การประมาณตำแหน่งสามมิติจากภาพเคลื่อนไหวของมือ
โดยใช้แบบจำลองการฉายแนวตั้งฉากตามมาตราส่วน

นาย โฆสิต นพวิชัย

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2552

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

3D COORDINATE ESTIMATION FOR HAND MOTION IMAGE
USING SCALED ORTHOGRAPHIC PROJECTION MODEL

Mr.Kosit Nopvichai

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Computer Science
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2009
Copyright of Chulalongkorn University

Thesis Title	3D Coordinate Estimation for Hand Motion Image Using Scaled Orthographic Projection Model
By	Mr. Kosit Nopvichai
Field of Study	Computer Science
Thesis Advisor	Assistant Professor Pizzanu Kanongchaiyos, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

.... Dean of the Faculty of Engineering

(Associate Professor Boonsom Lerdhirunwong, Dr.Ing.)

THESIS COMMITTEE

.... Chairman

(Assistant Professor Athasit Surarerks, Ph. D.)

.... Thesis Advisor

(Assistant Professor Pizzanu Kanongchaiyos, Ph.D.)

.... External Examiner

(Chakrit Watcharopas, Ph.D.)

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

โฆสิต นพวิชัย: การประมาณตำแหน่งสามมิติจากภาพเคลื่อนไหวของมือโดยใช้แบบจำลองการฉายแนวตั้งฉากตามมาตราส่วน. (3D COORDINATE ESTIMATION FOR HAND MOTION IMAGE USING SCALED ORTHOGRAPHIC PROJECTION MODEL) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ.ดร.พิษณุ คนองชัยยศ, 83 หน้า.

ภาพเคลื่อนไหวสองมิติมีศักยภาพที่จะถูกใช้เป็นแหล่งข้อมูลในการแอนิเมทโมเดลสามมิติได้ แต่ปัญหาสำคัญในการกระทำดังกล่าวคือการหาวิธีที่มีประสิทธิภาพในการคำนวณหาข้อมูลมิติที่สามจากภาพสองมิติ

ในปัจจุบัน หลายวิธีการได้ถูกเสนอขึ้นมาแต่เนื่องจากวิธีเหล่านี้ส่วนใหญ่จะใช้สูตรทางคณิตศาสตร์ที่ซับซ้อนจึงมีผลให้การทำงานล่าช้า นอกจากนี้ปัญหาเกี่ยวกับความคลุมเครือแบบสะท้อนกลับก็จะต้องถูกนำมาพิจารณาด้วยเพื่อให้ผลที่ได้มีความถูกต้องมากขึ้น

วิทยานิพนธ์ฉบับนี้เสนอเทคนิคการประมาณตำแหน่งสามมิติจากภาพเคลื่อนไหวของมือโดยใช้วิธีแบบจำลองการฉายแนวตั้งฉากตามมาตราส่วนในการคำนวณหาค่าพิกัดของมิติที่สาม ปัญหาการถูกบดบังหรือการขาดหายของข้อมูล ได้ถูกแก้โดยการพิจารณาใช้ข้อมูลจากเฟรมก่อนหน้าและความเกี่ยวเนื่องกันของการเคลื่อนไหวของข้อนิ้ว นอกจากนี้ ปัญหาความคลุมเครือแบบสะท้อนกลับก็ถูกแก้โดยการกำหนดข้อจำกัดที่ข้อนิ้ว

ในการทดลอง เราใช้ข้อมูลสองมิติจากมายาแอนิเมชั่นของการกำมือเป็นข้อมูลเข้าของโปรแกรมที่ใช้ในการคำนวณหาข้อมูลที่ขาดหายและข้อมูลมิติที่สามตามลำดับ จากนั้น เรานำข้อมูลมิติที่สามที่คำนวณได้มาวิเคราะห์โดยการเปรียบเทียบผลที่ได้จากการทดลองกับข้อมูลจริงที่ได้จากมายาแอนิเมชั่น ผลการวิเคราะห์แสดงให้เห็นว่าวิธีการที่เราใช้สามารถคำนวณหาค่าพิกัดของมิติที่สามได้แม่นยำและสามารถแก้ปัญหาความคลุมเครือแบบสะท้อนกลับได้ดีในหลายกรณี

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

ภาควิชา....วิศวกรรมคอมพิวเตอร์.... ลายมือชื่อนิสิต....โฆสิต นพวิชัย....
สาขาวิชา....วิทยาศาสตร์คอมพิวเตอร์.... ลายมือชื่ออ.ที่ปรึกษาวิทยานิพนธ์หลัก........
ปีการศึกษา....2552....

4971409721 : MAJOR COMPUTER SCIENCE
KEYWORDS: 3D COORDINATE ESTIMATION / HAND MOTION

KOSIT NOPVICHAI: 3D COORDINATE ESTIMATION FOR HAND MOTION IMAGE USING SCALED ORTHOGRAPHIC PROJECTION MODEL. THESIS ADVISOR: ASST. PROF PIZZANU KANONGCHAIYOS, Ph.D., 83 pp.

A 2D image sequence can be a great source of motions for animating a 3D model. However, depth information cannot simply be extracted from a two-dimension image. Thus, there is a need for a method to obtain this third dimension data.

Several methods have been presented over the years. But most of them employed a complex mathematical concept which makes it unavoidably slow. Moreover, there is an inherent problem of reflective ambiguity which must be addressed.

In this study, we present a technique to perform a 3D coordinate estimation of 2D hand motion from an image sequence. In our method, the orthographic projection model is used to determine the Z coordination. Additionally, information from the previous frames and interdependence of a hand model are used to handle occlusion. We also propose a set of constraints on the finger joints in order to deal with reflective ambiguity.

In our experiment, XY coordinates of a set of feature points are extracted from a Maya animated Hand Clinching motion. The missing data and depth information are then calculated. Finally the resulting Z coordinates are evaluated by comparing with the actual Z values from the Maya animation. The result shows that our method can estimate the Z coordinated quite well and can correctly solve the reflective ambiguity in most common cases.

Department : .....Computer Engineering.....Student's Signature....Kosit Nopvichai....
Field of Study : ....Computer Science..........Advisor's Signature....[signature]....
Academic Year : ....2009..................

Acknowledgements

I owe a depth of gratitude to Dr. Pizzanu Kanongchaiyos for his help, advice and understandings. I would like to thank all the teachers that I have a privilege to be their student. I also sincerely appreciate the help of the staff members at the Computer Engineering Department for their assistance in many occasions.

Lastly, I would like to thank my family for their tireless love and support and my friends for their great suggestions and supports.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Contents

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

List of Tables

List of Figures

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Chapter 1

Introduction

1.1 Background and Statement of Problems

Computer animation is the science and art of using a computer to create moving images. The idea is to make a character move in a way intended by the artists and convey their creativity to the audience. There are several ways to generate motions for an articulated character. Some of the more common are Kinematics, Dynamic control [1], Keyframing, Motion editing [2] [3] [4] [5] [6], and Motion capture. Recently more attention has been paid to an alternative to the traditional methods. It is the typical 2D video that is recorded by a typical camera or even a web cam.

There are certain advantages to this motion source. First, the source model does not need to be attached with sensors. Second, the cost is typically lower than the traditional motion capture. Third, there are enormous stocks of live action footage recorded as 2D videos. Some of them are of historic values and cannot be reproduced. An example is a number of classic sport moments. This can be readily used as a motion source.

Using 2D image sequence as a source of motions has a few challenges of its own that need to be addressed. First, the missing data (e.g. those caused by occlusion) need to be somehow recovered. Specific to hand motions, we may consider using interdependence in addition to constraints, motion library, sample space, etc. Second, the 2D nature of it necessitates the lack of depth information. Thus some variants of 3D reconstruction techniques are used to recover the missing Z coordinate. We will address these issues in our work.

After a motion is acquired through one of the means mentioned above and stored in a motion representation, a typical motion retarget proceeds. As part of the process, an acquired raw motion is typically processed in some ways to create a more appropriate motion for each target character. The output of this step is the adapted motion data used to drive the target motion. For the case of an articulated figure, the output is usually joint angle data for all the joints.

The animation of human articulate body has long been received numerous attentions. The works in this area vary in terms of the body parts on which they focus. As for the hand, it has been a focus of many researches in computer animation because not only it is one of the most animated parts of human body but also one of the most complex body parts. In addition it is essential for human communication and expression. Our work will focus on estimating the 3D coordinate of the hand motion from 2D monocular video sequence.

1.2 Objectives

The objective of this project is to perform a 3D coordinate estimaation by using the motions from 2D image sequence which is an alternative to the traditional motion capture. This work will focus on motions of the human hand. The expected end result is the technique that is capable of estimating 3D hand motions from 2D video sequence. The resulting 3D hand motions can then be used as motion retarget source. The motion input will be 2D frame sequence of hand gestures. The output will be the 3D coordinate estimation of the deformed hand.

1.3 Project Scope

1. This work considers the hand gestures only.
2. The hand in a scene is expected to be at a certain distance from the camera.
3. The length of each segment on the hand is assumed to be known.
4. The hand in a scene is expected to be facing the camera.
5. The palm of the hand is expected to stay still.
6. The experimental data are extracted from Maya animation of a hand gesture.
7. The result is evaluated by comparing our output to the data from the Maya animation whose X, Y coordinate data are used as the input.

1.4 Research Procedures

1. Acquire a 3D hand model. This may be obtained from a free repository on the web.

2. Prepare the input sequence of hand gestures. This is obtained from a Maya animation.
3. Study and write a software module to calculate the Z coordinates.
4. Test the system with our hand gesture motion.
5. Analyze and evaluate the result.

1.5 Expected Benefits

We expect that our experiment on applying a variety of techniques to build a working system for estimation of 3D coordinate of hand motion from 2D video input will afford us to find out how well these techniques are working in practice and hopefully to discover some new insights based on the experience of building such systems that will be beneficial to others attempting similar tasks in the future.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Chapter 2

Related Theories and Literature Review

2.1 Hand Model/Anatomy

Hand anatomy has long been studied and well understood in the field of anatomy and biomechanics [7]. Hand is one of the most complex body parts. Most animation research focuses on its two main functionalities which are grasping and fine motor skills. Many aspects have been studied such as its constraints, limitations, DOFs, bones, tendons, and muscles.

Several hand models have been proposed over the years. Each has its own strengths and weaknesses. Whichever one we should use depends on the task at hand. A *parametric hand* model has been designed for the semiautomatic grasping approach in [8]. In [9] a simple volume-based animatable hand model constructed from geometric primitives has been employed for tracking. Reference [10] builds a *statistical hand* shape model from simplex meshes fitted to MRI data for their tracking system. For model-based finger motion capturing, reference [11] employs a learning approach for the hand configuration space to generate natural movement. Reference [12] presents an anthropomorphic finger model with a tendon transmission system based on pulleys and a position controller. The controller is modeled by a neural network and transforms tendon pull into joint motion. A model of the hand and arms based on manifold mappings has been proposed by [13]. They also consider inter-joint dependencies. Reference [14] uses Dirichlet free-form deformations (DFFDs) to simulate the tissue and muscle layer between skin and bones. Muscles are not considered directly, but the use of DFFDs allows the authors to model wrinkles at joints and bulging of segments dependent on the angle of rotation of the respective proximal joint. In [15] the joint movements of a hand model composed of rigid bodies are constrained by biomechanical laws. The model was designed for use in animating American Sign Language. An approach for skinning a hand skeleton using Eigen displacements has been proposed in [16]. The resulting hand model can be animated in real-time using graphics hardware.

Our hand model is a relatively simple kinematic chain consisting of joints and segments. Each joint has a number of DOFs and limitations.

2.2 Depth Reconstruction

Depth reconstruction refers to the process of extracting the depth information from 2D data. Its challenge lies in the fact that it is an under-determined problem. To solve it, we need to pose some constraints or use some assumptions and find a solution under that framework.

Study on 3D Depth recovery from 2D input has been performed for some time. There have been several techniques proposed. Reference [17] proposes an algorithm to compute the three dimensional structure of a scene from a pair of stereo images. Reference [18] constructs a 3D object query from 2D drawings. Their algorithm can handle objects with both planar and curved faces. Reference [19] estimates 3D depth from a single still image. It proposes the use of monocular cues (e.g., texture variations and gradients, defocus, color/haze, etc.) in addition to the stereo cues (e.g.). Their approach is based on modeling depths and the relationships between them at multiple spatial scales using hierarchical, multiscale Markov Random Field. The model is trained with a set of training images and their corresponding ground-truth depth maps. The method works for unstructured images of indoor and out door containing forests, sidewalks, buildings, people, etc.

More recently, as the 2D monocular video sequence is recognized as a fertile source of motions, several researchers focus on perfecting techniques that use them as input. Reference [20] and [21] reconstruct a human-like figure motion from 2D video stream. It assumes an existence of a library of motions similar to the target motion video stream and assumes the length of each segment is known. A library of motions that are similar to the target motions is used to provide a reference frame that will be warped based on the target frame to get the final pose. Their method is capable of reconstruction a highly dynamic motion for a full body of 40 DOFs. A technique based on Motion Trend Analysis has been proposed in [22] [23]. The method uses the information solved in the previous frame to solve for the next frame except the first frame. Hence, a user help is required to identify the correct 3D poses for the first few frames. Reference [24] exploits the domain specific knowledge about the target motions to find certain joint location and to limit possible poses. References [25] [26] [27] [24] use the orthographic projection method to determine the Z coordination.

To derive the Z coordinate from a single image, they assume the point corresponding and segment lengths are known and the certain distance between object and the camera are maintained. The problem of standard reflective ambiguity is also mentioned and resolved mostly with constraints. Reference [27] improves upon [25] by allowing some perspective cases to work properly.

We adopt the method similar to the one described in [25] which uses the scaled orthographic projection model. Please refer to the Concepts & Methods section for details of the technique.

2.3 Interdependence

Interdependence refers to the influence of a finger joint on others. Each finger joint is not fully independent but to some degree depend on the movement of some other joints on the hand. This can be viewed as *dependence constraints* between the DIP and PIP joints of each finger and between fingers. This concept has been studied and used in several works. Reference [28] observes that naturally a DIP joint cannot be moved without moving the PIP joint of the same finger. In another word there is a dependency between them. Reference [28] approximates the relationship between the two joint angle to be DIP = 2/3 PIP. They use this dependency to reduce the number of DOF by making DIP fully depend on PIP. Reference [13] uses interdependence in their work. Reference [29] expands the idea by assigning the degree of dependency between each joint across fingers.

Chapter 3

Proposed Method

3.1 3D Coordinate Estimation

3D coordinate estimation refers to the process of calculating the depth information from a 2D input source. In our case, the input motion is a 2D image sequence of a hand. We perform the following steps as parts of the 3D coordinate estimation process:

1. Identify the feature points (XY coordinates) of a hand in a video frame
2. Fill in the missing data
3. Decide on the reflective ambiguity
4. Calculate Z coordinates of the feature points

3.2 Input Acquisition

Our system needs three inputs from the user

1. Reference hand model (on image plane). This reference hand model can be one of the input frames. It should show the full stretched hand on the image plane. This will be used to establish the segment lengths. A segment refers to a segment of a finger. For example each finger has three segments.
2. The length of each segment. We need the length of each segment for 3D depth reconstruction. The user may not need to explicitly specify the length of all the segments. Theoretically, we need only one segment length and we can calculate the rest using the information from the reference hand model image.
3. 2D monocular video sequence of hand gestures. In our experiment, Maya animation of a hand gesture is used.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

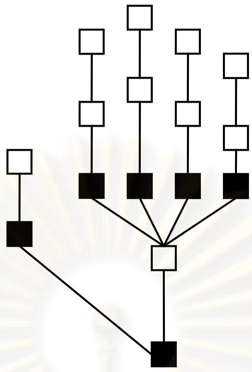

Figure 3.1 The DOFs of each of the joint in our hand model. The black node has 2 DOFs. The white node has 1 DOF.

3.3 Our Hand Model

The specification of our 3D hand model is as follows:

1. There are 14 joints and 19 degrees of freedom in each hand
2. Each finger except thumb has three joints and sum up to 19 DOFs in a hand (figure 3.1)

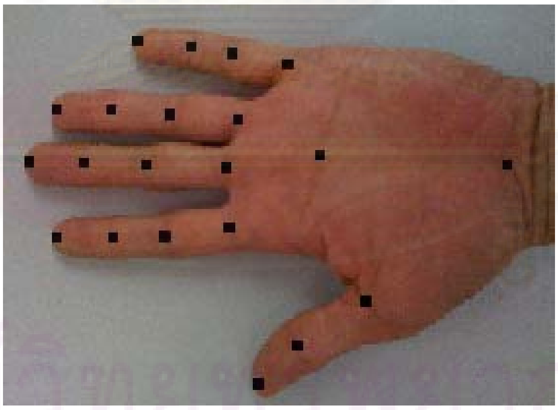

Figure 3.2 The feature point locations of our hand model.

3.4 Feature Points Identification (XY Coordinates)

For each input image sequence of a hand gesture, we identify the locations of all feature points (figure 3.2). The feature points in our case include the locations of joints and the tip of each finger and two more locations on the palm. A location is specified as the XY coordinates of the following locations:

\# Location

5 Tip of {Thumb, Index, Middle, Ring, Little}

4 *Distal interphalangeal joint* (DIP) of {Index, Middle, Ring, Little}

1 *Interphalangeal joint* (IP) of Thumb

4 *Proximal interphalangeal joint* (PIP) of {Index, Middle, Ring, Little}

5 *Metacarpophalangeal joint* (MCP) of {Thumb, Index, Middle, Ring, Little}

1 Folding on the palm

1 Wrist

In some images, it may be impossible to identify all of these feature point locations because of occlusion or blurred image. In such cases, we have employed a technique to approximate their locations. These techniques are discussed in details later. Also, one assumption is that if a feature point is occluded, probably its exact location is irrelevant in that context and it should be able to be estimated by its rest pose which is approximately somewhere in the middle of its range (in case of a joint) [30].

3.5 3D Depth Reconstruction

Since our input is a sequence of 2D image, the information we get for each feature point is 2D. Thus, we need a way to compute for the Z coordinate. To do this, we adopt the method in [25] which uses the scaled orthographic projection model. A projection of a point (x, y, z) in three-dimensional space to the point (x, y, 0) on the x-y plane can be represented as a matrix (equation 4).

