a single iteration. However, the current formulation of method 3 still contains the variable dependency. A possible modification to the formulations is to remove the dependency between the variables, and this modification will turn the exact solution in method 3 into an approximation solution of which an error function is required to measure the error of the approximation; however, such formulations have not been developed yet.

Chapter 6

Conclusion

This thesis presents a new topological retargeting method that enlarges the domain of the original retargeting problem aimed at characters with *dimension variations* to characters with *topological variations*. We have developed a new data representation named motion control net (MCN) in order to achieve a higher level of abstraction and separation of the motion data from the skeleton hierarchy in captured data. Hierarchical data and segment movement are stored separately in different datasets known as control net angles (CNA) inside MCN. In this way, MCN can encapsulate the different motion characteristics of the character effectively without a strong dependency on the original skeleton topology.

The flow of our method is first converting the original motion data to one or more MCNs, then making modification in the MCN domain, and finally converting the MCN(s) back to the original motion data format through the inverse control net (ICN) process. Topological retargeting can be performed by modifying, blending and/or cloning the CNA in the ICN stage or by adjusting the structure of the MCN itself.

The method described in the thesis considers only the direction of the end-effector but not the position. This is usually sufficient to generate the desired motion if the exact positioning of the end-effector is not a constraint. If more precise position adjustment is required, an IK algorithm can be applied to the end-effector to

perform further adjustment of the position.

One possible enhancement of our method is to develop an automatic sensitivity problem solver. In our implementation of the MCN, even the value of the hierarchical data in CNA is outside certain range, ICN will always give the solution with smallest error. The solution, however, will have a bad continuity with its neighbor frames in the restored motion. We have outlined the direction to tackle this problem in this thesis; however, an automatic sensitivity problem solver have not been developed yet. This problem solver is suggested as a future extension to the MCN.

Bibliography

[1] N. Badler, M. J. Hollick, and J. P. Granieri. Real-time control of a virtual human using mininal sensors. *Presence 2*, 1, 1993.

[2] Christoph Bregler, Lorie Loeb, Erika Chuang, and Hrishi Deshpande. Turning to the masters: motion capturing cartoons. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 399–407. ACM Press, 2002.

[3] Armin Bruderlin and Lance Williams. Motion signal processing. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 97–104. ACM Press, 1995.

[4] Gordon Cameron, Andre Bustanoby, Ken Cope, Steph Greenberg, Craig Hayes, and Olivier Ozoux. Motion capture and cg character animation (panel). In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 442–445. ACM Press/Addison-Wesley Publishing Co., 1997.

[5] Kwan W. Chin. Closed-form and generalized inverse kinematic solutions for animating the human articulated structure. Master's thesis, Curtin University of Technology, 1996.

[6] Kwang-Jin Choi and Hyeong-Seok Ko. On-line motion retargetting. In *Proceedings. Seventh Pacific Conference*, pages 32–42, 1999. TY - CONF.

[7] Min Gyu Choi, Jehee Lee, and Sung Yong Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics (TOG)*, 22(2):182–203, 2003.

[8] Michael F. Cohen. Interactive spacetime control for animation. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 293–302. ACM Press, 1992.

[9] John J. Craig. *Introduction to Robotics Mechanics and Control*. Addison-Wesley publishing company, 1986.

[10] Mira Dontcheva, Gary Yngve, and Zoran Popović. Layered acting for character animation. *ACM Trans. Graph.*, 22(3):409–416, 2003.

[11] Michael Gleicher. Motion editing with spacetime constraints. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 139–ff. ACM Press, 1997.

[12] Michael Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 33–42. ACM Press, 1998.

[13] Michael Gleicher. Animation from observation: Motion capture and motion editing. *ACM SIGGRAPH Computer Graphics*, 33(4):51–54, 2000.

[14] Michael Gleicher. Motion path editing. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 195–202. ACM Press, 2001.

[15] Akanksha Huang, Z. Huang, B. Prabhakaran, and Jr. C. R. Ruiz. Interactive visual method for motion and model reuse. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Austalasia and South East Asia*, pages 29–ff, Melbourne, Australia, 2003. ACM Press.