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad (1)$$

In scaled orthographic projection, we simply add a scale factor to the equation (equation 2). This results in a simple scaling of the object coordinates. The scaled-orthographic model amounts to parallel projection, with a scaling added to mimic the effect that the image of an object shrinks with the distance [31].

$$\begin{pmatrix} u \\ v \end{pmatrix} = s \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2}$$

The formula is expressed in equation 5.

$$\begin{aligned} l^2 &= (X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2 \\ (u_1 - u_2) &= s(X_1 - X_2) \\ (v_1 - v_2) &= s(Y_1 - Y_2) \\ dZ &= (Z_1 - Z_2) \\ dZ &= \sqrt{(l^2 - ((u_1 - u_2)^2 + (v_1 - v_2)^2) / s^2} \end{aligned} \tag{3}$$

$$\begin{aligned} u &= s \cdot X \\ v &= s \cdot Y \end{aligned} \tag{4}$$

$$(Z_1 - Z_2)^2 = l^2 - [(u_1 - u_2)^2 + (v_1 - v_2)^2] / s^2 \tag{5}$$

$$s \geq \frac{\sqrt{[(u_1 - u_2)^2 + (v_1 - v_2)^2]}}{l} \tag{6}$$

From equation 5 we assume an arbitrary depth for Z_1 and compute for Z_2. In this case, we also know u_1, u_2, v_1, v_2, and l. If we also know s, the scale factor, then we will be able to solve for Z_2. In our case we assume that the distance between the camera and the hand is much greater than the depth of Z coordinate. (Note that this assumption is needed for the scaled orthographic projection model to work.) With this assumption, the scale factor is almost constant for all the joints on the hand. So we can use the same scale value for all the feature points. Now to compute for the scale factor s, we use equation 6 to find the overall minimum value of s. Note that equation 6 comes from the fact that the equation 5 has a real solution. We will use the minimum overall value of s in our computation since the absolute values of X , Y and Z is not necessary. All we need is the relative depth between each feature point. Once we obtain s, we can use equation 4 to find the value of X and Y. We then use the computed Z_2 as the Z_1 of the next segment. We then repeat this process until all feature points are computed. One issue that we still have is the reflective ambiguity. This stems from the fact that the Z_1 or Z_2 in equation 5 can be the smaller one based on the 2D information we have. In our case, joint angle limit, physiological constraints are used to pick the correct configuration.

From this step, we get XYZ coordinates of feature points. These values are imported into the Maya scene to animate the result motion on our hand model.

3.6 Reflective Ambiguity

As stated earlier, the computed Z coordinate can be ambigious. This is because the Z coordinate value of two points along Z axis can be calculated from the same X and Y values. The figure 3.3 shows an example of two points in 3-D space which have the same X and Y values but different Z values.

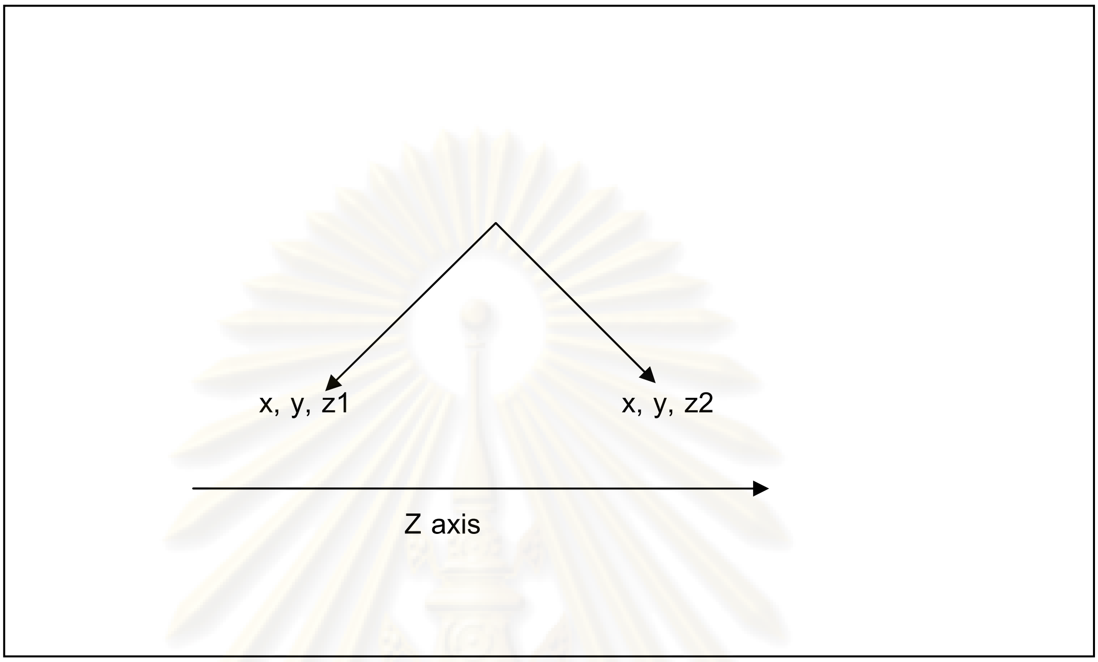

Figure 3.3 The reflective ambiguity of Z coordinate

We use the following constraints to resolve the ambiguity in most cases. Our constraints are based on the information from related feature points on the same finger.

In the following explanation, let us call the MP joint, the PIP joint, the DIP joint and the tip of the finger as the feature point A, B, C, and D respectively. In our system, we assume that the palm is facing the camera and the palm stays upright. From this assumption and our observations, we enforce the following constraints on the value of the Z coordinate of a feature point.

- Z coordinate of the feature point B is always greater than that of A
- Z coordinate of the feature point C is always greater than that of B
- Z coordinate of the feature point D is less than that of C when the ABC angle is less than or equal to 90 degree

Based on these constraints, the relative Z coordinate of the PIP and DIP feature point (B and C) are always the addition of its parent (MP and PIP respectively)'s Z coordinate. That is they are pointing away from the palm.

For the tip of the finger, our method considers the location of the MP, PIP and DIP joints simultaneously. In particular, we measure the inner angle at the PIP joint. If it is less than 90 degree, the tip of the finger should be pointing toward the palm. That is its calculated Z coordinate is subtracted from its parent (DIP)'s Z coordinate to form its world Z coordinate. The idea is depicted in the figure 3.4.

Figure 3.4 The tip of the finger points toward the palm if the inner angle of the PIP joint is less than 90 degrees.

The inner angle of the PIP joint is calculated using the law of the cosines as we already know the YZ coordinate of the MP, PIP, and DIP feature points.

3.7 Occlusion and Missing Data Handling

Occasionally, it is possible that some feature point input data cannot be obtained. This can be caused by several reasons. First, a feature point on a finger is occluded by other part of the hand. For example when a hand is clinching into a fist, the feature points at the tip of index, middle, ring and pinky fingers are all occluded when the palm is facing toward the camera. Second, an input image is not clear. There may be some part of the image that is unclear and cannot be detected.

In our experiment, we assume that the first frame is perfect. This means all the feature points are available in the first frame. If this is not the case in the real world, we need the help of the user to specify the missing feature point data to make sure that all the feature points of the first frame are available.

To fill in the msising data, we experimented with five different methods. The first method to deal with missing data is to use the data from the previous frame. This method is very simple and does not need any information from other feature points.

The second method is to apply the amount of change occurring between the previous two frames to the missing frame. For example, if the X coordinate value of feature point A is missing in frame 3 and the X values of this feature point in frame 1 and 2 are 5 and 8 respectively, the predicted X value in frame 3 will be 8 + 3 = 11.

Frame	A.X	B.X	Method 2	Method 3
0	3	3	3	3
1	5	5	5	5
2	7	4	7	7
3	?	3	9	5

Table 3.1 An example of missing data handling using the interfinger dependency

The third method is similar to the second method with the addition of interfinger dependency. This dependency will enable us to detect a directional change of the missing values. That is we monitor the trend of value change from a depended-on or parent feature point. If there is a change in the direction of value (for example from increasing to decreasing) of the parent feature point, the same directional change is applied to the predicted value. The table 3.1 shows an example. In this example, a feature point A depends on a feature point B. And the X value of A is missing in frame 3. After we evaluate the trend of B.X, we see that the value trend is changing from increasing (i.e. 3 to 5 from frame 0 to frame 1) to decreasing (i.e. 5 to 4 from frame 1 to frame 2). Thus we decide that the value of A.X should be decreasing in frame 3. As a result, we predict the value of A.X at frame 3 to be 7 – 2 = 5. As a comparison, method 2 without an interfinger dependency would predict the value to be 7 + 2 = 9.

The fourth method is similar to the third method. However, instead of using interfinger dependency, an intrafinger dependency is used. This method has a hypothesis that the intrafinger relationship is stronger than the interfinger relationship. Thus, intrafinger dependency should provide more accurate predicted value.

The fifth method is similar to the fourth method. However, in addition to the directional cue from the parent feature point, we also use its value change rate as well.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Chapter 4

Experiment and Result

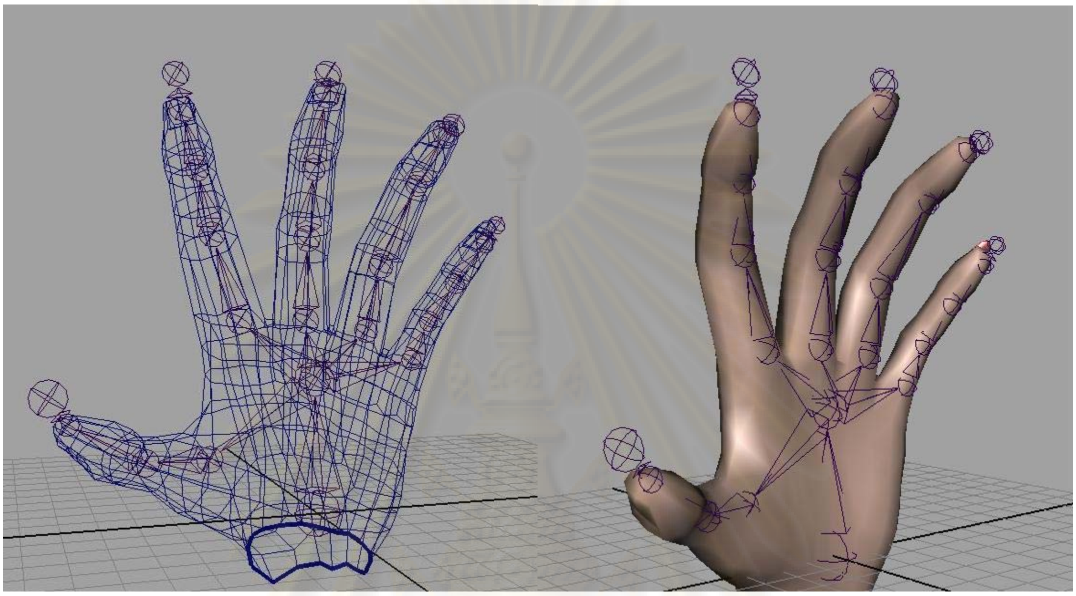

Figure 4.1 Our Maya hand model.

4.1 The Experiment

In our experiment, we first create a Maya hand model (figure 4.1) to have the joint as specified in the section 3.3. Then, we have created an animation of the Hand Clinching motion to be used as the input in our experiment. There are a few reasons for choosing Maya animation as the input in our experiment. First, we can get a very accurate XY coordinate to use. This will eliminate the input errors from our experiment. Second, in addition to X and Y coordinates, we also get the Z coordinates from the Maya animation. This is very useful for us as they can be used to validate our result. Our Hand Clinching animation contains the total of 100 frames. Some examples of the frames are shown in figure 4.2.

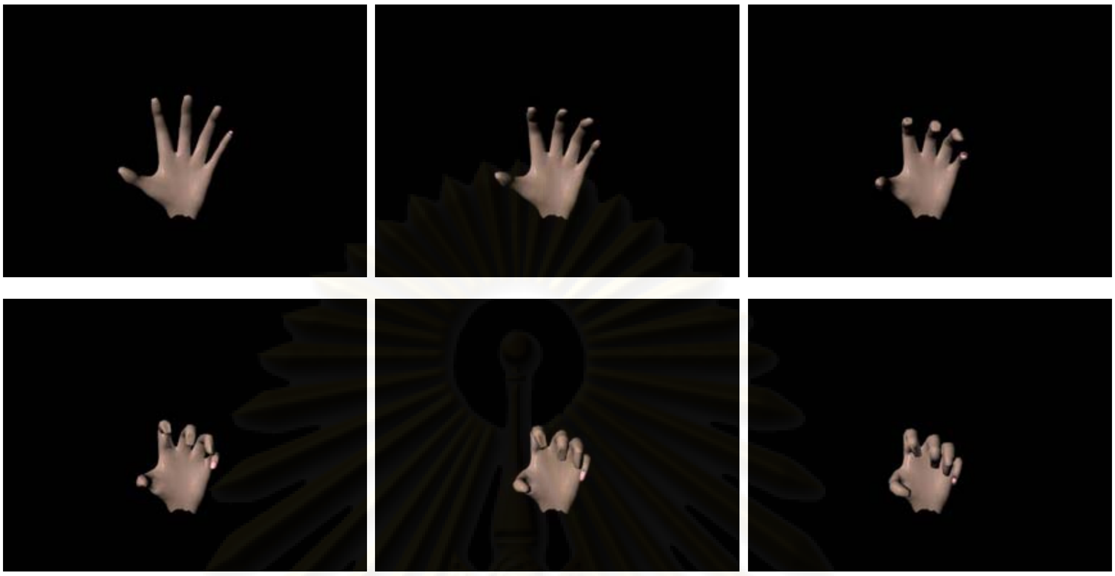

Figure 4.2 Examples of input from Maya animation.

To extract the XY coordinates of the feature points from this animation, we wrote a Maya plugin using Maya API. This program goes through each frame, extracts the X, Y, and Z coordinate of each feature point and writes them to an output file. The plug in code is listed in Appendix C. An example of the extracted coordinates is shown in figure 4.3.

```
j0 0 -4.6975 -0.206784 3.18228
j0 1 -4.71145 -0.252324 3.13728
j0 2 -4.72495 -0.297166 3.09138
j0 3 -4.73798 -0.341292 3.0446
j0 4 -4.75056 -0.384681 2.99695
j0 5 -4.76268 -0.427315 2.94846
j0 6 -4.77434 -0.469174 2.89914
j0 7 -4.78554 -0.510241 2.849
j0 8 -4.79629 -0.550497 2.79807
j0 9 -4.80657 -0.589926 2.74637
j0 10 -4.8164 -0.628508 2.69391
j0 11 -4.82578 -0.666229 2.64071
j0 12 -4.83469 -0.703071 2.58679
j0 13 -4.84316 -0.739018 2.53218
j0 14 -4.85117 -0.774055 2.47688
j0 15 -4.85873 -0.808167 2.42093
```

Figure 4.3 Examples of XYZ coordinates from Maya animation.

Please note that in addition to the X and Y coordinates, we have also extracted the Z coordinate. However, only the X and Y coordinate are used as the input to our Z coordination calculation software. The extracted Z coordinate will be used later to verify our result.

After we have obtained the file containing data as shown in figure 4.3, we pass it as the input to our Z coordinate calculation software. This software will calculate the Z coordinate of each feature point in each frame. The program code is listed in Appendix B. The details of the program are discussed in the next section.

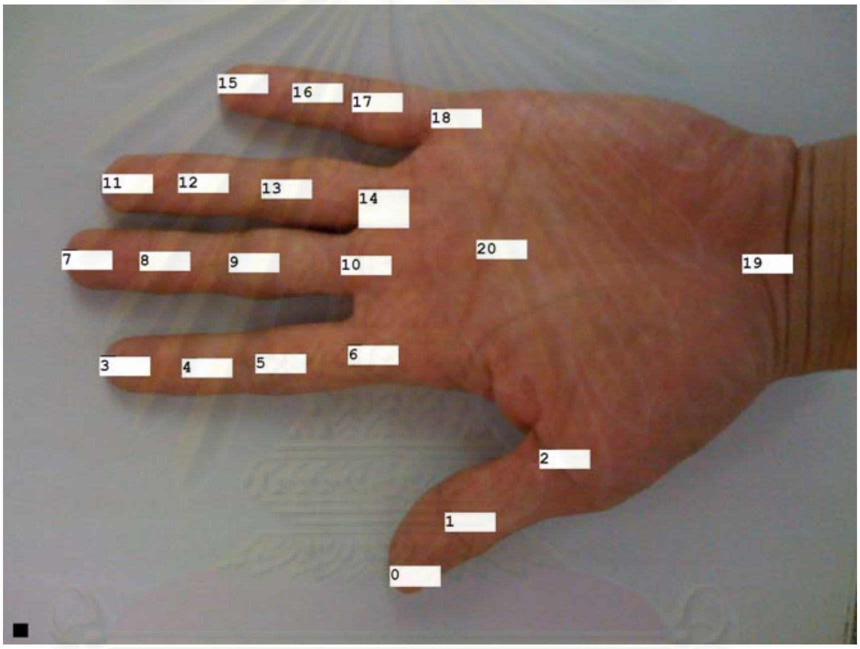

Figure 4.4 The feature points and their IDs.

4.1.1 Z Coordinate Calculation Program

This is the program to calculate the Z coordinate values based on the proposed techniques. The program inputs are the X and Y coordinates in all the frames, the actual segment lengths. The program also fills in the missing X and Y data with the value we guess using the values from the previous frame, the values of associated feature points in the current frame and the interdependence data. In each frame we have to calculate the Z coordinates of 21 feature points of the hand. The order of calculation is important. The output is the list of calculated Z coordinates of feature points in all frames. The program is listed in Appendix B.

4.1.1.1 The Order of Feature Point Calculation

The order of the Z coordinate calculation is defined. In our design, we assume that the palm is facing toward the camera and it does not move. So the Z coordinates of the feature point 19 and 20 are assigned to be 0. Next, each finger's feature points are calculated in order starting from the thumb, to the pinky finger.

For each finger, the calculation order starts from the base to the tip of the finger. For example, the feature point 2 is calculated before the feature point 1. And the feature point 1 is calculated before feature point 0.

The reason for the exact order of calculation is because we need the Z coordinate of the previous feature point to calculate the world coordinate of a feature point as their positions are related to each other. Also, when we try to fill in the missing feature point data (e.g. occlusion), we need the information of the previous feature point. So we need to make sure that this information is already available.

4.1.1.2 The Dependency of Feature Points

A feature point's Z coordinate is computed based on another feature point. This is the joint that together with the feature point forms a segment of a finger. In our design, the parent joint is used. Thus, a feature point is dependent on the joint above it in the joint tree. For example, from figure 4.4 the feature point 2, 6, 10, 14, 18 are dependent on the feature point 20. The feature point 0 is dependent on the feature point 1. The feature point 1 is dependent on the feature point 2 and so on.

The dependent feature point is used for two reasons. First, together with the feature point it forms a segment of a finger. We need this segment length in the Z coordinate calculation. Second, the calculated Z coordinate is relative to this feature point. So to obtain the world coordinate we add or substract the calculated Z coordinate value to the Z cooridinate value of this parent feature point.

4.1.1.3 Filling in Missing Data

When the program starts, it reads frame data from the input file, then for each frame, it determines whether the XY coordinate of any feature point is missing. If that is the case, it tries to guess the missing value using the algorithms described earlier in section 3.7. After this step, a frame has complete XY coordinate data of all feature points. And we are ready to compute the Z coordinate of each feature point.

4.1.1.4 Z Coordinate Calculation

For the first feature point (i.e. the folding palm or the feature point 20), the Z coordinate is assigned to 0. This is fine since we do not need to know the exact Z coordinates of these feature points. What we are trying to compute is the relative Z coordinate of these feature points.

For the rest of feature points, we compute their Z coordinate values as a relative value from the feature points they depends on. The Z coordinate is calculated form the following formula,

```
vertex1.w = vertex2.w +/- sqrt( pow2(l) - pow2(abs(vertex1.u-vertex2.u)) - pow2(abs(vertex1.v-vertex2.v)) );
```

One issue we have found is that The term *(pow2(l) - pow2(abs(vertex1.u-vertex2.u)) - pow2(abs(vertex1.v-vertex2.v)))* is sometimes negative. This can occur if there is an inaccuracy in such data we have obtained as a specified segment length or some of the XY coordinate values. To solve this problem, we force this term to become positive by adjusting the value of the segment length (l) little by little.

Once the relative Z coordinate value is calculated, we add it to the depended-on feature point's Z coordinate to obtain its world coordinate with the exception of the tip of the finger feature points (i.e. the feature point 3, 7, 11, 15 in figure 4.4). For the fingertip feature point, we check for the reflective ambiguity as detailed in section 3.6 and the addition or subtraction to the depened-on feature point's Z coordinate will be performed accordingly. We perform this calculation for every input frame and write the result to the output file. An example of the output is depicted in figure 4.5.

```
j0 0 -4.6975 -0.206784 5.0577
j0 1 -4.71145 -0.252324 5.01302
j0 2 -4.72495 -0.297166 4.96747
j0 3 -4.73798 -0.341292 4.92106
j0 4 -4.75056 -0.384681 4.87382
j0 5 -4.76268 -0.427315 4.82576
j0 6 -4.77434 -0.469174 4.7769
j0 7 -4.77434 -0.469174 4.77201
j0 8 -4.77434 -0.469174 4.766
j0 9 -4.77434 -0.469174 4.75892
j0 10 -4.77434 -0.469174 4.75082
j0 11 -4.82578 -0.666229 4.5214
j0 12 -4.83469 -0.703071 4.46822
j0 13 -4.84316 -0.739018 4.41439
j0 14 -4.85117 -0.774055 4.35997
j0 15 -4.85873 -0.808167 4.30496
```

Figure 4.5 Examples of output from Z coordinate calculation program.

4.1.2 Other Programs

Besides the Z coordination calculation program, we have written a number of other programs. First, we wrote a Maya plugin using Maya API to extract the X, Y, and Z coordinates of each feature point in a frame. The output of the program is the list of X, Y, and Z coordinates of each feature point in a frame. The code is listed in Appendix C.

Second, we wrote a program to compute the difference between the actual and calculated Z corrdinated and sort them in proper order. The programs are listed in Appendix E and F.

Third, we wrote a Maya plugin to import our calculated Z coordinate values and use them along the the original X and Y coordinates to create the output animation. The program is listed in Appendix D.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

4.2 Result and Analysis

4.2.1 Z Coordinate Calculation

In our experiment, we choose to use a motion of a clinching hand (see figure 4.2). We believe that this motion provides a wide range of motions of each finger and hence is a good candidate for being used in our experiment.

As stated earlier, we can validate the result of our computation and see how well it performs by comparing the calculated results of Z coordinates with the corresponding actual values we obtain from the Maya animation. The table 4.1 shows the result of Z coordinate calculation both with and without the reflective ambiguity check. The result is shown in the form of the difference between the actual and the calculated Z coordinate values. The table lists the minimum, the maximum, and the average difference for each feature point over 100 frames.

Feature Point	No Reflective Ambiguity Check				Reflective Ambiguity Check			
	Min	Max	Average	Std Dev	Min	Max	Average	Std Dev
0	0	1.06126	0.0500225	0.187208	0	1.06126	0.0500225	0.187208
1	0	0.00001	0.000003	0.0000046	0	0.00001	0.0000031	0.0000046
2	0.000001	0.000001	0.000001	0	0.000001	0.000001	0.000001	0
3	0.00001	2.79918	0.377438	0.774469	0.00001	0.3105	0.00582	0.0359771
4	0	0.03278	0.00034293	0.00326018	0	0.03278	0.00034293	0.00326018
5	0	0.000127	0.00001	0.0000181	0	0.000127	0.00001	0.0000181
6	0.0000033	0.0000033	0.0000033	0	0.0000033	0.0000033	0.0000033	0
7	0	3.92996	0.581189	1.09838	0	0.33847	0.0105844	0.0501362
8	0	0.28688	0.0051299	0.0334249	0	0.28688	0.0051299	0.0334249
9	0	0.000039	0.00000482	0.0000064	0	0.000039	0.00000482	0.0000064
10	0.000007	0.000007	0.000007	0	0.0000007	0.0000007	0.0000007	0
11	0.07749	3.2563	0.551863	0.882854	0.07671	0.335839	0.0825675	0.0288816
12	0.07737	0.33588	0.081987	0.0284531	0.07737	0.33588	0.081987	0.0284531
13	0.07745	0.116305	0.0782574	0.00543573	0.07745	0.116305	0.0782574	0.00543573
14	0.077459	0.077459	0.077459	0	0.077459	0.077459	0.077459	0
15	0	2.08428	0.240371	0.526386	0	0.31035	0.00828877	0.0407495
16	0	0.124255	0.00174183	0.0132183	0	0.124255	0.00174183	0.0132183
17	0.000004	0.00004	0.0000127	0.00000642	0.000004	0.00004	0.0000127	0.00000642
18	0.000006	0.000006	0.000006	0	0.000006	0.000006	0.000006	0
19	0.881104	0.881104	0.881104	0	0.881104	0.881104	0.881104	0
20	0	0	0	0	0	0	0	0

Table 4.1 The table shows the mimimum, maximum, average and standard deviation of the difference between the actual and calculated Z value of each feature point.

The result shows that the minimum difference between the actual and calculated Z values is the same for both options for most feature points. This is because the frame that produces the minimum difference does not exhibit the reflective ambiguity. So both options yield the same Z value.

One exception is for feature point 11. With the reflective amibiguity check turned on, the minimum difference between the actual and calculated Z coordinates occurs at frame 72 where the check detects the ambiguity and correctly decides that the feature point (which is the tip of the ring finger) should be pointing inward. With the reflective ambiguity check turned off, the minimum difference occurred at frame 14. However, in general, both options produce very similar minimum difference between the actual and calculated Z values.

From the maximum difference columns, it is evident that there is a difference in term of performance between the two options at all tip feature points where the Reflective ambiguity check is at work. The difference is caused by the fact that the reflective ambiguity check can detect the ambiguity and makes the right decision so the gap between the calculated and actual Z coordinates is small while the non ambiguity check option does not recognize the ambiguity and produces the Z coordinate in the wrong direction which results in a bigger gap. For all other feature points than the tip ones, both options produce the same result as our reflective ambiguity check works for the tip feature points only.

Another interesting point is that the DIP feature points produce a larger maximum gap than the PIP feature points which in turn produce a larger maximum gap than the MP feature points. This is because there is generally more motion change at the feature points nearer to the tip of the finger in our experiment.

From the experiment, most maximum differences between the actual and calculated Z values occur in the last frame. A few exceptions are for feature point 3, 7 and 15. For feature point 3 (the tip of index finger), the maximum difference occurs at the frame 77. This is because the reflective ambiguity check fails to detect the ambiguity as the measured angle just falls off the threshold of 90 degree. So the calculated Z coordinate is pointing in the wrong direction and produces a big gap. For feature point 7 (the tip of the middle finger), the biggest difference occurs at frame 69 where the ambiguity is wrongly detected and the algorithm decides that the tip should point inward instead of outward. For feature point 15 (the tip of the ring finger), the maximum difference occurs at frame 80, when the reflective ambiguity check option fails to detect the ambiguity and produces the biggest gap.

The difference in performance between the two options is evident in the average gap between the calculated and actual Z coordinates they produce. The uncheck option produces a much bigger gap on average for all the tip fingers where reflective ambiguity check is working.

The standard deviation also shows that the check option consistently calculates a closer Z values than the uncheck option. The wrong direction of Z values produced by the uncheck option in the frames that ambiguity occurrs accounts for the big standard deviation values. Again the big difference of the standard deviation occurs at the tip feature points. This indicates that the check option can correctly solve the ambiguity and keeps the gap between the calculated and actual Z coordinate values close throughout.

From the result, we observe that the accuracy of the segment length provided by the user has a significant impact on the outcome. In one of the experiments, the result shows noticably inaccurate values of Z coordinates. After an investigation we found that they were caused by the wrong values of segment length as we recreated our hand model but failed to update the corresponding segment lengths. Later on, the segment lengths were remeasured, and the result looked much better.

In addition to the sensitivity to the segment length inaccuracy, the accuracy of the XY coordinate input is also very important. In practice, this can potentially pose a serious issue to our technique. From our experience, a very accurate way of obtaining the XY coordinate input is critical to the accuracy of our method.

From the experiment, we learn that our method has the advantages of simplicity and speed. Since all the computation involves only simple formulas such as Pythagoras theorem and law of cosines, the implementation is quite simple and the computation time is very fast in comparison to some other more sophisticated methods that involve nonlinear functions. We concede that there may be a tradeoff between the accuracy and the speed. This however is not measured in our experiment. So we cannot say for sure.

Another advantage is the applicability to 2D input. This may be crucial for several applications. For example, we might want to reproduce a historical footage or some classic 2D cartoon in 3D. Our method is intended to work with this kind of media.

4.2.2 Missing Data Handling

In this study, we have experimented with five different methods for predicting the missing XY coordinates as described in section 3.7. In the experiment, we have intentionally excluded the XY coordinates of some feature points in certain frames. The decision for which feature points to be excluded in a frame is based on the animation of the clinching hand motion. The table 4.2 shows the list of missing data of each feature point.

Feature Point	Missing Frames	Feature Point	Missing Frames
0	-	11	98-99
1	37-47	12	24-37
2	-	13	37-47
3	64-99	14	52-99
4	37-47	15	-
5	-	16	33-37
6	86-99	17	37-47
7	92-99	18	50-99
8	31-37	19	-
9	37-47	20	-
10	52-99		

Table 4.2 The feature points and their missing frames for the Clinching Motion.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

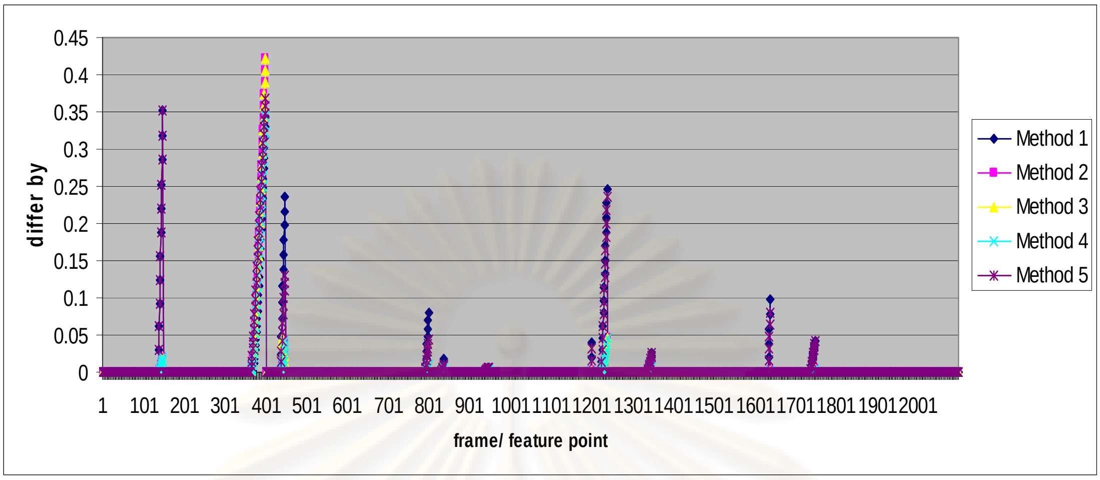

Figure 4.6 The graph shows the difference between the actual and predicted X values using the five different methods.

The chart in figure 4.6 shows the result of applying the five methods of estimating the missing data in our experiment. The Y axis of the graph in figure 4.6 shows the difference between predicted X and actual X values of a feature point. The X axis lists all the frames of each feature point. For example, the frame 1-100 is the frame 1-100 of feature point 0, the frame 101-200 is the frame 1-100 of feature point 1 and so on.

From the graph, there are several spikes. These are the points where there are noticable differences between the actual and predicted coordinate values.

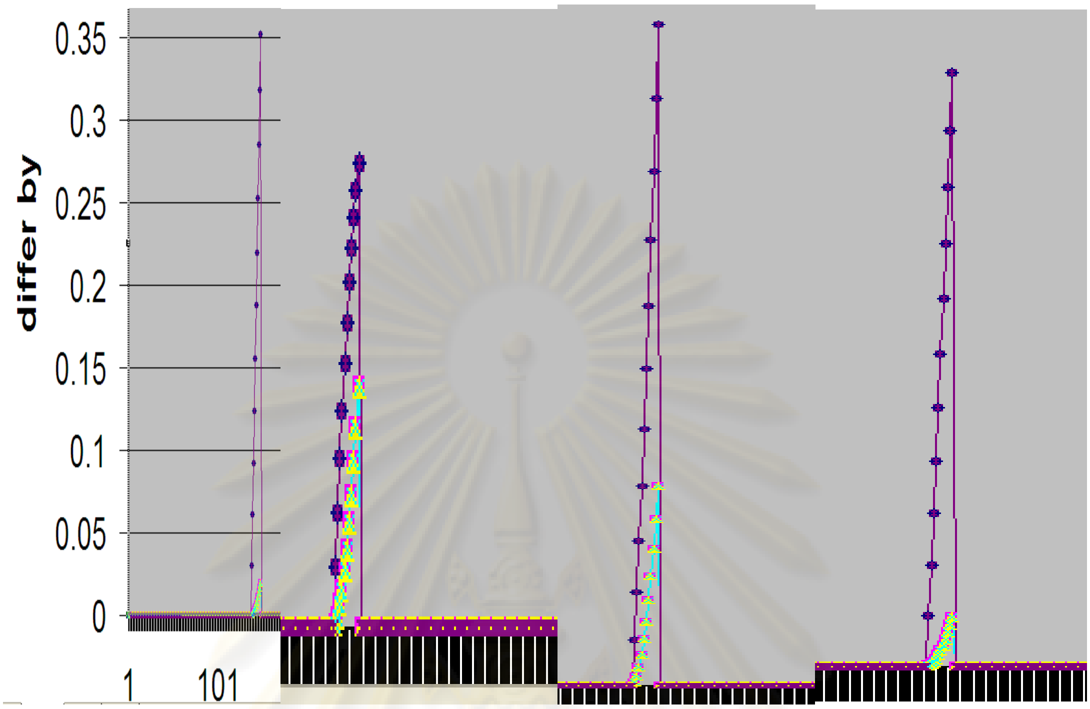

Figure 4.7 The zoomed-in figure of spike 1, 6, 9 and 11.

Spike 1, 6, 9, 11:

Spike 1 belongs to feature point 1. Spike 6 belongs to feature point 9. Spike 9 belongs to feature point 13. Spike 11 belongs to feature point 17. Although there is some difference in magnitude, all these spikes exhibit the same graphic pattern. For these spikes, method 2, 3, and 4 yield the same performance. This is because there is no directional change in X coordinate values for the duration of the missing frames.

Method 1 and 5 also yield the same performance. This is because method 1 uses the X value of the previous frame as the predicted values. So the predicted values stay the same for the whole period of the missing frames. And eventhough method 5 uses the rate of change of the parent feature point to predict the value of the child feature point. In this particular case, the rate of change of the parent feature point happens to be 0, so the predicted value also stays the same for all the missing frames. Hence both methods produce the same predicted values.

Method 2, 3, and 4 perform better than method 5 because the rates of value change of feature point 1 and of its parent feature point (2) are different in our experiment. In particular, the feature point 2's X values stay the same for the entire clip while the X values of feature point 1 linearly increases. As a result, method 5 which uses the change rate of the parent feature point to predict the value of the child feature point produces the flat predicted X values (as the rate of change of feature point 2 is 0). That results in a gap between the actual and predicted X values getting wider for each missing frame. This is the same case for spike 1, 6, 9 and 11. Notably, they are all PIP feature points whose parent feature points are MP. And in our experiment all MPs do not move.

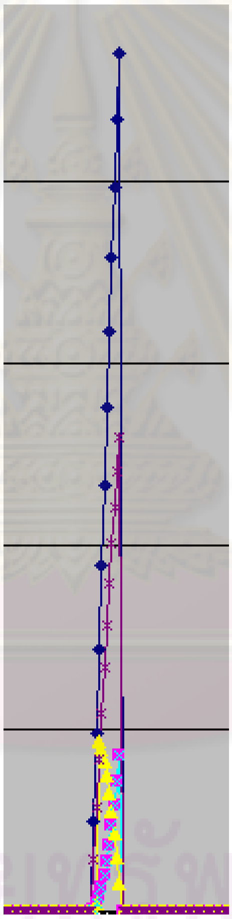

Figure 4.8 The zoomed-in figure of spike 3 which belongs to feature point 4.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Spike 3:

Spike 3 belongs to feature point 4. From the result, method 1 performs the poorest for this feature point as the predicted and actual values are getting further apart for each of the consecutive missing frames. This is because the actual X values are linearly increasing in the period of missing frames while the predicted values stay constant.