[16] Akanksha Huang, Z. Huang, B. Prabhakaran, and Jr. C. R. Ruiz. Interactive visual method for motion and model reuse. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Austalasia and South East Asia*, pages 29–ff. ACM Press, 2003.

[17] Lucas Kovar and Michael Gleicher. Flexible automatic motion blending with registration curves. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 214–224. Eurographics Association, 2003.

[18] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 473–482. ACM Press, 2002.

[19] Lucas Kovar, John Schreiner, and Michael Gleicher. Footskate cleanup for motion capture editing. In *Proceedings of the ACM SIGGRAPH symposium on Computer animation*, pages 97–104. ACM Press, 2002.

[20] Jeff Lander. Making kine more flexible. *Game Developer*, pages 15–22, 1998.

[21] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 39–48. ACM Press/Addison-Wesley Publishing Co., 1999.

[22] Yan Li, Tianshu Wang, and Heung-Yeung Shum. Motion texture: a two-level statistical model for character motion synthesis. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 465–472. ACM Press, 2002.

[23] Peter C. Litwinowicz. Inkwell: A 2-d animation system. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 113–122. ACM Press, 1991.

[24] C. Karen Liu and Zoran Popović. Synthesis of complex dynamic character motion from simple animations. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 408–416. ACM Press, 2002.

[25] Zicheng Liu, Steven J. Gortler, and Michael F. Cohen. Hierarchical spacetime control. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 35–42. ACM Press, 1994.

[26] David G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Pub Co, June 1984.

[27] Anthony A. Maciejewski. Motion simulation: Dealing with the ill-conditioned equations of motion for articulated figures. *IEEE Comput. Graph. Appl.*, 10(3):63–71, 1990.

[28] Jean-Sébastien Monzani, Paolo Baerlocher, Ronan Boulic, and Daniel Thalmann. Using an intermediate skeleton and inverse kinematics for motion retargeting. *Computer Graphics Forum*, 19, 2003.

[29] Michael Neff and Eugene Fiume. Aesthetic edits for character animation. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 239–244. Eurographics Association, 2003.

[30] Sang Il Park, Hyun Joon Shin, and Sung Yong Shin. On-line locomotion generation based on motion blending. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 105–111. ACM Press, 2002.

[31] Julien Pettré, Jean-Paul Laumond, and Thierry Siméon. A 2-stages locomotion planner for digital actors. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 258–264. Eurographics Association, 2003.

[32] Zoran Popović and Andrew Witkin. Physically based motion transformation. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 11–20. ACM Press/Addison-Wesley Publishing Co., 1999.

[33] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical recipes in C : the art of scientific computing*. Cambridge University Press, 1992.

[34] Katherine Pullen and Christoph Bregler. Motion capture assisted animation: texturing and synthesis. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 501–508. ACM Press, 2002.

[35] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 147–154. ACM Press, 1996.

[36] Hyun Joon Shin, Jehee Lee, Sung Yong Shin, and Michael Gleicher. Computer puppetry: An importance-based approach. *ACM Transactions on Graphics (TOG)*, 20(2):67–94, 2001.

[37] Maryann Simmons, Jane Wilhelms, and Allen Van Gelder. Model-based reconstruction for creature animation. In *Proceedings of the ACM SIGGRAPH symposium on Computer animation*, pages 139–146. ACM Press, 2002.

[38] Seyoon Tak, Oh young Song, and Hyeong-Seok Ko. Spacetime sweeping: an interactive dynamic constraints solver. *Computer Animation, 2002. Proceedings of*, 2002.

[39] L.-C.T. Wang and C.C. Chen. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *Robotics*

and Automation, IEEE Transactions on, 7(1042-296X):489–499, 1991. TY - JOUR.

[40] Chris Welman. Inverse kinematics and geometric constraints for articulated figure manipulations. Master's thesis, Simon Fraser University, 1993.

[41] Andrew Witkin and Michael Kass. Spacetime constraints. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 159–168. ACM Press, 1988.

[42] Andrew Witkin and Zoran Popović. Motion warping. *Computer Graphics*, 29(Annual Conference Series):105–108, 1995.

[43] Victor B. Zordan and Nicholas C. Van Der Horst. Mapping optical motion capture data to skeletal motion using a physical model. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 245–250. Eurographics Association, 2003.