Method 5 performs the second worst because the rate of change of the parent feature point is slower than that of the child feature point. So the predicted values which are calculated from the rate of change of the parent feature point does not keep up with the actual pace and thus a gap is getting wider with every missing frame. However, the predicted values are still closer to the actual values than those yielded by method 1.

Method 4 produces the same predicted values as method 2 in this spike because method 4 does not detect any directional change.

Frame	X Value
34	-0.247736
35	-0.24445
36	-0.241164
37	-0.237878
38	-0.241959
39	-0.24039
40	-0.23897
41	-0.237688

Table 4.3 The X values of feature point 8 for frames 34-41.

Method 3 doesn't perform well because it detects a false directional change. This incorrect detection is caused by the fact that the feature point 8 which is the parent feature point of feature point 4 also has missing frames at this period (frame 31-37). And the predicted values for feature point 8 are a bit ahead of the actual pace and that results in a misleading directional change at the point where an actual value follows the last predicted value (frame 37 and 38 in table 4.3). At the point of false directional change, the predicted value is moving in the opposite direction of the actual value, hence the spike goes up. However, at frame 39, another directional change is detected, and the X value of the feature point 8 goes back to the correct direction again. Hence the spike comes down.

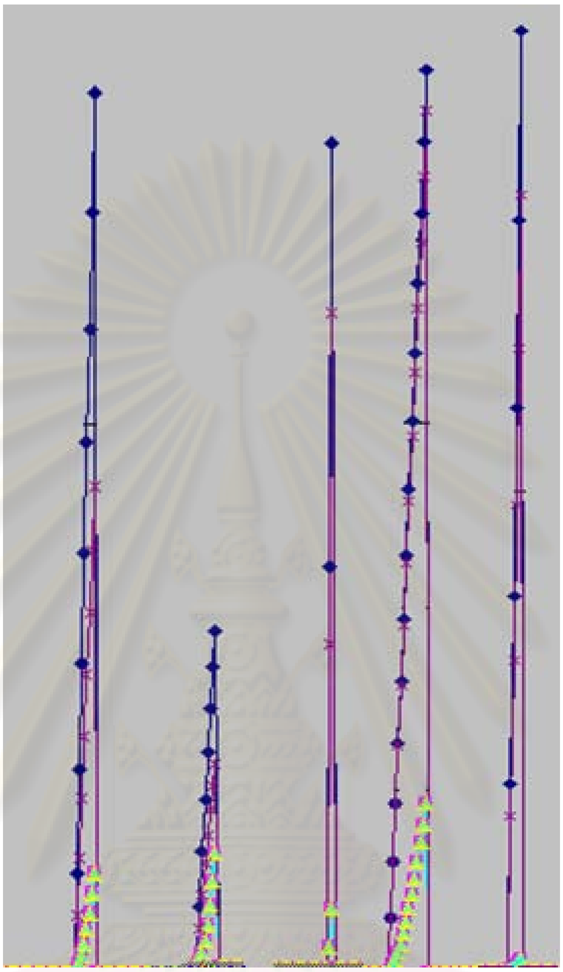

Figure 4.9 The zoomed-in figures of spike 4, 5, 7, 8 and 10 which belong to feature point 7, 8, 11, 12 and 16 respectively.

Spike 4, 5, 7, 8, 10:

Spike 4, 5, 7, 8 and 10 belong to feature point 7, 8, 11, 12 and 16 respectively. For these spikes, method 2, 3 and 4 yield the same performance as no directional change is detected neither with intrafinger (method 4) nor interfinger (method 3) dependency. Despite that, their predicted values are more accurate than those obtained from method 1 and 5.

Method 5 performs poorly but still beats method 1. This is because the rate change of the depended-on feature point is eventhough not consistent with that of the feature point but still is proven to be better than using the just previous frame value as done by method 1.

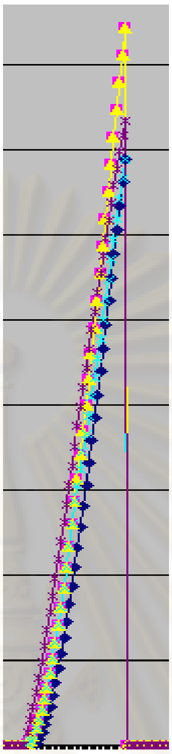

Figure 4.10 The zoomed-in figure of spike 2 which belongs to feature point 3.

Spike 2:

Spike 2 belongs to feature point 3. In this case, method 2 and 3 yield the same performance. Actually method 3 detects a directional change which occurs at frame 66 of the depended-on feature point (7). However, in this clip, the X value of feature point 3 and its depended-on feature point 7 head in the opposite direction. So the detection doesn't change the direction of the predicted value since it already moves in that direction. As a result, the predicted values keep going in the wrong direction and cannot produce a better result than method 2.

Method 1 surprisingly performs better than method 2 and 3 for this feature point. This is because of the directional change of the X value. So method 1 which uses the previous frame values and produces flat predicted values yields a smaller gap than method 2 and 3 which produce linearly increasing predicted values that move in the opposite direction of the actual values. As stated earlier, method 3 fails to work correctly because the X values of the parent and child feature point head in the opposite directions.

Method 4 yields similar result to method 2 and 3 albeit a bit better. This is because method 4 which employs intra-finger dependency can detect the directional change at frame 82 from the depended-on feature point (4) and turns to the right direction. However, the predicted values still keeps falling further behind the actual values as the predicted rate could not keep up with the faster actual rate.

Similar to method 4, method 5 can also detect the directional chnage and changes the direction accordingly. However, the difference between the predicted value and the actual value still grows larger for each missing frame because the rate of the predicted value is faster than the actual rate.

From these result, we see that method 1 and 5 perform poorer than the other three methods. The method 1 performs poorly because it blindly uses the value from the previous frame as the values of the missing frames. So if there are several contiguous missing frames, the predicted values of the missing frame will be further away from the actual value as the predicted values continue to stay the same while the actual values of the missing frames are likely to move in one direction away from the previous frames.

In our experiment, the method 5 does not perform as expected because for the clip used in our experiment, the rates of change of a feature point and its dependent feature point do not coincide. So when there are several contiguous missing frames, the predicted values which are derived from the rate of change of the depended-on feature point grow faster or slower and consequentially create a wider gap for each missing frame.

For method 3, 4, and 5, we assume both parent and child feature points are directionally compatible. So it doesn't work well in the case where their coordinate values are actually growing in the different directions. Also, we assume the same or similar rate change between the parent and child feature points in the dependency relationship. So the method fails when that assumption is not true. Thus, choosing the right dependency is important to the success of these methods.

In our study, we see that the methods that apply the amount of coordinate change from the previous frame (method 2, 3, 4) work well especially if the number of missing frames is small. This is because there seems to be a locality of value change. In other words, the rate of coordinate value change of neighbouring frames is very similar.

In the case that there is a directional change of coordinate values during the period of missing frames, method 3 and 4 proves to be useful. However, this feature is not very important if the number of consecutive missing frames is small. Also, this strategy very much depends on the depended-on feature point. So again choosing the right dependency between feature points is crucial.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Chapter 5

Conclusions and Suggestions

5.1 Conclusions

We have shown a method to estimate the 3D coordinate from the 2D hand motion. In this method, we employed a number of techniques to derive the missing Z coordinates and in some cases the X and Y coordinates. The main techniques that we use are the the orthographic projection method which is used to determine the Z coordination. The occlusion and the missing X and Y coordinate data are tackled with the interdependence, previous frame data, and natural rest pose of a hand.

The experiment uses the input from Maya animation. An added advantage of using Maya animation as an input in our experiment is that we are able to obtain the actual Z coordinate to verify our result.

In our study, the Z coordinate values are computed with both the reflective ambiguity option on and off. The result shows that our method with the reflective ambiguity option produces more accurate result at the tip feature points where the ambiguity check is employed.

In summary, we have seen from our experiment on applying the variety of techniques to build a system for estimating 3D coordinate from 2D video input and see how well these techniques are working in practice.

We hope that some new insights based on the experience of our experiment will be beneficial to others attempting similar tasks in the future. Moreover, we hope that our system can be used to generate interesting hand animation from 2D video. Some of the potential applications are sign language interpreter, game industry, etc.

5.2 Suggestions

The method that we have experimented with still has certain limitations. First, it requires that the hand input has to be in a direct angle with the camera and the hand must be at least at a certain distance from the camera. Second, the differences in the input and output hand sizes are not considered in our experiment. The proper scaling of

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

the data to fit the output hand model will render the system more practical to several applications.

In our experiment, the input we use is obtained from the Maya animation for the correctness purpose. It will be interesting to see the input that comes from an actual video sequence. This will require a visual based tracking technique for example.

Our reflective ambiguity check considers only the tip feature points. This is because we believe that that is where the ambiguity will occur in most cases. However, to obtain more accurate result, a more sophisticated technique may be studied and applied to some other feature points.

Also, some additional constraints may improve the correctness of the result. An example is the angle-limit constraint. Moreover, some other constraints may help improve the correctness of the missing data calculation. However, the constraints can also introduce a complexity to the system and may slow down the system. So a further study is needed for this issue.

The result of our study shows that the interdependency between feature points helps improve the correctness of missing data estimation. However, we feel that further study on finding the right interdependency can help improve the result even more.

Another interesting to see is the comparison of our method to other more sophisticated methods. It would be beneficial to measure the actual tradeoffs between our method which are simple and fast with a more sophisticated method and supposedly more accurate. The study may lead to a combination of our techniques with others to create a more efficient system.

Lastly, more animations may be experimented to hopefully yield more insights on how the method performs over a wider range of motions and how it can be improved to work more accurately with them.

References

[1] P. Faloutsos, F. Pighin, and A. Shapiro, Hybrid control for interactive character animation, Proceedings of 11th Pacific Conference on Computer Graphics and Applications (PG'03), pp. 455. 2003.

[2] A. Witkin and Z. Popovic, Motion warping, Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, pp. 105-108. 1995.

[3] D. J. Wiley and J. K. Hahn, Interpolation synthesis of articulated figure motion, IEEE Computer Graphics and Applications November-December 1997, vol. 17, no. 6, pp. 39-45. 1997.

[4] Y. Li, M. Gleicher, Y. Xu, and H. Shum, Stylizing motion with drawings, Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA '03), 2003.

[5] P. Faloutsos, A. Majkowska and V. B. Zordan, Automatic splicing for hand and body animations, Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 309 – 316. 2006.

[6] E. Hsu, M. da Silva, and J. Popovic, Guided time warping for motion editing, Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 45-52. 2007.

[7] I. Albrecht, J. Haber, and H. Seidel, Construction and animation of anatomically based human hand models, Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 98-109. 2003

[8] R. Lemperriére, N. Magnenat-Thalmann and D. Thalmann, Joint-dependent local deformations for hand animation and object grasping. In Proc. Graphics Interface '88, pp. 26-33. 1998.

[9] P. Horain and H. Ouhaddi, Conception et ajustement d'un modle 3D articul de la main. In Actes des 6èmes journées du Groupe de Travail Réalité Virtuelle, volume 12/13, pp 83 -90. 1998.

[10] T. Heap and D. Hogg, 3D deformable hand models, In Proc, Gesture Workshop '96, pp. 131-139. 1996.

[11] T. Huang, J. Lin, and Y. Wu, Modeling the Constraints of Human Hand Motion. In Proc. Workshop on Human Motion, pp. 121-126. 2000.

[12] J. I. Mulero, J. Feliú Batlle and J. López Coronado, Parametric Neurocontroller for Positioning of an Antropomorc Finger Based on an Opponent-Driven Tendon Transmission System, In Proc. IWANN '01, pp. 47-54. 2001.

[13] T. Kunii and J. Lee, Model-based Analysis of Hand Posture, IEEE Computer Graphics and Applications, 15(5), pp. 77-86. 1995.

[14] L. Moccozet and N. Magnenat-Thalmann, Dirichlet Free-Form Deformations and their Application to Hand Simulation. In Proc. Computer Animation '97, pp. 93-102. 1997.

[15] J. McDonald, J. Toro, K. Alkoby, A. Berthiaume, R. Carter, P. Chomwong, J. Christopher, M. Davidson, J. Furst, B. Konie, G. Lancaster, L. Roychoudhuri, E. Sedgewick, N. Tomuro, and R. Wolfe, An improved articulated model of the human hand, The Visual Computer, 17(3), pp. 158-166. 2001.

[16] P.G. Kry, D. L. James and D. K. Pai, EigenSkin: Real Time Large Deformation Character Skinning in Hardware, In Proc. ACM SIGGRAPH Symposium on Computer Animation (SCA '02), pp. 153-159. 2002.

[17] C. Sabharwal, Recovering 3D image parameters from corresponding two 2D image, Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice, pp. 402-409. 1993.

[18] A. Cao, J. Liu, J. Snyder and X. Tang, 3D object retrieval using 2D line drawing and graph based relevance feedback, Proceedings of the 14th annual ACM international conference on Multimedia, Volume 24, Issue 3, pp. 105-108. 2005

[19] A. Saxena, S. H. Chung and A. Y. Ng, 3-D Depth Reconstruction from a Single Still Image Source, International Journal of Computer Vision, 76, Issue 1, pp. 53 - 69. 2008

[20] M. J. Park, M. G. Choi and S. Y. Shin, Human motion reconstruction from inter-

frame feature correspondences of a single video stream using a motion library. Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 113-120. 2002.

[21] M. J. Park, M. G. Choi, Y. Shinagawa and S. Y. Shin, Video-guided motion synthesis using example motions, ACM Transactions on Graphics (TOG), 25, Issue 4, pp. 1327- 1359. 2006.

[22] L. Zhang and L. Li, Human Animation from 2D Correspondence Based on Motion Trend Prediction, Computer Graphics International 2006, 546-553. 2006.

[23] L. Zhang and L. Ling, Monocular Reconstruction of Human Translation in Motion Sequence by MTA, Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia 2007, pp. 79 – 86, 2007.

[24] V. Mamania, A. Shaji and S. Chandran, Markerless Motion Capture from Monocular Videos, ICVGIP 2004, 126-132. 2004.

[25] C. J. Taylor, Reconstruction of articulated objects from point correspondences in a single uncalibrated image, In Computer Vision and Image Understanding: CVIU, vol. 80, number 3, pp. 349-363. 2000.

[26] W. Lao and J. Han, 3D Modeling for Capturing Human Motion from Monocular Video, Proc. Symposium on Information Theory in the Benelux. 2006.

[27] F. Remondino and A. Roditakis, 3D Reconstruction of Human Skeleton from Single Images or Monocular Video Sequences, 25th Pattern Recognition Symposium (DAGM 03), Lecture Notes in Computer Science, Springer 2003, pp. 100-107. 2003.

[28] H. Rijpkema and M. Girard, Computer Animation of Knowledge-Based Human Grasping, Proceedings of SIGGRAPH Conference, pp. 339-348. 1991.

[29] G. ElKoura and K. Singh, Handrix: Animating the Human Hand, Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 110-119. 2003.

[30] R. Parent, Computer Animation Algorithms and Techniques, Morgan Kaufmann

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Publishers, ISBN: 1-55860-579-7, 2002.

[31] F. Remondino and A. Roditakis, 3D Reconstruction of Human Skeleton from Single Images or Monocular Video Sequences, 25th Pattern Recognition Symposium (DAGM 03), Lecture Notes in Computer Science, Springer 2003, pp. 100-107. 2003.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Appendices

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Appendix A
Publication

The followings is the paper in the title of “3D Hand Motion Retargeting From Video Image Sequence”. It has been presented at 2010 The 2^{nd} International Conference on Computer and Automation Engineering (ICCAE 2010), February 26 – 28, 2010, Singapore.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

3D Hand Motion Retargeting From Video Image Sequence

Kosit Nopvichai   Pizzanu Kanongchaiyos
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
kositn@gmail.com   pizzanu@cp.eng.chula.ac.th

***Abstract*— This paper presents a progress on building a system to perform motion retarget of 2D hand motion from video image sequence to a 3D hand model. In our method, the orthographic projection method is used to determine the Z coordination. Additionally, information from the previous frames, interdependence of a hand model and approximate rest pose of a hand are used to deal with occlusion.**

Keywords- motion retarget, hand motion

I. INTRODUCTION

Computer animation is the science and art of using a computer to create moving images. The idea is to make a character move in a way intended by the artists and convey their creativity to the audience.

There are several ways to generate motions for an articulated character. Some of the more common are Kinematics, Dynamic control [4], Keyframing, Motion editing [28] [27] [15] [5] [8], and Motion capture. Recently more attention has been paid to an alternative to the traditional methods. It is the typical 2D video that is recorded by a typical camera or even a web cam.

There are certain advantages to this motion source. First, the source model does not need to be attached with sensors. Second, the cost is typically lower than the traditional motion capture. Third, there are enormous stocks of live action footage recorded as 2D videos. Some of them are of historic values and cannot be reproduced. An example is a number of classic sport moments. This can be readily used as a motion source.

Using 2D video as a source of motions has a few challenges of its own that need to be addressed. First, the missing data (e.g. those caused by occlusion) need to be somehow recovered. Specific to hand motions, we may consider using interdependence in addition to constraints, motion library, sample space, etc. Second, the 2D nature of it necessitates the lack of depth information. Thus some variants of 3D reconstruction techniques are used to recover the missing Z coordinate. We will address these issues in our work.

After a motion is acquired through one of the means mentioned above and stored in a motion representation, a typical motion retarget proceeds. As part of the process, an acquired raw motion is typically processed in some ways to create a more appropriate motion for each target character. The output of this step is the adapted motion data used to drive the target motion. For the case of an articulated figure, the output is usually joint angle data for all the joints.

The animation of human articulate body has long been received numerous attentions. The works in this area vary in terms of the body parts on which they focus. As for the hand, it has been a focus of many researches in computer animation because not only it is one of the most animated parts of human body but also one of the most complex body parts. In addition it is essential for human communication and expression. Our work will focus on retargeting the hand motion from 2D monocular video sequence to a 3D hand model.

In summary, the aim of this work is to perform motion retarget by using the motions from the 2D monocular video sequence which is an alternative to the traditional motion capture. This work will focus on motions of the human hand. The expected end product is a software system that is capable of retargeting hand motions from 2D video sequence to a 3D hand model. The motion input will be 2D video sequence of hand gestures from a monocular video camera. The output will be the animation of the deformed hand.

II. RELATED WORK

A. Hand Model

Hand anatomy has long been studied and well understood in the field of anatomy and biomechanics [1]. Hand is one of the most complex body parts. Most animation research focuses on its two main functionalities which are grasping and fine motor skills. Many aspects have been studied such as its constraints, limitations, DOFs, bones, tendons, and muscles. Several hand models have been proposed over the years. Examples are [13], [7], [6], [9], [19], [11], [18], [17] [10], etc. Each has its own strengths and weaknesses. Whichever one we should use depends on the task at hand. More closely related to our work are [19] and [11]. In particular, they also consider inter-joint dependencies.

Our hand model will be a relatively simple kinematic chain consisting of joints and segments. Each joint has a number of DOFs and limitations. Also, interdependence between finger joints will be used. More details are further explained in the Methods section.

B. Depth Reconstruction

Depth reconstruction refers to the process of extracting the depth information from 2D data. Its challenge lies in the fact that it is an under-determined problem. To solve it, we need to pose some constraints or use some assumptions and find a solution under that framework.

Study on 3D Depth recovery from 2D input has been performed for some time. There have been several techniques proposed. Reference [24] proposes an algorithm to compute the three dimensional structure of a scene from a pair of stereo images. Reference [2] constructs a 3D object query from 2D drawings. Their algorithm can handle objects with both planar and curved faces. Reference [25] estimates 3D depth from a single still image. It proposes the use of monocular cues (e.g., texture variations and gradients, defocus, color/haze, etc.) in addition to the stereo cues.

More recently, Reference [21] and [22] reconstruct a human-like figure motion from 2D video stream. They assume an existence of a library of motions similar to the target motion video stream and assume the length of each segment is known. A library of motions that are similar to the target motions is used to provide a reference frame that will be warped based on the target frame to get the final pose. A technique based on Motion Trend Analysis has been proposed in [29] and [30]. The method uses the information solved in the previous frame to solve for the next frame except the first frame. Reference [16] exploits the domain specific knowledge about the target motions to find certain joint locations and to limit possible poses. Reference [26], [14], [23], and [16] use the orthographic projection method to determine the Z coordination.

To derive the Z coordinate from a single image, they assume the point corresponding and segment lengths are known and the certain distance between object and the camera are maintained. The problem of standard reflective ambiguity is also mentioned and resolved mostly with constraints. Reference [23] improves upon [26] by allowing some perspective cases to work properly.

Our method is similar to the one described in [26] which uses the scaled orthographic projection model. However, our system intends to work with a video sequence instead of a single image. Moreover, occlusion is also considered in our work.

C. Interdependence

Interdependence refers to the influence of a finger joint on others. Each finger joint is not fully independent but to some degree depend on the movement of some other joints on the hand. This can be viewed as dependence constraints between the joints of each finger and between fingers. This concept has been studied and used in several works. Reference [31] observes that naturally a DIP joint cannot be moved without moving the PIP joint of the same finger. In another word there is a dependency between them. The reference [31] approximates the relationship between the two joint angle to be DIP = 2/3 PIP. They use this dependency to reduce the number of DOF by making DIP fully depend on PIP. Reference [12] uses interdependence in their work. Reference [3] expands the idea by assigning the degree of dependency between each joint across fingers.

III. METHODS

A. Input Acquisition

Our retarget system will need two inputs from the user

- The length of each segment. We need the length of each segment for 3D depth reconstruction.
- The feature points (XY coordinates) of a hand in a video frame from a 2D monocular video sequence of hand gestures. In our experiment, a 3D hand model will be created and animated using Maya software. Then we write a MEL script to extract the XYZ coordinates of each feature point in each frame. The XY part will be used

as the input to our experimental system. A benefit to this method is that we will also have the Z coordinate to verify our result.

B. *Our Hand Model*

Our retarget system retargets input hand motion to a 3D hand model. The specification of our 3D target hand model (Fig. 1) is as follows:

- There are 16 joints and 22 degrees of freedom (DOF) in each hand and wrist
- The wrist has two DOFs
- Each finger except thumb has three joints and sum up to 16 DOFs in a hand

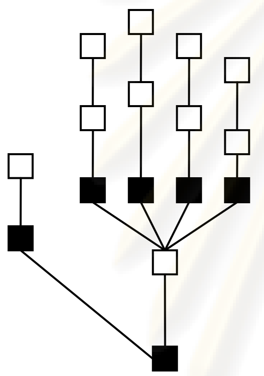

Figure 1. shows the DOFs of each of the joint in our hand model. The black node has 2 DOFs. The white node has 1 DOF.

C. *Feature Points Identification (XY Coordinates)*

For each input image sequence of a hand gesture, we assume that the locations of all feature points (Fig. 2) are available to us (unless they are occluded). A feature point in our case includes the location of a joint in each finger and the wrist location. The location will be specified as the XY coordinates of the following locations:

- 5 tips of Thumb, Index, Middle, Ring and Little fingers
- 4 Distal interphalangeal joints (DIP) of Index, Middle, Ring and Little fingers

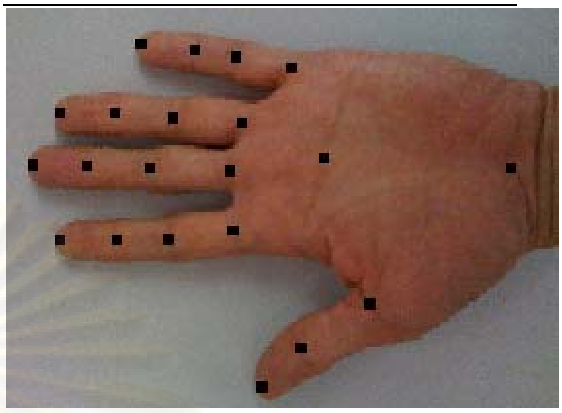

Figure 2. shows the feature point locations of our hand model.

- 1 Interphalangeal joints (IP) of Thumb
- 4 Proximal interphalangeal joints (PIP) of Index, Middle, Ring, and Little fingers
- 5 Metacarpophalangeal joints (MCP) of Thumb, Index, Middle, Ring and Little fingers
- 1 fold of the palm
- 1 wrist

In some images, it may be impossible to identify all of these feature point locations because of occlusion. In such cases, we will need some technique to approximate their locations. These techniques are interdependence, previous frame data and constraints. Also, one assumption is that if a feature point is occluded, probably its exact location is irrelevant in that context and it should be able to be estimated by its rest pose which is approximately somewhere in the middle of its range (in case of a joint) [20].

D. *3D Depth Reconstruction*

Since our input is a sequence of 2D images, the information we get for each feature point is 2D. Thus, we need a way to compute for the Z coordinate. To do this, we adopt the method in [26] which uses the scaled orthographic projection model. A projection of a point (x, y, z) in three-dimensional space to the point (x, y, 0) on the x-y plane can be represented as a matrix (1).

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (1)$$

In scaled orthographic projection, we simply add a scale factor, s, (2). This results in a simple scaling of the object coordinates. The scaled-orthographic model amounts to parallel projection, with a scaling added to mimic the effect that the image of an object shrinks with the distance [23].

$$\begin{pmatrix} u \\ v \end{pmatrix} = s \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \qquad (2)$$

The formula is expressed in (4). The followings show the derivation of (4). l denotes the segment length between point 1 and 2. X ,Y ,Z are the actual coordinates. u , v are the scaled X and Y respectively. s is the scale factor.

$$l^2 = (X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2$$

$$(u_1 - u_2) = s(X_1 - X_2)$$

$$(v_1 - v_2) = s(Y_1 - Y_2)$$

$$(Z_1 - Z_2) = \sqrt{(l^2 - ((u_1 - u_2)^2 + (v_1 - v_2)^2) / s^2}$$

$$u = s \cdot X$$
$$v = s \cdot Y \qquad (3)$$

$$(Z_1 - Z_2)^2 = l^2 - [(u_1 - u_2)^2 + (v_1 - v_2)^2] / s^2 \qquad (4)$$

$$s \geq \frac{\sqrt{[(u_1 - u_2)^2 + (v_1 - v_2)^2]}}{l} \qquad (5)$$

From (4) we assume an arbitrary depth (e.g. 0) for Z_1 and compute for Z_2. In this case, we also know u_1, u_2, v_1, v_2, and l. If we also know s, the scale factor, then we will be able to solve for Z_2. In our case we assume that the distance between the camera and the hand is much greater than the depth of Z coordinate. (Note that this assumption is needed for the scaled orthographic projection model to work.) With this assumption, the scale factor is almost constant for all the joints on the hand. So we can use the same scale value for all the feature points. Now to compute for the scale factor, s, we use (5) to find the overall minimum value of s. Note that (5) comes from the fact that (4) has a real solution. We will use the minimum overall value of s in our computation since the absolute values of X, Y and Z are not necessary. All we need is the relative depth between each feature point. Once we obtain s, we can use (3) to find the value of X and Y. We then use the computed Z_2 as the Z_1 of the next segment. We then repeat this process until all feature points are computed. One issue that we still have is the reflective ambiguity. This stems from the fact that the Z_1 or Z_2 in (4) can be the smaller one based on the 2D information we have. In our case, joint angle limit, physiological constraints are used to pick the more likely configuration.

From this step, we can obtain XYZ coordinates of feature points. These values are used to compute the joint angle data for each joint. However, in the case where the source and target model have different scale, we need to scale this coordinates data to the correct value before they can be used to compute the joint angle.

E. Interdependence

The purpose of using the interdependence in this work is two fold. Firstly, by taking the interdependence into account, the finger movement is more realistic. Secondly, the interdependence in conjunction with the coordinate and joint angle data help us fill in the missing data in case of a joint occlusion. We implement it as a dependency list of joints. The entry of this list will contain a joint ID and the list of its dependent joints together with the amount of dependency. For example,

Index PIP: Index DIP (50), Middle PIP (25), Ring PIP (15)

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

This entry says that if the Index Pip is moving x points, the Index DIP should be moving 1/2x points, the Middle PIP should be moving 1/4x points if no other force is exerted upon them.

The exact number and amount of dependence between each joint are studied from other research works such as [31], [12], [3], and our own observation. We plan to assign a default set of joint interdependence. But a user can optionally fine tune these values.

F. Constraint Identification

In addition to the joint angle and physiological constraints, another constraint is needed to make sure the end effectors are at the right position. For example, in a pose where the tip of thumb and the tip of index finger are touching, this fact should be enforced at the target hand as well.

To determine "coincident" constraint, we use the XYZ coordinate of the feature points and a threshold. If the distance between any feature points is less than the threshold, we will consider them touching. The exact value of the threshold will be determined later.

G. Joint Angle Data Calculation & Retargeting

The inverse kinematics is used to calculate the joint angle data given the XYZ coordinates of a desired pose obtained from the 2D input data and 3D depth reconstruction.

Since a hand model is fairly complex, the incremental approach of inverse Jacobian is used instead of the analytic approach.

From this step, we will get the joint angle data for all the joints ready to be retargeted to our 3D hand model.

IV. Result Evaluations

The result of the retarget will be evaluated by comparing the result of our calculation with the data retrieved from Maya software.

V. Conclusion

We have described a technique to retarget a 2D video sequence to a 3D hand model. The working horse in our techniques is the orthographic projection method which is used to determine the Z coordination. The occlusion is also tackled with the interdependence, previous frame data, and natural rest pose of a hand.

We expect that our experiment on applying a variety of techniques to build a working system for hand motion retarget from 2D video input will afford us to find out how well these techniques are working in practice and hopefully to discover some new insights based on the experience of building such systems that will be beneficial to others attempting similar tasks in the future. Moreover, we hope that our system can be used to generate interesting hand animation from 2D video. Some of the potential applications are sign language interpreter, movie and game industry, etc.

References

[1] I. Albrecht, J. Haber, and H. Seidel, "Construction and animation of anatomically based human hand models," Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2003, pp. 98-109 ISBN ~ ISSN5288-1727 :,-58113-1 5-659.

[2] A. Cao, J. Liu, J. Snyder, and X. Tang, "3D object retrieval using 2D line drawing and graph based relevance feedback," Proceedings of the 14th annual ACM international conference on Multimedia, vol. 24, issue 3, 2005, pp. 105-108.

[3] G. ElKoura and K. Singh, "Handrix: animating the human hand," Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2003, pp. 110-119, ISBN ~ ISSN5288-1727 :,5-659-58113-1 .

[4] P. Faloutsos, F. Pighin, and A. Shapiro, "Hybrid control for interactive character animation," Proceedings of 11th Pacific Conference on Computer Graphics and Applications (PG'03), 2003, pp. 455.

[5] P. Faloutsos, A. Majkowska and V. B. Zordan, "Automatic splicing for hand and body animations," Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2006, pp. 309 – 316.

[6] T. Heap and D. Hogg, "3D deformable hand models," In Proc, Gesture Workshop '96, 1996, pp. 131-139.

[7] P. Horain and H. Ouhaddi, "Conception et ajustement d'un modle 3D articul de la main," In Actes des 6èmes journées du Groupe de Travail Réalité Virtuelle, vol. 12/13, 1998, pp. 83 -90.

[8] E. Hsu, M. da Silva, and J. Popovic, "Guided time warping for motion editing," Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2007, pp. 45-52.

[9] T. Huang, J. Lin, and Y. Wu, "Modeling the constraints of human hand motion," In Proc. Workshop on Human Motion, 2000, pp. 121-126.

[10] P. G. Kry, D. L. James, and D. K. Pai, "EigenSkin: real time large deformation character skinning in hardware," In Proc. ACM SIGGRAPH Symposium on Computer Animation (SCA '02), 2002, pp. 153-159.

[11] T. Kunii, Y. Tsuchida, H. Matsuda , M. Shirahama, and S. Miura, "A model of the

hands and arms based on manifold mappings," In Proc. Computer Graphics International (CGI '93), 1993, pp. 381-398.

[12] T. Kunii and J. Lee, "Model-based analysis of hand posture," IEEE Computer Graphics and Applications, 15(5), 1995, pp. 77-86.

[13] R. Lemperriére, N. Magnenat-Thalmann, and D. Thalmann, "Joint-dependent local deformations for hand animation and object grasping," In Proc. Graphics Interface '88, 1988, pp. 26-33.

[14] W. Lao and J. Han, "3D modeling for capturing human motion from monocular video," Proc. Symposium on Information Theory in the Benelux, 2006.

[15] Y. Li, M. Gleicher, Y. Xu, and H. Shum, "Stylizing motion with drawings," Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA '03), 2003.

[16] V. Mamania, A. Shaji, and S. Chandran, (2004) "Markerless motion capture from monocular videos," ICVGIP 2004, pp. 126-132.

[17] J. McDonald, J. Toro, K. Alkoby, A. Berthiaume, R. Carter, P. Chomwong, J. Christopher, M. Davidson, J. Furst, B. Konie, G. Lancaster., L. Roychoudhuri, E. Sedgewick, N. Tomuro and R. Wolfe, "An improved articulated model of the human hand," The Visual Computer, 17(3), 2001, pp. 158-166.

[18] L. Moccozet and N. Magnenat-Thalmann, "Dirichlet free-form deformations and their application to hand simulation," In Proc. Computer Animation '97, 1997, pp. 93-102.

[19] J. I. Mulero, J. Feliú Batlle, and J. López Coronado, "Parametric neurocontroller for positioning of an antropomorc finger based on an opponent-driven tendon transmission system," In Proc. IWANN '01, 2001, pp. 47-54.

[20] R. Parent, Computer Animation Algorithms and Techniques, Morgan Kaufmann Publishers, ISBN: 1-55860-579-7, 2002.

[21] M. J. Park, M. G. Choi, and S. Y. Shin, "Human motion reconstruction from inter-frame feature correspondences of a single video stream using a motion library," Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2002, pp. 113-120.

[22] M. J. Park, M. G. Choi, Y. Shinagawa, and S. Y. Shin, "Video-guided motion synthesis using example motions," ACM Transactions on Graphics (TOG), vol. 25, issue 4, 2006, pp. 1327- 1359.

[23] F. Remondino and A. Roditakis, "3D reconstruction of human skeleton from single images or monocular video sequences," 25th Pattern Recognition Symposium (DAGM 03), Lecture Notes in Computer Science, Springer 2003, pp. 100-107.

[24] C. Sabharwal, "Recovering 3D image parameters from corresponding two 2D image," Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing :states of the art and practice, 1993, pp. 402-409 ISBN4-567-89791-0 :.

[25] A. Saxena, S. H. Chung, and A. Y. Ng, "3-D depth reconstruction from a single still image source," International Journal of Computer Vision, 76, issue 1, 2008, pp. 53-69.

[26] C. J. Taylor "Reconstruction of articulated objects from point correspondences in a single uncalibrated image," In Computer Vision and Image Understanding: CVIU, vol. 80, number 3, 2000, pp. 349-363.

[27] D. J. Wiley and J. K. Hahn, (1997) "Interpolation synthesis of articulated figure motion," IEEE Computer Graphics and Applications November-December 1997, vol. 17, no. 6, pp. 39-45.

[28] A. Witkin and Z. Popovic, "Motion warping," Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, 1995, pp. 105-108.

[29] L. Zhang and L. Li, "Human animation from 2D correspondence based on motion trend prediction," Computer Graphics International 2006, pp. 546-553.

[30] L. Zhang and L. Ling, "Monocular reconstruction of human translation in motion sequence by MTA," Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia 2007, pp. 79-86.

[31] H. Rijpkema and M. Girard, "Computer Animation of Knowledge-Based Human Grasping," Proceedings of SIGGRAPH Conference, July 1991, pp. 339-34

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Appendix B
Z Order Computation Program

This program computes the Z coordinate of all feature points on a hand in a frame. It is written in C++ using Visual Studio 2008. The input is the list of X, Y coordinates of the feature points on a hand for 100 frames. The program computes the corresponding Z coordinates of feature points on a hand of each frame. The output is written to a file.

```
#include <iostream>
#include <limits>
#include <cmath>
#include <map>
#include <vector>
#include <fstream>
#include <string>

using namespace std;


class MyException
{
     string m_msg;
     int m_num;

public:
     MyException(const string& msg, int num)
     {
          m_msg = msg;
          m_num = num;
     }

     void print()
     {
          cerr << m_msg << " (" << m_num << ")" << endl;
     }

     static const int EXC_OUT_OF_RANGE = 1;
};
////////////////////////////////////////////////
class Vertex
{
public:
  double x, y, z; //actual x, y, z
  double u, v, w;    // observed (scaled) x and y
  int id;
  bool m_uvSetFlag;

  public:
    Vertex():x(numeric_limits<double>::min()),
              y(numeric_limits<double>::min()),
              z(0), u(numeric_limits<double>::min()),
              v(numeric_limits<double>::min()),
              w(numeric_limits<double>::min()),
              id(-7777), m_uvSetFlag(false) {}
```

```
    Vertex(int iid):x(numeric_limits<double>::min()),
          y(numeric_limits<double>::min()),
          z(0),
          u(numeric_limits<double>::min()),
          v(numeric_limits<double>::min()),
          w(numeric_limits<double>::min()),
          id(iid),
          m_uvSetFlag(false) {}
    Vertex(int iid, double uu, double
vv):x(numeric_limits<double>::min()),
          y(numeric_limits<double>::min()),
          z(0),
          u(uu),
          v(vv),
          id(iid),
          m_uvSetFlag(false) {}

    void print()
    {
     cerr << "id = " << id << " u = "
              << u << " v = " << v
              << " w = " << w
              << " x = " << x
              << " y = " << y
              << " z = " << z << endl;
    }

    void setUV(double uu, double vv)
    {
     u = uu;
     v = vv;
     m_uvSetFlag = true;
    }

    /**
     * This will only work if setUVSetFlag is called properly
     * when U and V are set.
     */
    bool isUVSet() const
    {
     return m_uvSetFlag;
    }

    void setUVSetFlag(bool v)
    {
     m_uvSetFlag = v;
    }

};

////////////////////////////////////////////////
/**
 * Define our hand model
 * - how many feature points in the hand
 * - segment length of each segment
 */

/****************************************************
 *
 *
```

```
 *
 *                  |    |     |     |
 *                 3|   6|    9|   12|
 *                  |    |     |     |
 *                 4|   7|   10|   13|
 *                  |    |     |     |
 *             0|  5|   8|   11|   14|
 *              |
 *             1|         -16 palm fold
 *              |
 *             2|
 *                        -15 wrist
 *
 *
 *
 *
 *                  |    |     |     |
 *                 3|   7|   11|   15|
 *                  |    |     |     |
 *                 4|   8|   12|   16|
 *                  |    |     |     |
 *                 5|   9|   13|   17|
 *                  |    |     |     |
 *                 6|  10|   14|   18|
 *             0|
 *              |
 *             1|         -20 palm fold
 *              |
 *             2|
 *                        -19 wrist
 *
 *  - The reference point (the first point to compute Z order is 20
 *  - 2,20
 *  - 6,20
 *  - 10,20
 *  - 14,20
 *  - 18,20
 *  - 6, 10, 14, 18 bend forward only (no sideward or backward)
 *  - 2 bends in toward 20 only
 */

class HandModel
{
private:
     static map<int, int> m_associateVertices;
     static map<pair<int, int>, double > m_segmentLengths;
     static map<int, int> m_ZCoordinateComputeOrder;
     static map<int, int> m_interdepNeighbors;
     static map<int, int> m_interdepIntraFingerNeighbors;


public:
     const static int NUM_FEATURE_POINTS = 21;
     const static int PALM_FOLD_INDEX = 20;
     const static int PIVOT_POINT = 20;
     const static int WRIST_INDEX = 19;

     static void init()
     {
               m_associateVertices[0] = 1;
               m_associateVertices[1] = 2;
```

```
		m_associateVertices[2] = PALM_FOLD_INDEX;

		m_associateVertices[3] = 4;
		m_associateVertices[4] = 5;
		m_associateVertices[5] = 6;
		m_associateVertices[6] = PALM_FOLD_INDEX;

		m_associateVertices[7] = 8;
		m_associateVertices[8] = 9;
		m_associateVertices[9] = 10;
		m_associateVertices[10] = PALM_FOLD_INDEX;

		m_associateVertices[11] = 12;
		m_associateVertices[12] = 13;
		m_associateVertices[13] = 14;
		m_associateVertices[14] = PALM_FOLD_INDEX;

		m_associateVertices[15] = 16;
		m_associateVertices[16] = 17;
		m_associateVertices[17] = 18;
		m_associateVertices[18] = PALM_FOLD_INDEX;

		m_associateVertices[19] = 2;
		m_associateVertices[20] = -1; //mean its own z coordinate
is 0

		/////////////////////////////////////////////
		m_segmentLengths[pair<int, int>(0, 1)] = 2.391958;
		m_segmentLengths[pair<int, int>(1, 2)] = 2.092683;
		m_segmentLengths[pair<int, int>(2, 1)] = 2.092683;

		m_segmentLengths[pair<int, int>(3, 4)] = 1.70821;
		m_segmentLengths[pair<int, int>(4, 5)] = 1.83695;
		m_segmentLengths[pair<int, int>(5, 6)] = 2.430827;
		m_segmentLengths[pair<int, int>(6, 5)] = 2.430827;

		m_segmentLengths[pair<int, int>(7, 8)] = 2.109315;
		m_segmentLengths[pair<int, int>(8, 9)] = 2.017658;
		m_segmentLengths[pair<int, int>(9, 10)] = 2.29072;
		m_segmentLengths[pair<int, int>(10, 9)] = 2.29072;

		m_segmentLengths[pair<int, int>(11, 12)] = 1.719452;
		m_segmentLengths[pair<int, int>(12, 13)] = 2.559455;
		m_segmentLengths[pair<int, int>(13, 14)] = 1.914169;
		m_segmentLengths[pair<int, int>(14, 13)] = 1.914169;

		m_segmentLengths[pair<int, int>(15, 16)] = 1.422462;
		m_segmentLengths[pair<int, int>(16, 17)] = 1.363195;
		m_segmentLengths[pair<int, int>(17, 18)] = 1.198547;
		m_segmentLengths[pair<int, int>(18, 17)] = 1.198547;

		m_segmentLengths[pair<int, int>(19, 2)] = 3.655157;
		m_segmentLengths[pair<int, int>(PALM_FOLD_INDEX, 19)] =
3.755123;

		m_segmentLengths[pair<int, int>(2, PALM_FOLD_INDEX)] =
3.755729;
		m_segmentLengths[pair<int, int>(6, PALM_FOLD_INDEX)] =
2.81398;
		m_segmentLengths[pair<int, int>(10, PALM_FOLD_INDEX)] =
1.719374;
```

```
		m_segmentLengths[pair<int, int>(14, PALM_FOLD_INDEX)] = 
2.078661;
		m_segmentLengths[pair<int, int>(18, PALM_FOLD_INDEX)] = 
2.968683;
		///////////////////////
		m_ZCoordinateComputeOrder[0] = PALM_FOLD_INDEX; //or 
should be first ones?

		m_ZCoordinateComputeOrder[1] = 2;
		m_ZCoordinateComputeOrder[2] = 1;
		m_ZCoordinateComputeOrder[3] = 0;

		m_ZCoordinateComputeOrder[4] = 6;
		m_ZCoordinateComputeOrder[5] = 5;
		m_ZCoordinateComputeOrder[6] = 4;
		m_ZCoordinateComputeOrder[7] = 3;

		m_ZCoordinateComputeOrder[8] = 10;
		m_ZCoordinateComputeOrder[9] = 9;
		m_ZCoordinateComputeOrder[10] = 8;
		m_ZCoordinateComputeOrder[11] = 7;

		m_ZCoordinateComputeOrder[12] = 14;
		m_ZCoordinateComputeOrder[13] = 13;
		m_ZCoordinateComputeOrder[14] = 12;
		m_ZCoordinateComputeOrder[15] = 11;

		m_ZCoordinateComputeOrder[16] = 18;
		m_ZCoordinateComputeOrder[17] = 17;
		m_ZCoordinateComputeOrder[18] = 16;
		m_ZCoordinateComputeOrder[19] = 15;

		m_ZCoordinateComputeOrder[PALM_FOLD_INDEX] = 19; //or 
should be first ones?

		//////////////////////////////////////////////////
		m_interdepNeighbors[0] = 0;
		m_interdepNeighbors[1] = 1;
		m_interdepNeighbors[2] = 2; //-1; //mean its own z 
coordinate is 0

		m_interdepNeighbors[3] = 7;
		m_interdepNeighbors[4] = 8;
		m_interdepNeighbors[5] = 9;
		m_interdepNeighbors[6] = 10;

		m_interdepNeighbors[7] = 11;
		m_interdepNeighbors[8] = 12;
		m_interdepNeighbors[9] = 13;
		m_interdepNeighbors[10] = 14;

		m_interdepNeighbors[11] = 15;
		m_interdepNeighbors[12] = 16;
		m_interdepNeighbors[13] = 17;
		m_interdepNeighbors[14] = 18;

		m_interdepNeighbors[15] = 11;
		m_interdepNeighbors[16] = 12;
		m_interdepNeighbors[17] = 13;
		m_interdepNeighbors[18] = 14;
```

```
                m_interdepNeighbors[19] = 19;
                m_interdepNeighbors[20] = 20; //mean its own z coordinate
is 0


                //////////////////////////////////////////////////
                m_interdepIntraFingerNeighbors[0] = 1;
                m_interdepIntraFingerNeighbors[1] = 2;
                m_interdepIntraFingerNeighbors[2] = 2;


                m_interdepIntraFingerNeighbors[3] = 4;
                m_interdepIntraFingerNeighbors[4] = 5;
                m_interdepIntraFingerNeighbors[5] = 6;
                m_interdepIntraFingerNeighbors[6] = 6;

                m_interdepIntraFingerNeighbors[7] = 8;
                m_interdepIntraFingerNeighbors[8] = 9;
                m_interdepIntraFingerNeighbors[9] = 10;
                m_interdepIntraFingerNeighbors[10] = 10;

                m_interdepIntraFingerNeighbors[11] = 12;
                m_interdepIntraFingerNeighbors[12] = 13;
                m_interdepIntraFingerNeighbors[13] = 14;
                m_interdepIntraFingerNeighbors[14] = 14;

                m_interdepIntraFingerNeighbors[15] = 16;
                m_interdepIntraFingerNeighbors[16] = 17;
                m_interdepIntraFingerNeighbors[17] = 18;
                m_interdepIntraFingerNeighbors[18] = 18;

                m_interdepIntraFingerNeighbors[19] = 19;
                m_interdepIntraFingerNeighbors[20] = 20;



        }

        static double findSegmentLength(const Vertex& v1, const Vertex& v2)
        {
                cerr << "findSegmentLength for (" << v1.id << ", "
                        << v2.id << ") is " << m_segmentLengths[pair<int,
int>(v1.id,v2.id)]
                        << endl;
                return m_segmentLengths[pair<int, int>(v1.id,v2.id)];
        }

        /**
         * Returns the vertex associated with vertex v.
         * By association, we mean the vertex that together with v
         * defines a segment length
         */
        static int findAssociateVertex(Vertex v)
        {
                cerr << "findAssociateVertex for " << v.id
                         << " is " << m_associateVertices[v.id] << endl;
                return m_associateVertices[v.id];
        }

        static int findNeighborId(Vertex v)
        {
```

```
        cerr << "findNeighborId for " << v.id
              << " is " << m_interdepNeighbors[v.id] << endl;
        return m_interdepNeighbors[v.id];
    }

    static int findIntraFingerNeighborId(Vertex v)
    {
        cerr << "findIntraFingerNeighborId for " << v.id
             << " is " << m_interdepIntraFingerNeighbors[v.id] << endl;
        return m_interdepIntraFingerNeighbors[v.id];
    }

    /**
     * Returns the feature point to calculate at the order i th
     */
    static int findZCoordinateComputeOrder(int i)
    {
        cerr << "findZCoordinateComputeOrder for "
             << i << " is " << m_ZCoordinateComputeOrder[i] << endl;
        return m_ZCoordinateComputeOrder[i];
    }

};

map<int, int>  HandModel::m_associateVertices;
map<pair<int, int>, double > HandModel::m_segmentLengths;
map<int, int> HandModel::m_ZCoordinateComputeOrder;
map<int, int>  HandModel::m_interdepNeighbors;
map<int, int>  HandModel::m_interdepIntraFingerNeighbors;


////////////////////////////////////////////////
/* the data for each frame */
class Frame {

public:
    Frame():m_scale(numeric_limits<double>::max()),
        m_restedPalmScale(numeric_limits<double>::max()),
        m_id(-1) {}
    Frame(int id):m_scale(numeric_limits<double>::max()),
        m_restedPalmScale(numeric_limits<double>::max()),
        m_id(id) {}

private:
    Vertex m_featurePoints[HandModel::NUM_FEATURE_POINTS];
    double m_scale;
    double m_restedPalmScale;
    int m_id;


private:
    //helper
    double pow2(double d) { return pow (d, 2); }

public:
    int getId() { return m_id; }
    void setId(int id) { m_id = id; }

    void print()
    {
        cerr << "feature points: " << endl;
```

```
        for(int i=0; i< HandModel::NUM_FEATURE_POINTS; ++i)
        {
            m_featurePoints[i].print();
        }
        cerr << "scale: " << m_scale << endl;

    }
    Vertex& getFpRef(int index)
    {
        cerr << "entering Frame::getFpRef\n";
        if (index < 0 || index >= HandModel::NUM_FEATURE_POINTS)
        {
            cerr << "error: out of range\n";
            throw MyException("out of range",
MyException::EXC_OUT_OF_RANGE);
         }

        return m_featurePoints[index];
    }

    void setfp(int index, const Vertex& v)
    {
        cerr << "entering Frame::setfp " << "(" << this->getId() <<
")"
            << index << " " << v.u << ", "
            << v.v << "fp id is " <<  v.id << "\n";
        if (index < 0 || index >= HandModel::NUM_FEATURE_POINTS)
        {
            cerr << "error: out of range\n";
            return;
        }
        m_featurePoints[index] = v;
    }

  /**
   * Find only once per frame.
   * We reuse the same scale factor for all reference points in the
frame
   *
   * OUTPUT: m_scale is set if not already
   */
  double findMinimumScale()
  {
      cerr << "entering Frame::findMinimumScale\n";

      if (m_scale == numeric_limits<double>::max()) //first time check
      {
          // equation 8
          // Find the minimum overall scale over all reference point
pairs
          for(int i=0; i< HandModel::NUM_FEATURE_POINTS; ++i)
          {
              Vertex vertex1 = m_featurePoints[i];
              Vertex vertex2 =
m_featurePoints[HandModel::findAssociateVertex(vertex1)];

              const double l = HandModel::findSegmentLength(vertex1,
vertex2);
              cerr << "Frame::findMinimumScale(): the current segment
length is " << l << endl;
```

```
                double s = sqrt(pow2(abs(vertex1.u-vertex2.u)) +
pow2(abs(vertex1.v-vertex2.v))) / l;

                cerr << "Frame::findMinimumScale(): the current scale is
" << s << endl;

                //keep minimum over all
                if (s < m_scale)
                {
                    m_scale = s;
                }
            }
        }

        cerr << "exiting Frame::findMinimumScale(): the minimum scale is
" << m_scale << endl;
        return m_scale;
    }



    /**
     * Compute z coordinates of all feature points (of this frame)
     *
     * output: x, y, z of all feature points
     * outf: the output file
     */
    void computeZCoordinates(ofstream& outf)   //output: Z coordinates
    {
        for (int i = 0; i < HandModel::NUM_FEATURE_POINTS; ++i)
        {
            //doComputeZCoordinate(i);
            int j = HandModel::findZCoordinateComputeOrder(i);
            doComputeZCoordinate(j, outf);
        }
    }

    double getScaleBasedOnRestedPalm()
    {
        return 1;

        //input
        //segment length of palm
        //observed x,y of the two end points of palm
        if(m_restedPalmScale != numeric_limits<double>::max())
            return m_restedPalmScale;

        // equation 8
        Vertex vertex1 = m_featurePoints[HandModel::PALM_FOLD_INDEX];
        Vertex vertex2 = m_featurePoints[HandModel::WRIST_INDEX];
        const double l = HandModel::findSegmentLength(vertex1, vertex2);
        cerr << "Frame::getScaleBasedOnRestedPalm(): the current segment
length is " << l << endl;

        double s = l / sqrt(pow2(abs(vertex1.u-vertex2.u)) +
pow2(abs(vertex1.v-vertex2.v)));

        cerr << "Frame::getScaleBasedOnRestedPalm(): the scale is " << s
<< endl;
        return m_restedPalmScale = s;
    }
```

```
  double  findScaledSegmentlength(const Vertex& v1, const Vertex& v2,
double scale)
  {
       double segmentLength = HandModel::findSegmentLength(v1, v2);
       cerr << "findScaledSegmentlength() segmentLength: " <<
segmentLength << ", scale:" << scale << "= " << segmentLength/scale <<
endl;
       return segmentLength/scale;
  }

  /**
     * find the Z coordinate for the feature point i
     *
     * input: u, v of feature point i
     * output: x, y and z of feature point i
     */
    void doComputeZCoordinate(int i, ofstream& outf)
    {
       cerr << "entering Frame::doComputeZCoordinate\n";

       Vertex& vertex1 = m_featurePoints[i];
       Vertex& vertex2 =
m_featurePoints[HandModel::findAssociateVertex(vertex1)];

       // special case for the first feature point
       if(vertex1.id == HandModel::PIVOT_POINT)
       {
            const double s = getScaleBasedOnRestedPalm();
            vertex1.w = 0;
            vertex1.w = -0.219004; // <--- !!!! hard code with the
actual value
            vertex1.x = vertex1.u / s;
            vertex1.y = vertex1.v / s;
            vertex1.z = 0; // <--- hard code to 0
            vertex1.z = -0.219004;



            ////////////////////////////////////////////////////////
            //
            // FORMAT:
            // j0 32 -4.92007 -1.23411 1.3899
            //
            outf << "j" << vertex1.id << " " // node name e.g. "j0"
                 << this->getId() << " "        // frameId
                 << vertex1.x << " "
                 << vertex1.y << " "
                 << vertex1.z
                 << endl;

            return;
       }

       //find the scaled segment length
       double l = findScaledSegmentlength(vertex1, vertex2,
getScaleBasedOnRestedPalm());

       // check first if its gonna be a negative value (which cannot be
sqrt'ed)
```

```
        while (( pow2(l) - pow2(abs(vertex1.u-vertex2.u)) -
pow2(abs(vertex1.v-vertex2.v)) ) < 0)
        {
            cerr << "WARNING: length is adjusted (+0.001) before (" << l
<< ") after (" << l+0.001 << ")" << endl;
            // adjust the length segment length until the value is
positive
            l += 0.001;
        }

        vertex1.w = sqrt( pow2(l) - pow2(abs(vertex1.u-vertex2.u)) -
pow2(abs(vertex1.v-vertex2.v)) ) + vertex2.w;

        // Tip
        if (vertex1.id == 0 || vertex1.id == 3 || vertex1.id == 7 ||
vertex1.id == 11 || vertex1.id == 15)
        {
            // what we do here is using the angle ABC to determine
whether D's z should be less than C's z
            // if the ABC is < 90 degree then D should be point toward
the palm
            Vertex& vertexB =
m_featurePoints[HandModel::findAssociateVertex(vertex2)];
            Vertex& vertexA =
m_featurePoints[HandModel::findAssociateVertex(vertexB)];
            //                    A         B         C
            double ag = angle(vertexA, vertexB, vertex2); //get angle at
B
            if (0  < ag && ag <= 90)
            {
                vertex1.w = vertex2.w - ( sqrt( pow2(l) -
pow2(abs(vertex1.u-vertex2.u)) - pow2(abs(vertex1.v-vertex2.v)) ) );
            }
            else
            {
                cerr << "YYYY point away from the palm" << endl;
            }
        }

        const double s = getScaleBasedOnRestedPalm();
        // equation 6
        vertex1.x = vertex1.u / s;
        vertex1.y = vertex1.v / s;
        vertex1.z = vertex1.w / s;

        ////////////////////////////////////////////////////////
        //
        // FORMAT:
        // j0 32 -4.92007 -1.23411 1.3899
        //
        outf << "j" << vertex1.id << " " // node name e.g. "j0"
            << this->getId() << " "      // frameId
            << vertex1.x << " "
            << vertex1.y << " "
           << vertex1.z
            << endl;
        ////////////////////////////////////////////////////////
    }

    double angle(const Vertex& vertexA, const Vertex& vertexB, const
Vertex& vertexC)
```

```
    {
     // law of cosines
     //
     // b2 = a2 + c2 - 2ac cos x
     //
     //             C
     //            /\
     //           /   \
     //     b    /       \   a
     //         /          \
     //        /             \
     //       /          (x    \
     //     A -----------------B
     //                c
     //
     // 2ac cos x = a2 + c2 - b2
     //      x = arccos ((a2 + c2 - b2)/2ac)
     //
     double a =  sqrt(pow2(vertexC.z - vertexB.z) + pow2(vertexC.y -
vertexB.y));
          double b =  sqrt(pow2(vertexC.z - vertexA.z) + pow2(vertexC.y
- vertexA.y));
          double c =  sqrt(pow2(vertexA.z - vertexB.z) + pow2(vertexA.y
- vertexB.y));

          double x = acos((pow2(a) + pow2(c) - pow2(b))/(2*a*c));
          const double PI = 3.14159265;

          double result = x * 180.0 / PI;

          return result;
    }
};

//////////////////////////////////////////////////
class DataTracker
{
private:
     vector<Frame> m_frames;
     int m_cur_frame;
     int m_total_frames;

public:
     DataTracker():m_cur_frame(0),m_total_frames(0) {}

     int init() //Currently read data from an input file
     //The input file provides a list of 2D feature points of ALL frames
     {
          cerr << "entering DataTracker::init\n";

          //open file
          ifstream is;
          is.open("c:\\data.txt");

          if (!is.is_open())
          {
               cerr << "cannot open input file\n";
               return -1;
          }

          Frame f[100];
```

```
        m_total_frames = 100;
        cerr << "total frames is " << m_total_frames << endl;

        // set frame id :(
        for (int i=0; i< m_total_frames; ++i)
        {
            //print frame for debugging
            f[i].setId(i);
        }

        while(!is.eof())
        {
            //double u[HandModel::NUM_FEATURE_POINTS],
v[HandModel::NUM_FEATURE_POINTS];

            string jointName;
            int frameNumber;
            double u;
            double v;
            double w;

            is >> jointName >> frameNumber >> u >> v >> w;

            int j;
            char c;
            // parse for j from jointName e.g. "x12" => 12
            sscanf(jointName.c_str(), "%c%d", &c, &j);

            cout << jointName << " => " << c << ", " << j << endl;

            //set it
            f[frameNumber].setfp(j, Vertex (j, u, v));
            f[frameNumber].getFpRef(j).setUVSetFlag(true);
        }

        for (int i=0; i< 100; ++i)
        {
            m_frames.push_back(f[i]);
        }

        for (int i=0; i< m_total_frames; ++i)
        {
            //print frame for debugging
            m_frames[i].print();
        }

        return 0;
    }

    Frame& getCurrentFrame() //2D feature point from data tracking
    {
        if (m_cur_frame >= m_total_frames)
        {
            cerr << "ERROR: entering DataTracker::getCurrentFrame\n";
            return m_frames[0];
        }
        return m_frames[m_cur_frame++];
    }

    int getTotalFrames()
    {
```

```
        return m_total_frames;
    }

    vector<Frame>& getFrames()
    {
        return m_frames;
    }

};

////////////////////////////////////////////////
class TwoDResolver
{
private:
    static TwoDResolver* m_instance;

public:
    static TwoDResolver* instance()
    {
        cerr << "entering TwoDResolver::instance\n";
        if ( m_instance == 0)
        {
            m_instance = new TwoDResolver();
        }
        return m_instance;
    }

    enum FillInMissingDataMethod {
            FILL_IN_MISSING_DATA_NOTHING = 1,
            FILL_IN_MISSING_DATA_PREVIOUS_FRAME_DATA = 2,
            FILL_IN_MISSING_DATA_PREVIOUS_DIFF_FRAME_DATA = 3,
            FILL_IN_MISSING_DATA_PREVIOUS_DIFF_INTERDEP_FRAME_DATA = 4,

FILL_IN_MISSING_DATA_PREVIOUS_DIFF_INTERDEP_INTRAFINGER_FRAME_DATA = 5,

FILL_IN_MISSING_DATA_PREVIOUS_DIFF_INTERDEP_INTRAFINGER_VALUE_FRAME_DATA
= 6
         };

    /**
     * Interdependence
     *
     * Use different techniques
     */
    void fillInMissingData(Frame* f, DataTracker& dt, enum
FillInMissingDataMethod method)
    {
        //which feature points are missing
        for(int i=0; i< HandModel::NUM_FEATURE_POINTS; ++i)
        {
            if(f->getFpRef(i).isUVSet())
                continue;

            // CLUTCH
            // if it gets here it means this feature point uv is
missing
            // which means there feature point ismissing so
DataTracker didn't read it from the input file
            // So we have to add it
            // since the call f->getFpRef(i) above automatically add
it (with default value)
```

```
            // we need to correct its id.
            f->getFpRef(i).id = i;

            //missing data, synthesize one
            switch (method)
            {
                case FILL_IN_MISSING_DATA_NOTHING:
                    break;

                case FILL_IN_MISSING_DATA_PREVIOUS_FRAME_DATA:
                    cerr << "previous : " << f->getFpRef(i).id <<
endl;
                    fillInVertexUsePreviousFrame(f->getFpRef(i), f-
>getId(), dt);
                    break;

                case FILL_IN_MISSING_DATA_PREVIOUS_DIFF_FRAME_DATA:
                    cerr << "previous diff : " << f->getFpRef(i).id
<< endl;
                    fillInVertexUsePreviousDiffFrame(f->getFpRef(i),
f->getId(), dt);
                    break;

                case
FILL_IN_MISSING_DATA_PREVIOUS_DIFF_INTERDEP_FRAME_DATA:
                    cerr << "previous diff interdep : " << f-
>getFpRef(i).id << endl;
                    fillInVertexUsePreviousDiffInterdepFrame(f-
>getFpRef(i), f->getId(), dt);
                    break;


                case
FILL_IN_MISSING_DATA_PREVIOUS_DIFF_INTERDEP_INTRAFINGER_FRAME_DATA:
                    cerr << "previous diff interdep intrafinger : "
<< f->getFpRef(i).id << endl;

    fillInVertexUsePreviousDiffInterdepIntraFingerFrame(f->getFpRef(i),
f->getId(), dt);
                    break;


                case
FILL_IN_MISSING_DATA_PREVIOUS_DIFF_INTERDEP_INTRAFINGER_VALUE_FRAME_DATA
:
                    cerr << "previous diff interdep intrafinger
value: " << f->getFpRef(i).id << endl;

    fillInVertexUsePreviousDiffInterdepIntraFingerValueFrame(f-
>getFpRef(i), f->getId(), dt);
                    break;
            };
        }
    }

    void fillInVertexUsePreviousFrame(Vertex& v, int frameId,
DataTracker& dt)
    {
        // get the previous frame
        if (frameId == 0)
        {
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

```
            //first frame missing :(
            throw 9999; //give up
        }
        Frame& f = dt.getFrames()[frameId-1];
        double prevFrameU = f.getFpRef(v.id).u;
        double prevFrameV = f.getFpRef(v.id).v;

        v.setUV(prevFrameU, prevFrameV);
    }

    void fillInVertexUsePreviousDiffFrame(Vertex& v, int frameId,
DataTracker& dt)
    {
        // get the previous frame
        if (frameId == 0 || frameId == 1)
        {
            //first frame missing :(
            throw 9999; //give up
        }

        //previous frame
        Frame& pf = dt.getFrames()[frameId-1];
        double prevFrameU = pf.getFpRef(v.id).u;
        double prevFrameV = pf.getFpRef(v.id).v;

        //previous's previous frame
        Frame& ppf = dt.getFrames()[frameId-2];
        double prevprevFrameU = ppf.getFpRef(v.id).u;
        double prevprevFrameV = ppf.getFpRef(v.id).v;


        double currentU = prevFrameU + (prevFrameU - prevprevFrameU);
        double currentV = prevFrameV + (prevFrameV - prevprevFrameV);
        v.setUV(currentU, currentV);
    }

    void fillInVertexUsePreviousDiffInterdepFrame(Vertex& v, int
frameId, DataTracker& dt)
    {
        // get the previous frame
        if (frameId == 0 || frameId == 1 || frameId == 2) // bec we
need at least three to determine if direction reverses
        {
            throw 9999; //give up
        }

        //previous frame
        Frame& pf = dt.getFrames()[frameId-1];
        double prevFrameU = pf.getFpRef(v.id).u;
        double prevFrameV = pf.getFpRef(v.id).v;

        //previous's previous frame
        Frame& ppf = dt.getFrames()[frameId-2];
        double prevprevFrameU = ppf.getFpRef(v.id).u;
        double prevprevFrameV = ppf.getFpRef(v.id).v;

        //check if neighbor's direction is reversed now
        //if so , we should move in the reverse direction
        Frame& pppf = dt.getFrames()[frameId-3];
        double neighborPrevPrevPrevFrameU =
pppf.getFpRef(HandModel::findNeighborId(v.id)).u;
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

```
          double neighborPrevPrevPrevFrameV =
pppf.getFpRef(HandModel::findNeighborId(v.id)).v;
          double neighborPrevPrevFrameU =
ppf.getFpRef(HandModel::findNeighborId(v.id)).u;
          double neighborPrevPrevFrameV =
ppf.getFpRef(HandModel::findNeighborId(v.id)).v;
          double neighborPrevFrameU =
pf.getFpRef(HandModel::findNeighborId(v.id)).u;
          double neighborPrevFrameV =
pf.getFpRef(HandModel::findNeighborId(v.id)).v;

          double currentU;
          double currentV;

              //U
              // 4 > 3 < 5 or 3 < 5 > 4 == reverse
               // if trend is bucking down and we're going up, reverse it
               if (
                    (neighborPrevPrevPrevFrameU < neighborPrevPrevFrameU
&& neighborPrevPrevFrameU > neighborPrevFrameU) &&
                    (prevprevFrameU < prevFrameU)
                  )
               {
                    cerr << "reverseU\n";
                    //reverse U direction
                    currentU = prevFrameU - (prevFrameU -
prevprevFrameU);
               }

               // if trend is bucking up and we're going down, reverse it
               else if (
                    (neighborPrevPrevPrevFrameU > neighborPrevPrevFrameU
&& neighborPrevPrevFrameU < neighborPrevFrameU) &&
                    (prevprevFrameU > prevFrameU)
                  )
               {
                    cerr << "reverseU\n";
                    //reverse U direction
                    currentU = prevFrameU - (prevFrameU -
prevprevFrameU);
               }
               // otherwise don't reverse it
               else
               {
                    cerr << "not reverseU\n";
                    currentU = prevFrameU + (prevFrameU -
prevprevFrameU);
               }


               //V
               if ( (neighborPrevPrevPrevFrameV > neighborPrevPrevFrameV
&& neighborPrevPrevFrameV < neighborPrevFrameV) &&
                    (prevprevFrameV < prevFrameV)
                    )
               {
                    cerr << "reverseV\n";
                    //reverse U direction
                    currentV = prevFrameV - (prevFrameV -
prevprevFrameV);
               }
```

```
              else if (
                   (neighborPrevPrevPrevFrameV < neighborPrevPrevFrameV
&& neighborPrevPrevFrameV > neighborPrevFrameV) &&
                   (prevprevFrameV > prevFrameV)
                   )
              {
                   cerr << "reverseV\n";
                   //reverse U direction
                   currentV = prevFrameV - (prevFrameV -
prevprevFrameV);
              }
              else
              {
                   cerr << "not reverseV\n";
                   currentV = prevFrameV + (prevFrameV -
prevprevFrameV);
              }

         v.setUV(currentU, currentV);
     }

     void fillInVertexUsePreviousDiffInterdepIntraFingerFrame(Vertex& v,
int frameId, DataTracker& dt)
     {
         // get the previous frame
         if (frameId == 0 || frameId == 1 || frameId == 2) // bec we
need at least three to determine if direction reverses
         {
              throw 9999; //give up
         }

         //previous frame
         Frame& pf = dt.getFrames()[frameId-1];
         double prevFrameU = pf.getFpRef(v.id).u;
         double prevFrameV = pf.getFpRef(v.id).v;

         //previous's previous frame
         Frame& ppf = dt.getFrames()[frameId-2];
         double prevprevFrameU = ppf.getFpRef(v.id).u;
         double prevprevFrameV = ppf.getFpRef(v.id).v;

         //check if neighbor's direction is reversed now
         //if so , we should move in the reverse direction
         Frame& pppf = dt.getFrames()[frameId-3];
         double neighborPrevPrevPrevFrameU =
pppf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).u;
         double neighborPrevPrevPrevFrameV =
pppf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).v;
         double neighborPrevPrevFrameU =
ppf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).u;
         double neighborPrevPrevFrameV =
ppf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).v;
         double neighborPrevFrameU =
pf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).u;
         double neighborPrevFrameV =
pf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).v;

         double currentU;
         double currentV;

         //U
```

```
        if (
             (neighborPrevPrevPrevFrameU > neighborPrevPrevFrameU &&
neighborPrevPrevFrameU < neighborPrevFrameU) &&
             (prevprevFrameU > prevFrameU)
             )
        {
             cerr << "reverseU\n";
             //reverse U direction
             currentU = prevFrameU - (prevFrameU - prevprevFrameU);

        }
        else if (
              (neighborPrevPrevPrevFrameU < neighborPrevPrevFrameU &&
neighborPrevPrevFrameU > neighborPrevFrameU) &&
              (prevprevFrameU < prevFrameU)
             )
        {
             cerr << "reverseU\n";
             //reverse U direction
             currentU = prevFrameU - (prevFrameU - prevprevFrameU);
        }
        else
        {
             cerr << "not reverseU\n";
             currentU = prevFrameU + (prevFrameU - prevprevFrameU);
        }

        //V
        if (
             (neighborPrevPrevPrevFrameV > neighborPrevPrevFrameV &&
neighborPrevPrevFrameV < neighborPrevFrameV) &&
              (prevprevFrameV > prevFrameV)
        )
        {
             cerr << "reverseV\n";
             //reverse U direction
             currentV = prevFrameV - (prevFrameV - prevprevFrameV);
        }
        else if (
             (neighborPrevPrevPrevFrameV < neighborPrevPrevFrameV &&
neighborPrevPrevFrameV > neighborPrevFrameV) &&
             (prevprevFrameV < prevFrameV)
             )
        {
             cerr << "reverseV\n";
             //reverse U direction
             currentV = prevFrameV - (prevFrameV - prevprevFrameV);
        }
        else
        {
             cerr << "not reverseV\n";
             currentV = prevFrameV + (prevFrameV - prevprevFrameV);
        }

        v.setUV(currentU, currentV);
    }

    void
fillInVertexUsePreviousDiffInterdepIntraFingerValueFrame(Vertex& v, int
frameId, DataTracker& dt)
    {
```

```
        // get the previous frame
        if (frameId == 0 || frameId == 1 || frameId == 2) // bec we
need at least three to determine if direction reverses
        {
            throw 9999; //give up
        }

        //previous frame
        Frame& pf = dt.getFrames()[frameId-1];
        double prevFrameU = pf.getFpRef(v.id).u;
        double prevFrameV = pf.getFpRef(v.id).v;

        //previous's previous frame
        Frame& ppf = dt.getFrames()[frameId-2];
        double prevprevFrameU = ppf.getFpRef(v.id).u;
        double prevprevFrameV = ppf.getFpRef(v.id).v;

        //check if neighbor's direction is reversed now
        //if so , we should move in the reverse direction
        Frame& pppf = dt.getFrames()[frameId-3];
        double neighborPrevPrevPrevFrameU =
pppf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).u;
        double neighborPrevPrevPrevFrameV =
pppf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).v;
        double neighborPrevPrevFrameU =
ppf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).u;
        double neighborPrevPrevFrameV =
ppf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).v;
        double neighborPrevFrameU =
pf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).u;
        double neighborPrevFrameV =
pf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).v;

        //prevFrameU and prevprevFrameU of intra neighbor
        double neighbour_prevFrameU =
pf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).u;
        double neighbour_prevFrameV =
pf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).v;

        //Frame& ppf = dt.getFrames()[frameId-2];
        double neighbour_prevprevFrameU =
ppf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).u;
        double neighbour_prevprevFrameV =
ppf.getFpRef(HandModel::findIntraFingerNeighborId(v.id)).v;
        double currentU;
        double currentV;

        ////U
        if ( (neighborPrevPrevPrevFrameU > neighborPrevPrevFrameU &&
neighborPrevPrevFrameU < neighborPrevFrameU) &&
             (prevprevFrameU > prevFrameU)
            )
        {
            //reverse U direction
            //apply rate of change instead
            double chnageby = abs(neighbour_prevFrameU -
neighbour_prevprevFrameU);
            double percenttochange = (chnageby * 100.0) /
neighbour_prevprevFrameU;
            double amounttochange = (prevFrameU *
percenttochange)/100.0;
```

```
                // apply the amount with the correct sign
                if(neighbour_prevFrameU - neighbour_prevprevFrameU < 0)
                     currentU = prevFrameU - (-1.0* abs(amounttochange));
                else
                     currentU = prevFrameU - (abs(amounttochange));


          }
          else if (
                (neighborPrevPrevPrevFrameU < neighborPrevPrevFrameU &&
neighborPrevPrevFrameU > neighborPrevFrameU) &&
                (prevprevFrameU < prevFrameU)
                )
          {
                //reverse U direction
                //apply rate of change instead
                double chnageby = abs(neighbour_prevFrameU -
neighbour_prevprevFrameU);
                double percenttochange = (chnageby * 100.0) /
neighbour_prevprevFrameU;
                double amounttochange = (prevFrameU *
percenttochange)/100.0;

                // apply the amount with the correct sign
                if(neighbour_prevFrameU - neighbour_prevprevFrameU < 0)
                     currentU = prevFrameU - (-1.0* abs(amounttochange));
                else
                     currentU = prevFrameU - (abs(amounttochange));
          }
          else
          {
                cerr << "not reverseU\n";
                currentU = prevFrameU + (neighbour_prevFrameU -
neighbour_prevprevFrameU);
          }

          ////V
          if ( (neighborPrevPrevPrevFrameV > neighborPrevPrevFrameV &&
neighborPrevPrevFrameV < neighborPrevFrameV) &&
                 (prevprevFrameV > prevFrameV)
          )
          {
                //apply rate of change instead
                double chnageby = abs(neighbour_prevFrameV -
neighbour_prevprevFrameV);
                double percenttochange = (chnageby * 100.0) /
neighbour_prevprevFrameV;
                double amounttochange = (prevFrameV *
percenttochange)/100.0;

                // apply the amount with the correct sign
                if(neighbour_prevFrameV - neighbour_prevprevFrameV < 0)
                     currentV = prevFrameV - (-1.0* abs(amounttochange));
                else
                     currentV = prevFrameV - (abs(amounttochange));
          }
          else if (
                (neighborPrevPrevPrevFrameV < neighborPrevPrevFrameV &&
neighborPrevPrevFrameV > neighborPrevFrameV) &&
                (prevprevFrameV < prevFrameV)
```

```
            )
        {
            //apply rate of change instead
            double chnageby = abs(neighbour_prevFrameV -
neighbour_prevprevFrameV);
            double percenttochange = (chnageby * 100.0) /
neighbour_prevprevFrameV;
            double amounttochange = (prevFrameV *
percenttochange)/100.0;

            // apply the amount with the correct sign
            if(neighbour_prevFrameV - neighbour_prevprevFrameV < 0)
                currentV = prevFrameV - (-1.0* abs(amounttochange));
            else
                currentV = prevFrameV - (abs(amounttochange));
        }
        else
        {
            cerr << "not reverseV\n";
            currentV = prevFrameV + (neighbour_prevFrameV -
neighbour_prevprevFrameV);
        }
        v.setUV(currentU, currentV);
    }

    void addFrame(Frame& f)
    {
        cerr << "entering TwoDResolver::addFrame\n";
    }
};

TwoDResolver* TwoDResolver::m_instance = 0;

////////////////////////////////////////////////
class Renderer {
private:
    static Renderer* m_instance;

public:
    static Renderer* instance()
    {
        cerr << "entering Renderer::instance\n";
        if ( m_instance == 0)
        {
            m_instance = new Renderer();
        }
        return m_instance;
    }

    void addFrame(Frame& f)
    {
        cerr << "entering Renderer::addFrame\n";
    }
};

Renderer* Renderer::m_instance = 0;
////////////////////////////////////////////////

int main()
{
    try {
```

```
        HandModel::init();

        DataTracker dt;
        if (dt.init() != 0) // may pass in some input file
        {
             return 0;
        }

        // open an output file
        // This will keep our computed z order
        // plugin myTranslateTo will read this file into maya
        //
        ofstream outf;
        outf.open("c:\\computedData.txt");

        if (!outf.is_open())
        {
             cerr << "cannot open output file\n";
             throw 7777;
        }


        TwoDResolver* twoDResolver = TwoDResolver::instance();
        Renderer* renderer = Renderer::instance();

        for(int i=0; i< dt.getTotalFrames(); ++i)
        {
             cerr << "****************** START FRAME " << i << "
***************************\n";
             // get current frame
             Frame& f = dt.getCurrentFrame(); //2D feature point from
data tracking

             f.print();

             // Missing data synthesis techniques: pick one
             //output: all 2D feature points
             //twoDResolver->fillInMissingData(&f, dt,
FILL_IN_MISSING_DATA_NOTHING);
             //twoDResolver->fillInMissingData(&f, dt,
TwoDResolver::FILL_IN_MISSING_DATA_PREVIOUS_FRAME_DATA);
             //twoDResolver->fillInMissingData(&f, dt,
TwoDResolver::FILL_IN_MISSING_DATA_PREVIOUS_DIFF_FRAME_DATA);
             //twoDResolver->fillInMissingData(&f, dt,
TwoDResolver::FILL_IN_MISSING_DATA_PREVIOUS_DIFF_INTERDEP_FRAME_DATA);
             //twoDResolver->fillInMissingData(&f, dt,
TwoDResolver::FILL_IN_MISSING_DATA_PREVIOUS_DIFF_INTERDEP_INTRAFINGER_FR
AME_DATA);
             twoDResolver->fillInMissingData(&f, dt,
TwoDResolver::FILL_IN_MISSING_DATA_PREVIOUS_DIFF_INTERDEP_INTRAFINGER_VA
LUE_FRAME_DATA);

             //once we get the scaled length of each segment
             //compare this with the observed length of each segment
             //we know if the segment is tilting (has depth)
             //and we can compute the depth (Z dimension) from the
             //scale factor we have, the obserb
             f.computeZCoordinates(outf);            //output: Z
coordinates
```

```
            cerr << "***** END FRAME " << i << "******\n";
        }

        outf.close();

    }
    catch(MyException &e)
    {
        e.print();
    }
    return 0;
}
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Appendix C

Coordination Export Program

This program is to export X, Y, Z coordinates of feature points of each animation frame to a text file. It is written as a Maya Plugin using Maya C++ API. To load the plugin to Maya, first make sure the .mll library is in a plugin path recognized by Maya. Then, open the script editor in Maya and type in the following command:

loadPlugin myCmd;

.

```
#include <math.h>
#include <maya/MIOStream.h>
#include <maya/MSimple.h>
#include <maya/MPoint.h>
#include <maya/MPointArray.h>
#include <maya/MDoubleArray.h>
#include <maya/MFnNurbsCurve.h>

#include <maya/MSimple.h>
#include <maya/MGlobal.h>
#include <maya/MString.h>
#include <maya/MDagPath.h>
#include <maya/MFnDagNode.h>
#include <maya/MFnTransform.h>
#include <maya/MVector.h>
#include <maya/MSelectionList.h>
#include <maya/MIOStream.h>

#include <fstream>

DeclareSimpleCommand( doHelix, "Autodesk - Example", "8.0");

MStatus doMe( const MArgList& )
{
    MDagPath            node;
    MObject             component;
    MSelectionList      list;
    MFnDagNode          nodeFn;
     MFnTransform        transformFn;
    MGlobal::getActiveSelectionList( list );

     // open output file
     //
     std::ofstream outf;
     outf.open("c:\\data.txt");

     // loop through all selected nodes
     //
    for ( unsigned int index = 0; index < list.length(); index++ )
    {
        list.getDagPath( index, node, component );
        nodeFn.setObject( node );
```

```
          transformFn.setObject( node );

          for (int i =0; i < 100; ++i)
          {
               MGlobal::viewFrame(i);
               MVector transformVector = transformFn.getTranslation(
MSpace::Space::kWorld );

               outf << nodeFn.name().asChar() << " "
                     << i  << " "
                     << transformVector.x << " "
                    << transformVector.y << " "
                     << transformVector.z << endl;
          }
    }

     // close output file
     //
     outf.close();

    return MS::kSuccess;
}


MStatus doHelix::doIt( const MArgList& args)
{
    MStatus stat;

     doMe(args);
    return stat;
}
```

Appendix D
Coordination Import Program

This program is to import X, Y, and our computed Z coordinates of feature points of each animation frame to Maya. It is written as a Maya Plugin using Maya C++ API. To load the plugin to Maya, first make sure the .mll library is in a plugin path recognized by Maya. Then, open the script editor in Maya and type in the following command:

loadPlugin myTranslateTo;

```
#include <math.h>
#include <maya/MIOStream.h>
#include <maya/MSimple.h>
#include <maya/MPoint.h>
#include <maya/MPointArray.h>
#include <maya/MDoubleArray.h>
#include <maya/MFnNurbsCurve.h>

#include <maya/MSimple.h>
#include <maya/MGlobal.h>
#include <maya/MString.h>
#include <maya/MDagPath.h>
#include <maya/MFnDagNode.h>
#include <maya/MFnTransform.h>
#include <maya/MVector.h>
#include <maya/MSelectionList.h>
#include <maya/MIOStream.h>

#include <fstream>
#include <map>
#include <string>

// Use helper macro to register a command with Maya.  It creates and
// registers a command that does not support undo or redo.  The
// created class derives off of MPxCommand.
//
DeclareSimpleCommand( mtt, "", "8.0");

using std::string;
using std::map;

map<string, map<int, MVector> > g_inputData;

MStatus readComputedData()
{
     // open input file
     // this is the file that contains our computed z order
     //
     std::ifstream inf;
     inf.open("c:\\computedData.txt");

     if (!inf.is_open())
     {
          cerr << "cannot open input file\n";
```

```
        return MS::kFailure;
    }

    // read all data for all frames
    //
    // FORMAT:
    // j0 32 -4.92007 -1.23411 1.3899
    string nodeName;
    int frameId;
    double u;
    double v;
    double w;

    while (!inf.eof())
    {
        inf >> nodeName >> frameId >> u >> v >> w;

        //add it to heap
        g_inputData[nodeName][frameId] = MVector(u, v, w);
        cerr << "INPUT readComputedData(): from input file: " <<
nodeName << " , " << frameId << ", "
            << g_inputData[nodeName][frameId].x
            << g_inputData[nodeName][frameId].y
            << g_inputData[nodeName][frameId].z
            << endl;
    }

    inf.close();
    return MS::kSuccess;
}


MStatus doMe( const MArgList& )
{
    MDagPath            node;
    MObject             component;
    MSelectionList      list;
    MFnDagNode          nodeFn;
     MFnTransform        transformFn;
    MGlobal::getActiveSelectionList( list );


    // read input file
    // keep it in heap
    if (readComputedData() != MS::kSuccess)
    {
        return MS::kFailure;
    }

    // loop through all selected nodes
    //
    for ( unsigned int index = 0; index < list.length(); index++ )
    {
        list.getDagPath( index, node, component );
        nodeFn.setObject( node );

        transformFn.setObject( node );

        // find the last frame
        //
        unsigned int max_frame = 0;
```

```
        for(map<string, map<int, MVector> >::iterator mit =
g_inputData.begin(); mit != g_inputData.end(); ++mit)
        {
              //map<int, MVector>& rmap = g_inputData[i];
             cout << "mit->second.size() > max_frame " << mit-
>second.size() << " " << max_frame << endl;
             if (mit->second.size() > max_frame)
             {
                  cout << "set mit->second.size() " << mit-
>second.size() << endl;
                  max_frame = mit->second.size();
             }
        }

        cout << "max_frame " << max_frame << endl;
        for (int i =0; i < max_frame; ++i)
        {
             MGlobal::viewFrame(i);
             //MVector transformVector = transformFn.getTranslation(
MSpace::Space::kWorld );

             // Set translate fo this frame for this feature point
             //tatus MPxTransform:: translateTo (const MVector &
newTrans, MSpace::Space space , const MDGContext &context )

             // Set to what we read from our computed data file
             //
             if (MS::kSuccess !=
transformFn.setTranslation(g_inputData[nodeFn.name().asChar()][i],
MSpace::Space::kWorld))
             {

                  cerr << "ERROR!!!!!!!!!!!!!!: SET TRANSLATE TO: "
                   << nodeFn.name().asChar() << " "
                   << i  << " "
                   << g_inputData[nodeFn.name().asChar()][i]
                   << g_inputData[nodeFn.name().asChar()][i].x << " "
                  << g_inputData[nodeFn.name().asChar()][i].y << " "
                   << g_inputData[nodeFn.name().asChar()][i].z << endl;

             }
             else
             {
                  cerr << "SUCCESS: SET TRANSLATE TO: "
                       << nodeFn.name().asChar() << " "
                       << i  << " "
                       << g_inputData[nodeFn.name().asChar()][i]
                       << g_inputData[nodeFn.name().asChar()][i].x << "
"
                       << g_inputData[nodeFn.name().asChar()][i].y << "
"
                       << g_inputData[nodeFn.name().asChar()][i].z <<
endl;
             }
        }

    }

    return MS::kSuccess;
}
```

```
MStatus mtt::doIt( const MArgList& args )
{
    return doMe(args);
}
```

Appendix E

Coordination Data Sort Program

```
open($fh, "<c:\\computedData.txt") || die ("cannot open input file");
$i = 0;

while ($line=<$fh>)
{
$lines[$i] = $line;
$i++;
}

print sort numerically @lines;

sub numerically {
    @as = split (/ /, $a);
    @bs = split (/ /, $b);

$as[0] cmp $bs[0]
 ||
$as[1] <=> $bs[1]
}
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Appendix F

Original and Computed Data Diff Program

```
#include <iostream>
#include <string>
#include <fstream>
#include <map>

using namespace std;

class ArgumentParser
{
private:
     ArgumentParser();

public:
     static void parse(int argc, char** argv);
     static const string& getFile1() { return file1; }
     static const string& getFile2() { return file2; }
     static bool getsortByFrame() { return sortByFrame; }
     static bool getsortByJoint() { return sortByJoint; }

private:
     static bool sortByFrame;
     static bool sortByJoint;
     static string file1;
     static string file2;
};

bool ArgumentParser::sortByFrame = false;
bool ArgumentParser::sortByJoint = false;
string ArgumentParser::file1;
string ArgumentParser::file2;

void ArgumentParser::parse(int argc, char** argv)
{
     for (int i=0; i< argc; ++i)
     {
          if(string(argv[i]) == "-s")
          {
               sortByFrame = false;
               sortByJoint = false;

               string nextArg(argv[++i]);
               if (nextArg == "f") //sort by frame
                    sortByFrame = true;
               else if (nextArg == "j") //sort by joint
                    sortByJoint = true;
          }
          else if(string(argv[i]) == "-f1")
          {
               file1 = string(argv[++i]);
          }
          else if(string(argv[i]) == "-f2")
          {
               file2 = string(argv[++i]);
          }
     }
}
```

```
/*************************************************
   * j0 0 -4.6975 -0.206784 5.0577
   * name frame x y z
   *
   * - we open two input files
   * - for file1, file2
   * - read line by line and put in
   * map1[jointNumber][frameNumber] = {x,y,z}
   * map2[jointNumber][frameNumber] = {x,y,z}
   *
   * compare choices
   * - compare z value
   *
   * sort choice
   * - by joint
   * - by frame
   *
   * name -> "j0" we'll extract to 0
   *
 */
class Point3D
{
public:
     Point3D(double xx, double yy, double zz): x(xx), y(yy), z(zz) {}
     Point3D(): x(0), y(0), z(0) {}

     double x;
     double y;
     double z;
};

/**************
 * Per file
 * keep in map
 */
class DataSet
{
public:
     // read data into map
     DataSet(const string& inputFileName);
     std::map <int, std::map<int, Point3D> >& getAllData() {  cerr <<
"map size is " << m_map[2].size() << endl ; return m_map; }

private:
     map<int, map<int, Point3D> > m_map;
};

DataSet::DataSet(const string& inputFileName)
{
     std::ifstream inf;
     inf.open(inputFileName.c_str());

     int i = 0;
     while(!inf.eof())
     {
          char j;
          string jointName;
          int jointNumber;
          int frameNumber;
          double x;
```

```
        double y;
        double z;

        //j0 5 -4.76268 -0.427315 4.82576
        inf >> jointName >> frameNumber >> x >> y >> z;
        sscanf(jointName.c_str(), "%c%d", &j, &jointNumber);

        m_map[jointNumber][frameNumber] = Point3D(x, y, z);

        cerr << "input line = " << ++i << endl;
    }
}

/*********************************    MAIN PROCEDURE
*************************************/
void compareZ(map<int, map<int, Point3D> >& map1, map<int, map<int,
Point3D> >& map2)
{
    // assume 2 maps have the same number of entries
    //
    //j0 5 -4.76268 -0.427315 4.82576
    //map [joint] [ frame]

    map < int, map <int, double> > results;

    for(int i= 0; i < map1.size(); ++i) // i is joint; j is frame
    for(int j= 0; j < map1[i].size(); ++j)
    {
        results[i][j] = map1[i][j].z - map2[i][j].z;
    }

    if (ArgumentParser::getsortByJoint())
    {
        // sort by joint
        for(int i= 0; i < map1.size(); ++i) // i is joint; j is frame
        for(int j= 0; j < map1[i].size(); ++j)
        {
            cout << "j" << i << " "
                 << j << " "
                 << map1[i][j].x << " "
                 << map1[i][j].y << " "
                 << results[i][j] << endl;
        }
    }
    else if (ArgumentParser::getsortByFrame())
    {
        // sort by frame
        for(int j= 0; j < map1[0].size(); ++j)
        for(int i= 0; i < map1.size(); ++i) // i is joint; j is frame
        {
            cout << "j" << i << " "
                 << j << " "
                 << map1[i][j].x << " "
                 << map1[i][j].y << " "
                 << results[i][j] << endl;
        }
    }
}

/*********
 * 2 argument
```

```
 * a.out -s[j,f] -f1 <inputfile1> -f2 <inputfile2>
 */
int main(int argc, char** argv)
{
     try
     {
          ArgumentParser::parse(argc, argv); //assume argument is
correct
          DataSet dataSet1(ArgumentParser::getFile1());
          DataSet dataSet2(ArgumentParser::getFile2());

          // now we got all data in two maps
          // let's compare
          //
          map<int, map<int, Point3D> >& map1 = dataSet1.getAllData();
          map<int, map<int, Point3D> >& map2 = dataSet2.getAllData();

          //compare Z
          compareZ(map1, map2);
     }
     catch (...)
     {
     }
}
```

Biography

Mr. Kosit Nopvichai received his Bachelor Degree in Computer Science from Thammasat University. He is persuing a Master Degree in Computer Science at Chulalongkorn University. Currently, he is working at a financial software company as a Senior Software Engineer.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย