2014

Automatic 3D human modeling: an initial stage towards 2-way inside interaction in mixed reality

Yiyan Xiong
University of Central Florida

AUTOMATIC 3D HUMAN MODELING: AN INITIAL STAGE TOWARDS 2-WAY INSIDE INTERACTION IN MIXED REALITY

by

YIYAN XIONG
M.S. University of Central Florida, 2011

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2014

Major Professor: Charles E. Hughes

ABSTRACT

3D human models play an important role in computer graphics applications from a wide range of domains, including education, entertainment, medical care simulation and military training. In many situations, we want the 3D model to have a visual appearance that matches that of a specific living person and to be able to be controlled by that person in a natural manner. Among other uses, this approach supports the notion of human surrogacy, where the virtual counterpart provides a remote presence for the human who controls the virtual character's behavior.

In this dissertation, a human modeling pipeline is proposed for the problem of creating a 3D digital model of a real person. Our solution involves reshaping a 3D human template with a 2D contour of the participant and then mapping the captured texture of that person to the generated mesh. Our method produces an initial contour of a participant by extracting the user image from a natural background. One particularly novel contribution in our approach is the manner in which we improve the initial vertex estimate. We do so through a variant of the ShortStraw corner-finding algorithm commonly used in sketch-based systems. Here, we develop improvements to ShortStraw, presenting an algorithm called IStraw, and then introduce adaptations of this improved version to create a corner-based contour segmentatiuon algorithm. This algorithm provides significant improvements on contour matching over previously developed systems, and does so with low computational complexity.

The system presented here advances the state of the art in the following aspects. First, the human modeling process is triggered automatically by matching the participant's pose with an initial pose through a tracking device and software. In our case, the pose capture and skeletal model are provided by the Microsoft Kinect and its associated SDK. Second, color image, depth data, and human tracking information from the Kinect and its SDK are used to automatically extract the

contour of the participant and then generate a 3D human model with skeleton. Third, using the pose and the skeletal model, we segment the contour into eight parts and then match the contour points on each segment to a corresponding anchor set associated with a 3D human template. Finally, we map the color image of the person to the 3D model as its corresponding texture map.

The whole modeling process only take several seconds and the resulting human model looks like the real person. The geometry of the 3D model matches the contour of the real person, and the model has a photorealistic texture. Furthermore, the mesh of the human model is attached to the skeleton provided in the template, so the model can support programmed animations or be controlled by real people. This human control is commonly done through a literal mapping (motion capture) or a gesture-based puppetry system.

Our ultimate goal is to create a mixed reality (MR) system, in which the participants can manipulate virtual objects, and in which these virtual objects can affect the participant, e.g., by restricting their mobility. This MR system prototype design motivated the work of this dissertation, since a realistic 3D human model of the participant is an essential part of implementing this vision.

I dedicate this to my mother.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Charles E. Hughes, for his support to my research and life. I would also like to thank the rest of my committee members, Joseph J. LaViola Jr., J. Michael Moshell, and Sumanta Pattanaik, for their suggestions to my work. Many colleagues from the Institute for Simulation and Training at UCF, especially the Synthetic Reality Lab, helped during this research with technical discussions or providing test data. I also want to thank the IARPA, SAIC, NSF CAREER award IIS-0845921, and NSF research awards: Water's Journey through the Everglades, DRL-0638977, and Interconnections: Revisiting the Future, DRL-0840297 for their funding support of this work.

In the end, I would like to express my appreciation for my family's support over the years, especially my mother and my husband. Without their understanding and support, it would have been impossible to attain my current accomplishments. As a single mom, my mother told me and showed me the importance of strong willpower. My husband always stands by me and makes me want to be a better person.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Background

3D modeling from real world data has received a great deal of attention, resulting in many existing systems to generate 3D models of small static objects or to reconstruct scenes. However, most of the 3D human models in current applications are still created by artists empirically. Some applications need a 3D model that looks like one specific real person, such as a sport game that needs 3D models of real players, a medical care application that may want the doctor to look like a typical doctor or a system that supports remote presence through a look-alike human surrogate. The artists normally use one photo of the real person or multiple photos from different view points as references. How close the model and the real person end up looking depends on the skills of and tools available to the artists. Furthermore, creating these models is time consuming and expensive. If we want to create a 3D human model for a specific participant in the application, it is currently challenging to get one quickly, e.g., in a few minutes.

In scene or object reconstruction systems, 3D scanners or time-of-flight (ToF) cameras are normally used to collect real world data. In order to reconstruct an object with 3D data only, scans from different view angles of the object are required to collect all the necessary data. Three techniques are normally used for the data collecting process: the first is to set up a multiple scanning device system and collect the data simultaneously; the second approach is to move the device around the object; and the last is to move the object. However, unlike buildings and other kinds of static objects, the human body is complicated and has many moving parts. Since a participant will not be able to hold the same pose during the data collecting process, it is a challenge to align the real world data from different view points. Even with a perfect alignment algorithm, the reconstructed 3D human model is just a 3D surface without skeleton, so users cannot control the

generated model by mapping the skeleton of the model and corresponding tracking data.

Our goal is to create a 3D model, which looks like a specific participant and can follow the participant's movement. The motivation is to create a 2-way inside interaction mixed reality system, in which the displayed participant can manipulate the virtual world and the virtual object(s) can affect the participant. In order to achieve this MR system, the first and essential stage is to generate the 3D human model of the participant in the scene. A photorealistic 3D model with a skeleton of the participant can provide precise collision detection, be controlled by the participant, and be rendered naturally in the MR scene.

In this dissertation, we present a novel 3D human modeling technique for using depth data and human tracking data from a Microsoft Kinect and the associated SDK to deform a 3D human template. There are existing techniques that are able to create 3D models of an object from 2D input, but they need user input for their modeling algorithms, like drawing the contour or providing a trimap of the object on top of an image. Our method is able to get the contour of a participant by automatically generating the trimap from the Kinect human tracking result and applying a natural matting algorithm that takes this trimap as input. The human template deformation is driven by matching each anchor vertex on the template to a point on the 2D contour. In order to improve the accuracy of the matching process, the anchor vertices are grouped into eight sets and the contour is segmented into eight corresponding segments before establishing correspondences between points on each segment and vertices in each anchor set. After acquiring the 3D geometry of the participant, we can add texture to the human model from the texture map provided by the color image from the camera. The proof of concept is developed using the Microsoft Kinect, but the algorithms and processes developed here can be employed with any device having similar capabilities.

1.2 Definition

Participants are a special group of users, who will be displayed inside the scene. They can interact with the virtual object(s) naturally by moving their body. This means that the participant(s) is part of the system, rather than being external to it, and can therefore interact with the virtual world with his/her own body instead of through some extra input device, like keyboard or mouse. Our system will generate 3D human models for these users.

Human Tracking includes human whole-body tracking and face tracking. The whole-body tracking is the overall tracking of the participant and the movement capture of the head, torso, and limbs. The face tracking contains facial feature tracking as well as the orientation and position of the whole face. What is not included by the above definition is hand gestures.

Mixed Reality (MR) merges the real and the virtual worlds into one visual display, and falls somewhere along a virtuality continuum[59]. The virtuality continuum covers concepts from a purely real environment to a totally virtual one.

2-way inside interaction refers to the interaction between the participant in the MR scene and the virtual world. What we are talking about is not the interaction between the MR world and users beyond the system, such as a user who moves a virtual object without being displayed in the scene. The inside interaction is 2-way, since not only can the participant manipulate the virtual world, but also virtual objects are able to affect the participant.

Trimap is a mask map used for the image matting algorithm. The map is made up of three tones: black, white and gray. Normally, black represents the known background pixels; white represents the known foreground pixels; and gray represents the unknown pixels to be determined by the matting result.

Anchors are special vertices on the template models. The new position of these points is inputted by the user or matched to some real world data at the beginning of the deforming process. For example, some approaches require the user to drag the anchors to the target manually. The locations of the rest of the vertices are determined by the new positions of the anchors and the reshaping algorithm.

1.3 Objectives and Difficulties

People like being involved in a mixed reality (MR) system and interacting with the virtual world naturally with their own bodies. During our experience testing MR applications, we have encountered many users, especially young children, who love to enter the MR scene and try to interact with the virtual objects. In these simple MR tests, there is no detailed human tracking system, which means these participants can only be in the background or foreground of the virtual objects. Therefore, many unnatural situations happen, like a virtual animal floating on the participant's body or a virtual car passing through the participant. The users find these odd scenes funny but distracting, since the rules in our real world are violated in the MR experience. Despite the momentary humor, we do not want users to observe these unnatural situations, as they disrupt the user's sense of presence in the MR world. The initial motivation for choosing this dissertation topic is that we want to achieve a 2-way interaction between the participants and the virtual objects, so they can be seamlessly merged in the MR scene.

With current devices and techniques, we can track real objects and register them within the virtual world to determine whether the real objects will be background or foreground as well as whether there are any collisions between the real and the virtual objects. Real-time video matting algorithms are able to composite the real and the virtual objects into one scene. Improved illumination methods are created for MR specifically in order to generate correct shadows and lighting between

the real and the virtual worlds. Therefore, a real object can stand in front of or hide behind the virtual object(s), partially or wholly occluding or being occluded, based upon their relative positions, and participants can manipulate the virtual object(s) as well.

However, no existing techniques work well for complex objects, like human beings. The complex shape, different body parts and dynamic articulated movements of a person cause lots of problems that won't occur with static rigid objects. For example, a participant might hold a virtual ball in his/her hands, which might be in the foreground of the ball while the body is in the background of the ball. Without a precise 3D model, skeleton tracking, and matting of the participant, you will see parts of the fingers inside the ball, or the ball floating without being attached to the hands. Furthermore, there is still one missing requirement to fill the gap between the real and the virtual world. How could the virtual object(s) affect the participant in the display? At the moment, you can program the animal and the car to avoid the kids in the scene with human tracking and collision detection methods. How about a participant walking towards a virtual wall? How does one avoid the collision?

One possible solution is to create a 3D model of the participant that represents the participant in the scene. When there is no conflict between the participant's movement and the virtual world, the generated 3D human model is controlled by the participant. When there is conflict, the human model won't follow the participant's movement, but move based upon the constraints of the whole scene. The simplest case is that, when the participant comes to a virtual wall and keeps moving forward, the 3D model will stop in front of the virtual wall instead of walking through it. With this, we can keep the presence of the participant without breaking any physical rule. There are other possible schemes to achieve 2-way interaction in MR system, but the first stage is always creating 3D human models of the participants. Although introducing a participant into an interactive MR scene is the initial motivation, the usage of our research in this paper is not limited to MR applications. 3D human models are also used in the wide domain of virtual-only applications. You may

want to be Michael Jordan or be yourself in a basketball video game, or see your colleagues as 3D models in a remote conference.

This dissertation will focus on the automatic 3D human modeling, so our first objective is to finish the whole process automatically without any need for a user to retouch the input data. The next objective is to convince the users that the generated 3D human model is real. According to Ferwerda [31], three varieties of realism are considered in Computer Graphics: physical realism, photorealism, and functional realism, so we need to consider the following problems for each aspect of realism.

• The idea of physical realism is that the geometry of the 3D model and the shape of the participant are identical in different poses from different view angles. However, this is imposible without precise simulation of the human muscles, since the human body is not a rigid object and the different poses will cause shape change. The 3D model geometry generated by our system matches the contour of the participant in the initial pose from the view point of the Kinect color camera and is close to the participant's real body from other view points.

• After including the human model, the resulting scene needs to look natural. The virtual object(s) must not stand out from other objects. Our system will process the color image of the participant and set it as the texture map of the generated 3D human model. With the assumption that the lighting is consistant and unchanged when we create the human model and render it, the rendering result will be identical to the real participant because of the image texture.

• Functional realism requires natural interaction between the 3D human model and other objects in the scene. The human template used in our modeling pipeline has a skeleton and all the vertices are attached to the skeleton. The model reshaping process not only changes the vertices' positions but also keeps the relationship between the vertices and the skeleton, so the human model can be animated by moving the skeleton. Furthermore, we can match the model's skeleton to the Kinect tracked human skeleton, so the 3D human model can be naturally controlled by a participant.

The thesis posed in this dissertation is that the contour of a 3D human model is a critical element for people to believe that the model represents a specific person when placed in a virtual world; moreover, we pose that contour techniques originally designed to smooth hand drawn sketches provide an excellent basis for developing precise body contours from imprecise approximations provided by low-cost 2D scanners.

1.4 Structure of Dissertation

This dissertation has seven chapters in total. The remaining six chapters will be organized in the following manner.

Chapter Two LITERATURE REVIEW: This chapter examines technologies related to the dissertation, including morphable 3D human models, human modeling, matting algorithms, and human tracking.

Chapter Three IMPLEMENTED ALGORITHMS: Here we present existing technologies employed in this dissertation along with the researcher's theoretical assumptions. This chapter will begin with an overview of the principles of choosing these approaches, and be followed by detailed descriptions of these technologies, including human tracking, single image matting, contour matching, and 3D model reshaping.

• Human Tracking presents the devices and detailed techniques of human skeleton and face tracking used in this dissertation.

• Single Image Matting describes the natural image matting algorithm to get the contour of the participant.

• Contour Matching is about the matching between the contour points of a participant and the anchors of the 3D human template.

• 3D Model Reshaping presents the algorithm that is able to deform a 3D template based upon moving anchors, while keeping the salient features of the template.

Chapter Four ISTRAW: This chapter reviews the IStraw [96] corner finding algorithm, which will be used for the contour segmentation process to improve the matching between the contour and the anchors. IStraw is developed upon ShortStraw algorithm, so we review ShortStraw at the beginning. We then discuss the issues that motivated and the changes that were implmented to create the IStraw algorithm. Finally, we present an evaluation process and the results that support our claims that IStraw is a significant improvment over ShortStraw.

Chapter Five METHODOLOGY: This chapter presents novel algorithms developed as part of the research contributions of our human modeling system. The most novel result here is the development of efficient algorithms to create a 3D human model of a participant correctly and automatically with a new human segmentation algorithm based upon modified IStraw.

Chapter Six EVALUATION: This chapter presents the interface design of our system, test process, and the evaluation results. We compare the 3D models generated from our human modeling system with the results from Kraevoy's [48] to justify the advantages of our system.

Chapter Seven CONCLUSIONS: In this final chapter, we summarize the contributions of the research associated with this dissertation and provide suggestions for possible future work, including ones that might improve on our human modeling result and those that might result in the development of associated MR applications.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

3D human modeling is the problem of creating a 3D human model from real world data of a specific person. There are a number of existing approaches to solve the problem with a variety of devices and techniques. In this chapter, we will review the existing devices and techniques for human modeling. As we do so, we will put these other efforts into context as regards the method presented in this dissertation. Our approach deforms a human form template model based upon real-world data collected by the Kinect. To do so, we need three fundamental components: one or more templates of 3D human models; real-world data of the participant; and a reshaping algorithm for fitting the real-world data to the template. Different approaches can be used to get and process these datas.

In this chapter, we will review research related to addressing the following problems, which are essential to the success of the methodology used in this dissertation.

• human shape capture devices – What are the available devices that can collect the real world data of the participant for shape deformation?

• human modeling approaches – How do we process the collected real world data to generate the 3D human model? The features of the collected real world data will determine the approaches that can be used for human modeling.

• morphable human models – How do we model a human face and body to enable human animation? How do we represent a digital human model that is able to be reshaped based upon real world data?

• human tracking – How do we track the participant in real-time? Human tracking information is important for both the data collecting and model reshaping process.

• matting algorithms – How do we get the silhouette of the participant to distinguish her/him from the background? Results from matting can provide contour information for model reshaping.
• reshape algorithms – How do we change the geometry of a 3D template by changing some controlling points or parameters? 3D model reshaping algorithms are one kind of human modeling approach.

2.2 Human Shape Capture Devices

The real-world data used for human modeling can be roughly divided into three groups: images from several unsynchronized devices, a number of synchronized camera images, or a monocular image. The first group needs to set up multiple cameras to cover the whole object from different points of view, and all the images from the cameras must be taken at the same time. The second group only needs one camera/scanner and the device (scanner) will move around the participant or the participant will move in front of the device (camera) to get different views. The last group also only needs one camera, but it only takes one image of the object. These three different kinds of devices for data capturing can be divided by the price range, data quality, and data type.

The first kind of device is a high quality 3D scanner, which uses either laser beams or structured light. The 3D laser scanner measures distance with controlled steering of laser beams. As shown in Figure 2.1, the structured light 3D scanner projects light patterns onto the scanned object and captures it with a camera system to measure the 3D shape of the object. These 3D scanners can provide good quality, high resolution 3D information, but they have limitations and shortcomings. Traditional scanning devices based on structured light or laser scan often cost around $50,000 and require professional operation. The scanning process will take several minutes or even longer. During the scanning process, the participant needs to stand still, and the 3D scanner will move

around the participant to get the 3D point cloud from different views. Generally, the human face only needs to be scanned once, such as [19]. However, multiple views are required for a whole-body scan, and some extra marker will be introduced to track the human pose, like the process in [5][6]. Rocchini et al. [70] built a low cost 3D scanner based on structured light, but the device needs precise system settings. Furthermore, it can only scan objects located approximately at 1.3 meters from the artifact, which means the scanned object size is limited to approximately 1 meter. Their results showed that only one side of the upper body, excluding the head, can be scanned, so a larger and more complicated system will be needed for human whole body scanning.

The second group of devices are traditional color cameras only. [3][10][83][37] use data from multiple cameras to capture human pose and shape at the same time. Some research, like Pighin et al. [65], has taken images simultaneously from different views to create face models. Unlike a 3D scanner, these approaches can track human performance, which means the participant can move freely in front of the cameras. The limitation of these methods is that these images have no hint of depth information. Therefore, the background must be monochrome or static during the data capture process to distinguish the background from the participant. Some techniques use monocular image to get 3D body pose and body shape. Guan et al. [35] used the SCAPE database [6] and a single image to compute the initial pose, light direction, shape and segmentation of a person. Recently, Zhou et al. [105], using the approach in [48], were able to generate a 3D human model from a single image. These techniques also need monochrome background, else they require one to manually mark the background and the foreground.

The last group of devices are depth cameras, which are distance measuring hardware as the first group, but the cost is much lower and the quality of the data is lower as well. For example, Beeler et al. [15] used a consumer binocular-stereo camera to capture the 3D geometry of a human face. At the moment, the most popular one is the Microsoft Kinect, which dominates other devices with its price of only $250. The Kinect sensor provides more accurate depth map in closer range and

the related SDK has real-time face tracking feature, so some recent research [107][39] focused on automatic reconstruction of a human face from a single color image and its corresponding depth map from the Kinect. Weiss et al. [92] used SCAPE model and one single Kinect to estimate the human body shape. Tong et al. [82] developed a scanning system with multiple Microsoft Kinects for capturing 3D full body models.

2.3 Human Modeling Approaches

2.3.1 Introduction

There are many existing systems that are able to create 3D model of static artifacts, like furniture, buildings, etc. Human modeling is a special case of 3D modeling or reconstruction and it takes advantages of previous work to model human beings. Based upon whether the collected real world data is 3D or 2D, the human modeling approaches can be grouped into two categories. Those employing the first kind of approach process 3D points and use a triangulation algorithm to create a mesh. While the second category uses the 2D data to extract the contour of the participant and then deforms a template human model with the 2D contour.

This section explores the previous work in each category and their pros and cons. The ideal approach should be low cost, fast at data collecting and processing, as well as precise with respect to the ground truth. However, there is no silver bullet, so how to tradeoff the price and the accuracy is based upon the requirement of a specific application. In our case, we prefer an affordable, fast, and reasonably accurate method.

2.3.2 3D data input

In order to get the complete 3D data of a human being, the device(s) must collect 3D data from different point of views. These systems can be set up in three ways to cover the whole body. The first is moving one device around the participant, the second is setting up multiple devices around the participant, and the last is asking the participant to move in front of the device.

Some devices, such as laser scanners and depth cameras, can move around an object to get data from different views and then align these 3D points. Theoretically, we can move a laser scanner to surround a participant and collect precise 3D point cloud of the person. However, the participant cannot move during the whole scanning process. Normally, it takes at least one minute to run one scan and 2 minutes between each scan. If we took three scans to cover the whole body, the process needs 7 minutes at least. It is impossible to require a participant to hold a pose for 7 minutes, so it is impractical to use a laser scanner for real people. Dr. Paul Debevec mentioned in [69] that his lab at USC only takes two seconds to scan a face by using a structured light system, but it only works for front surface and limited scale. If we use this system to scan a human body, it requires at least 10 scans to cover the front surface and 3 point of views to cover the whole body. Suppose it takes 2 seconds to move the participant to the right location, the whole scanning process at least need 2 minutes. Microsoft Research has the KinectFusion project [40], which is able to move the Kinect through space and perform high quality 3D surface reconstructions. Although KinectFusion is easier to move and scans much faster than a laser scanner, the participant still needs to hold the pose for a while during the scanning process. Furthermore, these approaches need precise tracking of the device.

The second way is setting up multiple devices in the environment to scan the participant at the same time. Compared to previous methods, these approaches can save scanning time. The problem is that the overlapped scanning area will cause problems. Like a laser scanner, the Kinect also emits

laser beams, and multiple devices mean that one device will receive signals from the other devices. However, unlike the laser scanner, which gets distance by computing the time interval between the laser beam emission and receiving, the Kinect uses the pattern of the signals to measure the depth information. Schroder et. al. [73] introduce the revolving disks to rotate the Kinects in order to get rid of the interference. Some systems [23][8] set the Kinects far away and with sertain facing direction to minimize the interference between the devices. Theoretically, it is also possible to set up a super structured light system around the participant with a large number of lights and several cameras. However, this body scanning system will be very expensive, and hard to set up.

The last kind of approaches requires the participant to move instead of the device. In order to track the human pose during the movement, Allen et al. [5] and Auguilov et al. [6] used extra markers in a structured light system or laser scanner for whole-body modeling. The system created by Cui et al. [30] uses a single Kinect and requires the participant to turn around 360 degrees for 20 to 30 seconds while maintaining an approximate ”T” pose. They use overlapping areas for registration and then reconstruct the 3D human model from the depth data. Li et al‘[52]. used similar approach, but generated seamless global textures using Poisson blending. Tong et al. [82] set up three Kinects around the participant. Like some previous work for object scanning, they ask the participant to stand on a turntable and get depth information from different view points, since the turntable can simplify the scanning point alignment. There is no overlap area between the Kinects, so essentially, this system belongs to the third group. This multi-Kinect system is able to get more precise depth data by putting the sensor closer to the participant.

Overall, human models created from 3D data input are accurate, but the data collection process is hard for the participants. They need to hold a pose for at least from 20 seconds to 7 minutes based upon the system design. Furthermore, these models are impossible to be controlled by users, since they have no skeleton to which the vertices can attach. Therefore, they can only be used in applications that do not need any interaction between the human model and other objects.

2.3.3 2D data input

2D real world data for human modeling are the silhouette(s) of the participant from color image or depth data. Based upon the usage of the 2D data, there are two types of modeling approaches. The first type is called Shape-From-Silhouette (SFS), which constructs a 3D shape estimate of an object using silhouette images of the object. The second type is reshaping based approaches, which deform an existing 3D model based upon the silhouette.

In 1971, Baumgart first introduced the idea of using silhouettes for 3D reconstruction in his PhD dissertation [12]. Cheung et al. [25] summarized a large number of research projects in computer vision that use multiple silhouettes from different view points to generate the model. Like results from 3D data input modeling, the human models generated by these approaches have no skeleton for animation.

For the second group of approaches, the silhouette vertices of a 3D human template are matched to the captured silhouette points and a deforming algorithm reshapes the geometry of the template to meet the real person. Some approaches use one color image or depth map to get one silhouette of the human body. In the system created by Kraevoy et al. [48], users can draw the contour on the color image of a person and then get the 3D human model of the person by deforming a 3D template with skeletons. Other systems use multiple silhouettes to address the ambiguity problem of using one silhouette. Weiss et al. [92] created a body scanning system with a single Kinect, requiring the participant to rotate in front of the sensor. This system works much slower than Kraevoy's system, since it needs longer scanning time and extra silhouette extracting time. Furthermore, the participant needs to hold the pose during the scans and follow the correct movement.

2.4 Morphable 3D Human Models

Morphable 3D human models have been studied for a long time in both the computer graphics and computer vision fields. In 1972, Parke [64] did pioneering research on 3D face models, which are cartoon faces with facial expression animation by analyzing a set of real face images. Thereafter, a large number of face models and whole-body models have been developed. Morphable 3D models for human face and body are the fundamental components of the reshaping process and human animation. After changing some parameters of these models, the shape and the pose of the models is deformed automatically but still keep some features of the original model.

There are two different classes of morphable human models based upon the generating process of the model: empirical knowledge-based and statistical learning-based. The models from the first class are created by artists with their experience and some reference images. Whether the model and the animation look real relies on the skill of the artists. This kind of model digitizes a 3D mesh through geometric units representing muscles, tissues, and skin. One of the prevalent models is the surface model developed by Magnenat-Thalmann and Thalmann [81] in 1987. It is a conceptually simple but powerful model, that contains a skeleton and outer skin. In order to support animation, the vertices on the skin mesh are attached to the skeletons, so the skin will deform corresponding to the movement of the skeleton. Some reshaping algorithms [66][102][6] use these face or body models for geometric change by fitting some vertices to corresponding real-world data.

The model in the second class contains a 3D mesh and a mathematical function generated with statistical learning of real human beings with different appearances. The features of human beings are obtained during the statistical learning process and set as input parameters of the function. Animations are enabled by changing these parameters, which control the 3D mesh to present the result. During the reshaping process, these parameters are extracted from the real-world data, including 3D geometry and texture information collected by laser scanners or multi-camera system.

One of the most famous face database examples is developed by Blanz and Vetter [19]. They created a multidimensional morphing function by analyzing statistics of a large dataset of 3D face scans with both geometry and texture data, and derived some face attributes such as gender, fullness, ”hooked” noses or the weight of the person.

Unlike face models, which can support different expressions and appearances at the same time by changing the shape of a template face, statistical learning-based whole-body models are more complicated, because of the combination of various poses and shapes. Allen et al. [5] analyzed the whole-body range scans of a wide variety of individuals, and demonstrated consistent parameterization of the 250 models. Kraevoy and Sheffer [47] created a mean-value geometry encoding algorithm to encode any 3D model by setting some anchor points. The encoded model can be deformed by decoding with changed anchor points, and the result will fit the new anchor points and keep the shape feature of the template model.

2.5 Human Tracking

2.5.1 Whole-body Tracking

Human tracking includes whole-body tracking and face tracking. Whole-body tracking is also known as human motion capture, which refers to tracking the overall human body as well as a skeleton structure with a number of joints. The term ”whole-body tracking” is used instead of ”human motion capture” to distinguish it from small scale body movements, such as facial expressions and hand gestures. A whole-body tracking system consists of two subsystems: a sensing system and a processing system, and the complexity of these subsystems is related. Typically, if one of the subsystems is highly complex, the other will be correspondingly simple.

There are two classes of sensing subsystems: passive sensing, which is based on natural signals,

like visual light or other electromagnetic wavelengths; active sensing, which places devices on the tracked object and/or in the surroundings to transmit or receive signals. Passive sensing normally uses RGB cameras to get natural signals, and sometimes attaches markers to the participant to ease the processing subsystem. Markers are not as intrusive as the devices used in active sensing, since they do not produce any signal by themselves. Active sensing is used for applications situated in well-controlled environments, and the corresponding processing subsystem is simpler than what is used with passive sensing.

Generally, passive sensing approaches are computer vision-based by analyzing one or multiple video sequences from RGB cameras. Moeslund and Granum summarized these human body tracking approaches in the survey [61] and proposed a general structure of a motion capture system for videos as four steps: initialization, tracking, pose estimation, and recognition.

• Initialization step covers the actions needed for a correct interpretation of the current environment, like offline camera calibration, scene characteristics calculation, the model representing the subject, and the initial pose of the subject. Moeslund et al. [62] reviewed the approaches for the initialization of kinematic structure, human shape, and appearance. These intializations are essential to the pose estimation, and many algorithms still use a manually initialized generic model and pose. Some model-based approaches need the participant starting with an initial pose, specifying the pose of the first frame, or labeling the poses of key frames. Some systems try to find the initial pose automatically, such as [71], [77]. The problem of fully automatic initialization for human pose estimation with monocular image sequences remains open.

• Tracking step refers to establishing coherent relations of the subject between frames to extract specific image information for the next step, pose estimation. In [62], there are two processes in tracking: figure-ground segmentation and temporal correspondences. The former is the process of extracting the humans from the background, and the latter will associate the detected humans from frame to frame.

• Pose estimation is the process of identifying the configuration of the underlying kinematic or skeletal articulation stucture of a person and it can be broken into three class: model-free (i.e. points, boundary boxes), indirect model use (i.e. aspect ratio between limbs, pose recognition), and direct model use (3D mophable human models). A large number of research efforts over the last half decade [10] [4] [41] use 3D morphable human model(s) to reshape and track the human simultaniously.

• Recognition step is a post processing phase to classify the captured motion as one type of action. There are different ideas and approaches depending on the goal of the researcher and applications for activity recognition.

Active sensing systems can collect more information with active sensors, and they can be classified into two groups based on with or without on-body active sensors. Some systems set external sensors in the environment and attach sensors to the human body in order to collect movement information. There are different categories of on-body sensors, which can be categorized as mechanical, inertial, acoustic, radio, and magnetic based sensors. These sensors have their own advantages and limitations for different environments. For example, some augmented reality systems use inertial sensors and vision tracking for registration (i.e. [101]), or glove-based devices for analysis of hand gestures (i.e. [22]). The second group of systems only set up active sensors in the environment without any on-body sensors. Microsoft started a revolution in this area by introducing the Kinect on November 4th, 2010 and the free official SDK for non-commercial use on June 16th, 2011. The depth camera in Kinect uses projected infrared signals to generate depth images, and the Microsoft Kinect SDK can provide real-time human body and face tracking. Microsoft researchers, Shotton et al., introduced real-time human pose recognition from single depth images in [76]. Some research projects, like [16][100]citeAsteriadis13, tried to set up a multi-Kinect system to improve the human motion tracking acuracy.

2.5.2 Face Tracking

Face tracking, also known as facial performance capture, includes the tracking of the head as well as the facial expression. The head tracking information can be obtained from whole-body capture results, but extra facial features can improve the accuracy of estimating the orientation, scale and transform of the head. Like whole-body tracking, there are also two kinds of facial performance capture devices: passive sensors and active sensors. A large number of previous research uses these devices for applications in many domains. These applications have different requirements, such as high accuracy, robustness, or real-time execution.

Generally, passive sensing systems use a single or multiple RGB cameras, and can be classified into two categories: marker-based and marker-free systems. Marker-based techniques (i.e. [36], [53], [57]) are designed for greatest possible accuracy for movie production. However, these approaches cannot provide natural texture of the subject because of the markers on the face. These systems can only track 2D feature points and require that the markers be carefully placed. For the realistic modeling of the human face, marker-free techniques (i.e. [103], [20], [56], [15], [21]) use multiple color cameras or structured light 3D scanner to get depth information of the human face. These systems need a specified setup and work better under controlled environments, such as the structured light system needing to be operated in a dark room to get rid of extra lights.

Active sensors used for face tracking are set in the environment instead of some on-body devices. One reason is that active sensors are too big or heavy to be attached to the face, and the other is that the subtle movement of the face is hard to be detected by on-body devices. Microsoft Kinect has both a passive sensor provided by an RGB camera and an active sensor system consisting of an infrared laser projector combined with a monochrome CMOS sensor,. The current Kinect sdk also has a face tracking engine that takes both color images and depth images as input for face tracking. Some applications take advantage of the this face tracking. For example, Weise et al. [91] tracked

the facial expressions of the participants with the Kinect in real-time and mapped these to control a digital character.

2.6 Matting

2.6.1 Introduction

Matting algorithms extract a foreground object from an image by estimating the opacity of the object at each pixel, and the generated opacity data is typically called an alpha map. With an alpha map, the foreground object can be composited with another background. Our system uses different matting algorithms to get the silhouette of the participant for modeling and to render a mixed reality scene. For example, if a virtual object appears behind the participant, he/she should be distinguished from the background and the virtual object should be rendered between the participant and the background. Based upon the usage, the requirements for the matting algorithms are different. Human modeling will be preprocessed at the beginning of running the system and the matting result determines the accuracy of the human model. Therefore, this matting algorithm needs to provide a precise silhouette, and the computational complexity is not that important. On the other hand, matting algorithms for composing an MR scene must be run in real-time, and be as accurate as possible.

In 1984, Porter and Duff [67] introduced the alpha channel and the most common compositing equation:

$$C_i = \alpha F_i + (1 - \alpha) B_i, \tag{2.1}$$

where C_i, F_i, and B_i are the composite, foreground, and background colors of each pixel. The quantity α is the pixel's foreground opacity, and its value is between 0 and 1. This is a severely under-constrained problem since for a three-channel color image, there are three equations, one for

each channel, and seven unknowns at each pixel.

In order to extract foreground element from a natural background, extra inputs are necessary for the initial ground truth of some known pixels as 1 for foreground and 0 for background. A large amount of research efforts [17], [72], [28], [33], [80], [34] start with manually segmenting the image into three regions: foreground, background, and unknown, which is known as a trimap, and then estimating F_i, B_i, and α for the pixels in the unknown region. Some approaches [84], [34], [87], [50] use a sketch-based interface for interactive matting, and only a few strokes indicating the background and foreground pixels are needed as input instead of a trimap. Other approaches, such as [90], [55], [13], and [14], take advantage of depth information to generate the trimap automatically.

2.6.2 Single Image Matting

Matting techniques for a single image try to solve the matting problem of one color image with additional constraints in the form of user input, like trimaps or scribbles, to identify known foreground (i.e. $\alpha = 1$) and background (i.e. $\alpha = 0$). Wang and Cohen [89] classified these methods into three classes: color sampling, pixel affinities, or a combination of the two.

Color sampling methods use sample points of known foreground and background to estimate the α value of the unknown area. According to the image statistics, the color of an image is locally smooth. Therefore, the true foreground and background colors are close to samples of nearby known foreground and background. There are two subclasses [89]: parametric sampling methods and nonparametric sampling methods. Once samples are collected, the former methods usually fit low order parametric statistical models to the samples, such as Bayesians [28] and Gaussians [72]. These models work well for images with smooth regions and distinct foreground and background colors, else it will generate large fitting errors. Some previous work, like [60], [17], [84], [34],

[86], and [87], uses nonparametric methods to estimate the alpha of unknown pixels with the nearby samples. He et al. [38] proposed a new nonparametric sampling-based matting algorithm, which considers global samples to avoid missing true samples.

To avoid misclassification of color samples in a complex scene, another way of using local image statistics is proposed by defining various affinities between immediately connected pixels or pixels in a 3x3 window. A large number of recent research results use different affinities to deal with complex images. For example, poisson matting [79], random walk matting [33], geodesic matting [9], fuzzy connectedness for matting [104], closed-form matting [50], and spectral matting [49]. These approaches focus on estimating alpha values first instead of estimating alpha, foreground, and background of unknown pixels jointly. Another problem is that the alpha matte is estimated in a propagation fashion, so small errors can produce more significant errors.

The last class of methods use energy functions to combine sampling and affinities and can be classified into two sub-classes based upon the optimization methods: non-closed-form optimization and closed-form optimization. Wang and Cohen [84] developed the iterative matting approach by modeling the matte as a Markov Random Field (MRF), a non-closed-form optimization. The second sub-class includes newly proposed approaches, such as the Easy Matting system [34], the Robust Matting system [86], real-time Shared Sampling system [32]. By combining sampling-based methods (more accurate) and affinity-based methods (more robust), it is able to achieve a good trade-off between accuracy and robustness.

2.6.3 Video Matting

Video matting is the process of extracting a dynamic foreground element against a background. The same techniques for a single image can be transferred to video matting by processing each frame. This sub-section will only focus on methods for video matting with a natural background.

All techniques for the natural image matting can be extended to video matting by hand-marking each frame with either a trimap or scribbles. However, this naive approach is time-consuming for a long video sequence and the matting results are temporal incoherence. At the same time, videos have some advantages, like a more complete background model, as in [7], and easier to determine edges, by comparing neighboring frames.

A larger number of approaches applied a two-step framework: the first step is generating a trimap for each frame, and the second step uses the trimaps and a matting algorithm to refine the foreground boundary. To address the temporal incoherence problem, most techniques perform spatio-temporal optimizations during the trimap generation. Chuang et al. [29] use optical flow [18] to propagate user specified trimaps on a few keyframes to all other frames, and then adopt the Bayesian matting algorithm [28] for each frame. Some video matting systems applied a keyframe-based rotoscoping system [1] for trimap generation. Graph-cut optimization is widely used in video matting system due to its efficiency for spatio-temporal video object segmentation. There are two successful techniques, the video object cut and paste [51] and the interactive video cutout [85]. Bai and Sapiro [9] extended their geodesic segmentation and matting approach to video sequences. Beato presented a real-time video matting algorithm using modified Shared Sampling algorithm [90] in his dissertation [14].

2.6.4 Matting with Extra Information

Since matting for a single image or video is a severely under-constrained problem, any extra information of the unknown variables will significantly reduce the size of the solution space. In order to result in more accurately estimated mattes, there are many different kinds of approaches proposed, such as controlled background, flash matting, compositional matting, matting with camera arrays, and matting with depth information.

Some early matting techniques capture each foreground element in front of a controlled background, like a blue screen, or other known background. Blue screen matting systems [78] [60] set the background color to be constant, normally blue or green. Vlahos [98][99] introduced some constraints between green and blue channels in his patents to make the problem tractable. Smith and Blinn [78] summarized these constraints nicely, but their approach requires an expert to tune the parameters. In order to get rid of the tuning process. Mishima [60] sampled the foreground and background to calculate the alpha map. Unlike blue screen matting, the background of difference matting is a non-constant color but still a known one. The foreground element can be extracted by taking the difference between the image with and without the element. For example, Qian and Sezan [68] determined α to be 0 or 1 based on a threshold, and the mattes are jagged without some smoothing process, which still cannot compensate for gross errors.

If the background scene is far enough away from the foreground element, the most noticeable difference between the flash image and the non-flash image of the same scene is in the foreground element. Sun et al. [80] created the flash matting approaches to process flash and non-flash image pairs based upon this principle. This approach works for complex foregrounds and backgrounds, but the system will fail if the scene is moved or the appearance of the foreground is not dramatically changed by the flash.

Generally, matting approaches treat the matting and compositing processes separately: estimating a foreground matte at first, and then recomposing the foreground element onto a new background. In [88], the compositional matting system is proposed, and it is the first approach to integrate matting and compositing into one optimization process. There are some similar systems, like the photomontage system [2] and the drag-and-drop pasting system [42]. These systems will carry some original background to the composed novel image, therefore they only work well when the new background has regions that are similar to those in the original one.

McGuire et al. [58] created a multi-sensor camera to capture three synchronized video streams with different focuses, and this approach can generate the trimap automatically. Instead of using multi-sensor camera, another approach [43] applied a camera array to extract high quality foreground mattes from video streams by capturing foreground with different parts of the background scene.

A large number of approaches capture the scene with both color and depth cameras. The depth image will provide initial segmentation of the foreground element, and then generate the trimap automatically. During the natural image matting process, Wang et al. [90] use the depth information along with the Bayesian or Poisson matting in the error minimization step to get rid of the undesirable artifacts. Beato [13] [14] improved Wang's work in [90] to get better performance and also make the algorithm more tolerant to video matting. Depth information is combined with other alpha matting approaches, such as the closed form matting approach [106] [46] [27], robust matting approach [26], and for both single image and video matting. Cho et al. [27] developed an adaptive method to generate trimaps instead of morphological operations to grow unknown regions. Previous techniques need accurate user-specified thresholds for the segmentation and/or accurate depth map. Lu and Li [55] proposed an image matting system using the Kinect with the depth image enhancement techniques in [24] to remove the unavoidable noise and holes in the depth image.

2.7 Reshaping Algorithms

After getting a human template and the real-world data of the participant, some reshaping algorithm is required to fit these data to the template and optimize the generated new model. Currently, the techniques for human reshaping fit the scanned 3D point cloud or silhouette to a 3D template mesh. Depending on the input real-world data, there are three different kinds of approaches for human reshaping: 3D-based, image-based, and a combination of the previous methods.

3D-based human reshaping technologies use 3D point clouds scanned by high quality 3D scanners or cheaper depth cameras. Allen et al. [5] formulated an optimization problem for fitting high-resolution template meshes to detailed human body range scans, 3D geometry data, with sparse 3D markers. This approach only works for individuals under the same pose, Anguelov et al. [6] introduced the SCAPE model, which covers variations in both human shape and pose. They used markers for pose deformation with a non-rigid surface deformation function, and data from a laser scanner for a non-rigid and a rigid component of the shape deformation. The approach created by Zhang et al. [102] is able to reflect the shape and behavior of a human face with data from a constructed light scanner system. These approaches normally need expensive devices and computationally-expensive optimization algorithms to get photorealistic models. Rocchini et al. [70] created a cheaper constructed light system to capture geometry of objects with limited size.

Image-based methods fit the template to silhouette or other features from one or several 2D images. A large number of approaches [45][3] in the computer vision area use images from a single or multiple cameras to get both the pose and shape of a human body by extracting a human silhouette from these images. Kraevoy et al. [48] introduced a fast reshaping algorithm to fit a contour drawing to an encoded model [47]. Beyond silhouette and feature points, some research, like Kemelmacher-Shlizerman and Basri [44], tries to use shading information in the image(s) to create more accurate 3D face or body models.

The last group of methods combine 3D geometry data and silhouettes from images to create better models. A great deal of related work created algorithms for new devices, which are able to provide both depth data and color images, such as Beeler et al. [15] use of a stereo camera, and Hernandez et al. [39] use of the Microsoft Kinect for face modeling. These devices provide both 3D geometry point cloud as well as color image(s), which contains the human silhouette, facial features, and shading information. Other approaches integrate different technologies vertically to take advantage of human model databases created by previous work. Since most of the time con-

suming preprocess work has been done during database construction, these approaches can get more accurate result with noisy input by matching the input data to a model in the database. For example, [10][11][35][105] obtained both human shape and pose by fitting the SCAPE model [6] to video(s) or a single image. Recently, there are a large number of research efforts [92][82], that use SCAPE model and Kinect(s) to generate human whole-body model.

Figure 2.1: A low cost structured light 3D scanner [70].

CHAPTER 3: IMPLEMENTED ALGORITHMS

3.1 Introduction

Our system will extract one silhouette of the participant and then reshape an existing 3D human template based upon the silhouette. The disadvantage of this 2D real world data based human modeling approach is that the contour from one view point will bring ambiguity to the generated model. However, we still choose this method based upon the following properties.

• **Fast** – Our approach is fast for both data collection and 3D model creation. Collecting 3D points or multiple silhouettes of the participant take extra time.

• **Easy** – The whole process can be finished by the participant alone. Using laser scanner or Kinect-Fusion requires extra help to setup devices or collect real world data to cover the whole body. With these other approaches, the participant needs to carefully hold a pose for several minutes or rotate in front of the device during the data collecting process, .

• **Flexible** – The 3D human model created with reshaping algorithms will be similar to the initial template, so it is able to have skeletons and be controlled by the participant or other users. However, models created from 3D point clouds only have triangular surfaces and need extra manual work to set up a skeleton for animation.

We took advantage of existing techniques, like human tracking, matting algorithms, and model reshaping. This chapter concludes all the previous work that we have implemented within our system and the reason why we choose these algorithms. The process of human modeling requires complex optimization and there is no real-time algorithm so far to generate a photorealistic 3D human model. In our system, the 3D human model can be created in a minute and controlled by the participant later. It is inappropriate for the participant to wait a while to see his/her 3D model

in the MR scene. Therefore, a trade-off between accuracy and efficiency needs to be considered.

One example of the trade-off is the tracking subsystem in our system. The Microsoft Kinect and its related SDK are used for real-time human body and face tracking, and provide acceptable but non-perfect human tracking. The Kinect cannot track body details (i.e. fingers) since the accuracy of the depth information varies according to the distance between the subject and the sensor. The locations of the feet tracked by the Kinect are not precise since it is hard to distinguish the feet from the floor in the depth image. Other limitations of the Kinect include its practical tracking range of around 3x3 square meters and the environment lighting, which cannot have too much infrared light or be too dark for the color camera. Nonetheless we still use the Kinect for human tracking, because it is inexpensive, easy to set up, has free SDK for research projects, and provides reasonable results.

Natural image matting is another time-consuming process. A single image matte of the foreground element provides a silhouette of the participant, the accuracy of which is essential to the success of the human body modeling, so the chosen matting approach needs to be accurate without the requirement of real-time execution. On the other hand, video matting is used to render the participant in front of any virtual object. Therefore, the matting algorithm must be fast and output a spatio-temporal foreground participant.

3.2 Human Tracking

3.2.1 Introduction

Tracking a real object in a MR system will register the real object in the virtual world, and enable the tracked object to interact with the virtual objects. The goal of our system is to make the two-way interaction between a real human and virtual objects possible. Therefore, human whole-body

tracking and face tracking are important to the success of this system for the following reasons:

• When the system starts, the tracked skeleton of the participant will be compared with the initial pose as in Figure 3.1. Once the pose matches, the system will trigger the process of 3D human modeling.

• The body tracking information is necessary to generate the trimap automatically for the matting algorithm, which is required for human modeling and the MR scene rendering.

• The face tracking can provide feature points of the participant's face, and these points will be fitted to the vertices on the human template during the 3D human face modeling process.

• Under the real mode, the tracked skeleton is used to control the virtual representative by matching the skeleton from the Kinect to the skeleton of the 3D human model. Although the virtual representative won't be displayed in the scene during the real mode, it will be used to detect whether there is any interaction between the participant and the virtual objects. If the virtual force is so strong that it should affect the movement of the participant, the virtual representative will replace the real representative and be displayed in the scene.

• Furthermore, tracking is required to determine when the system should switch the participant from the virtual representative to the real representative by comparing the skeleton of the 3D human model and the tracked model. During the virtual mode, the human skeleton is compared with the skeleton of the virtual representative to check the proximity between the virtual and the real representatives. Once the user moves back to the right position and pose, the system will switch back to the real mode

We use the Microsoft Kinect sensor for human tracking, because the device is inexpensive, robust and easy to setup in the environment. Another important reason is that Microsoft provides free non-commercial licenses to the Kinect SDK, which can track human body and face in real-time. Microsoft also offers some sample codes to show how to use the SDK to get raw data from Kinect (i.e. color images), processed data (i.e. depth images), and tracking result (i.e. skeleton

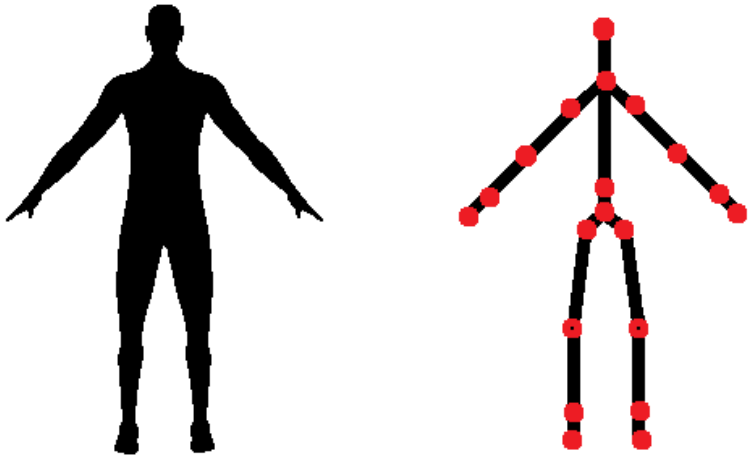

Figure 3.1: The initial pose of the 3D human prototype (left) and the matching Kinect skeleton (right)

information of the tracked body, feature points of the tracked face).

3.2.2 Tracking Devices

Microsoft published Kinect for their game station XBox 360 in 2010, and then published the official SDK to enable the device for personal Windows-based computers. Currently, Microsoft distinguishes the Kinect: one version for XBox 360 ($90) and the other for Windows ($250). The SDK only works with Kinect for Windows now. The price of a Kinect is affordable to everyone who wants to play with a depth camera. Figure 3.2 shows the structure of the Kinect sensor, which is a horizontal bar connected to a stable base. The three round objects on the bar, from left of to the right, are the infrared laser projector, the RGB camera, and the monochrome CMOS sensor (IR camera). The depth sensor of Kinect is composed of the infrared laser projector, which will emit infrared pattern signals to the space, and the CMOS sensor, which will capture all infrared signals

in the environment. By analyzing the pattern in the captured infrared image, the depth sensor can generate the depth video data under any ambient light conditions.

Figure 3.2: Kinect sensor.

The default resolution of the RGB video stream is 640 x 480 pixels, but its highest capable resolution is 1280 x 1024 pixels at a lower frame rate. The monochrome depth sensing video stream is 640 x 480 pixels with 11-bit depth, which provides 2,048 levels of sensitivity, but the depth data is not linearly proportional to the depth in the real world. The IR camera can provide raw infrared image data as well. The suggested practical range of the Kinect is from 1.2 meters to 3.5 meters, but the Kinect can recognize standing users between 0.8 meters and 4.0 meters. The angular field of view is 57° horizontally and 43.5° vertically, while the motorized tilt can rotate the bar up or down up to 27°.

3.2.3 Tracking Techniques

The Kinect software can simultaneously recognize up to six people in real-time, including two active users with motion analysis. These two users have 3D skeleton data with 20 joints per user as shown in the right figure in Figure 3.1. Microsoft researchers, Shotton et al., discussed the method used in Kinect for human body tracking in the paper [76]. This technique is capable of predicting

3D positions of body joints from a single depth image in real-time. This pose recognition system runs at 200 frames per second, but the bottleneck of the Kinect human tracking is the frame rate of the sensors, which is around 30 frames per second with 640 x 480 resolution.

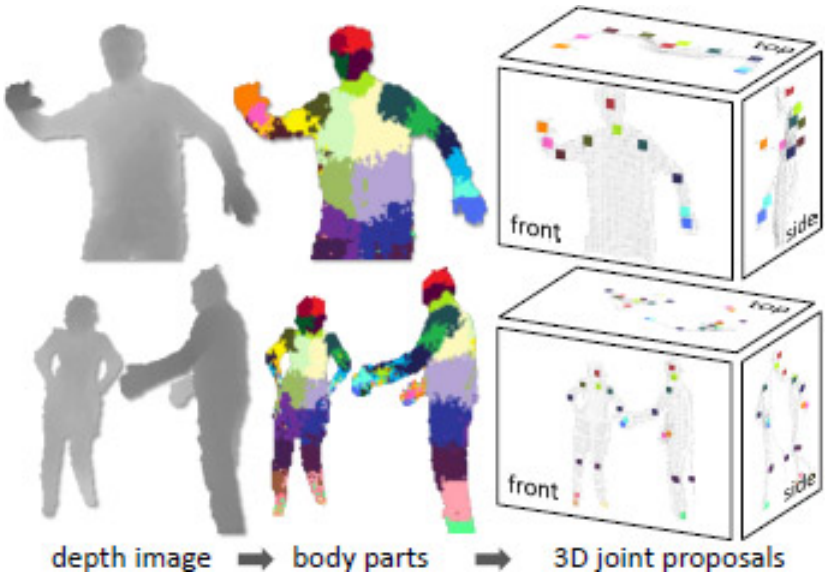

Figure 3.3: Overview of the Kinect human pose recognition system [76].

Microsoft Kinect SDK human body recognition system takes depth images from the Kinect, such as the single user example and multiple users example in the left part of Figure 3.3. Each depth image is segmented into dense probabilistic body parts, which are labelled for each pixel (see two examples in the middle of Figure 3.3). Then the inferred body parts are reprojected into world space to localize spatial modes of each part distribution. In the right part of Figure 3.3, local modes of this signal are estimated to give high-quality estimates for the 3D locations of each skeletal joint.

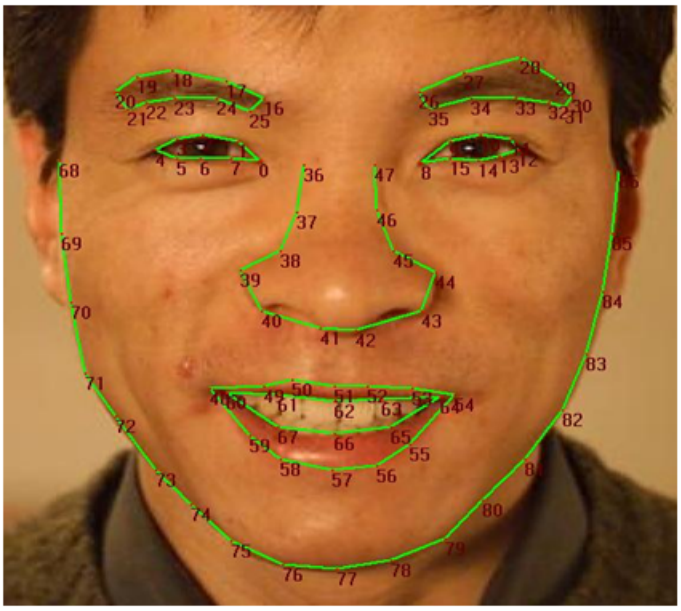

Figure 3.4: Feature points of Kinect face tracking (from Microsoft msdn website).

The Kinect enables face recognition in real-time since the publication of Kinect for Windows SDK 1.5 and 1.6. The tracking quality is affected by the input image quality and the distance between the face and the Kinect. Their face tracking engine analyzes RGB images and depth images input from the Kinect sensor, and provides the 3D head pose, 2D shape points, animated unites (AUs), and shape unites (SUs). The head pose includes head scale, rotation, and translation. These 2D points are feature points on the face, and are defined in the coordinate space of the RGB image as shown in Figure 3.4. In addition to the 87 points in the Figure, there are 13 more points, including the center of the eyes, the corners of the mouth, the center of the nose, and a bounding box around the head. There are six AUs expressed as a numeric weight varying between -1 and 1 to present the expression of the user. The eleven SUs, expressed as a 3D vector, include head height, eyebrows vertical position, eyes vertical position, eyes width, eyes height, eye separation distance, nose vertical position, mouth vertical position, mouth width, eyes vertical difference, and chin width.

3.3 Single Image Matting

Our system needs a robust and fast single image matting algorithm to extract the participant's silhouette from a Kinect color image. In order to get real-time data from the Kinect, we choose the low resolution setting 640×480 for 30 frames per second. The frame rate is only 12 for the high resolution setting 1280×960, so it is impossible to give the participant good feedback in the system. This low resolution color image will introduce a significant amount of noise, especially along the edges with obvious color changing. We also use the Kinect depth data to generate a trimap automatically and this process introduces extra noise or even errors. In 2011, He et al. [38] created a new sample based matting algorithms to consider global samples instead of limited samples within a window. This algorithm is fast, works better for low resolution input image, and handles noise.

3.3.1 Sample Selection Criteria

The criteria in [38] to select the sample pairs, whose linear combination will explain the unknown pixels, are based on both color and spatial distance between the samples and the unknown pixels. The alpha value $\hat{\alpha}$ of an unknown pixel I can be estimated with foreground sample F^i and background sample B^j as:

$$\hat{\alpha} = \frac{(C_I - C_{B^j}) \dot{(} C_{F^i} - C_{B^j})}{|C_{F^i} - C_{B^j}|^2}, \tag{3.1}$$

where C_I, C_{F^i}, and C_{B^j} are the color (in RGB space) of the unknown pixel, the ith sample from the foreground boundary, and the jth sample from the background boundary.

The color cost ε_c is used to describe how good a sample pair (F^i, B^j) fits the unknown pixel I and can be computed as:

$$\varepsilon_c(F^i, B^j) = |C_I - (\hat{\alpha} C_{F^i} + (1 - \hat{\alpha}) C_{B^j})|. \tag{3.2}$$

The color cost is the absolute value of the distance between the color of the unknown pixel I and the color combined with α, F^i, and B^j.

The spatial cost ε_s of F^i is represented as:

$$\varepsilon_s(F^i) = \frac{|X_{F^i} - X_I|}{M_F}, \tag{3.3}$$

where X_{F^i} and X_I are the coordinates of the foreground sample and the unknown pixel. The term $M_F = min_i|X_{F^i} - X_I|$ is the nearest distance between the unknown pixel and the foreground boundary. The spatial cost of B^j is defined similarly.

Depth information from the Kinect can provide another cost to increase the accuracy of the global sampling method. The depth cost ε_d is defined as the minimum depth distance from the unknown pixel to F^i or to B^j:

$$\varepsilon_d(F^i, B^j) = min(|D_{F^i} - D_I|, |D_{B^j} - D_I|), \tag{3.4}$$

where D_I, D_{F^i}, and D_{B^j} are the depths of the unknown pixel, the foreground sample, and the background sample.

Our final sample selection cost ε is a linear combination of the color cost, the spatial cost, and the depth cost:

$$\varepsilon(F^i, B^j) = w_c\varepsilon_c(F^i, B^j) + w_s\varepsilon_s(F^i) + w_s\varepsilon_s(B^j) + w_d\varepsilon_d(F^i, B^j), \tag{3.5}$$

where w_c, w_s, and w_d are the weight of different costs, and the sum, $w_c + 2w_s + w_d$, is equal to 1.

3.3.2 SampleMatch Algorithm

First of all, all the foreground samples F and the background samples B are sorted based upon their color. He et. al [38] denoted them as ordered sets $F^i|i = 1, 2, ..., n_F$ and $B^j|j = 1, 2, ..., n_B$. The goal is to find a sample pair (F^i, B^j) with the lowest cost for each unknown pixel $I(x, y)$. This current optimal sample pair is represented as $\Phi(x, y) = (F^i, B^j)$. Unlike [38], which initializes $\Phi(x, y)$ by a random point in the FB search space, we choose the spacially closest foreground and background samples.

The following process tries to find the best sample pair in the FB search space by iterating between two steps: *propagation* and *random search*. Propagation is the process to update $\Phi(x, y)$ of the unknown pixel $I(x, y)$ by finding the sample pairs $\Phi(x', y')$, which provide the smallest cost. (x', y') is in the first order neighborhood of (x, y) and includes (x, y). This step can improve the efficiency of random search, because generally neighboring pixels tend to have similar sample pairs. Random search will test a sequence of random trials, $(F^{i_k}, B^{j_k}|k = 0, 1, 2, ...,$ on the FB search space, and this sequence is defined as:

$$(i_k, j_k) = (i, j) + \omega\beta^k R_k, \tag{3.6}$$

where R_k is a uniform random number in $[-1, 1] \times [-1, 1]$, $\beta = 0.5$, and ω is the size of the search space. This search will go on until the search window radius $\omega\beta^k$ is below 1. $\Phi(x, y)$ will be set to the new pair (F^{i_k}, B^{j_k}) if it has a smaller cost.

3.4 Contour Matching

Before deforming the human prototype with the contour information, we need to match each anchor vertex of the human prototype to one point on the contour path. The anchor points of the template will then move to the correct locations based upon the participant's silhouette and the rest of the vertices will be deformed. In order to align these two point sets properly, we consider all the possible matching results to get the minimum cost. If we solve the problem in a straightforward manner, one mismatch may cause the following errors. In [48], a Hidden Markov Model (HMM) can be used to describe this problem and the existing dynamic programming approach can solve it by testing different matching possibilities.

3.4.1 Match Criteria

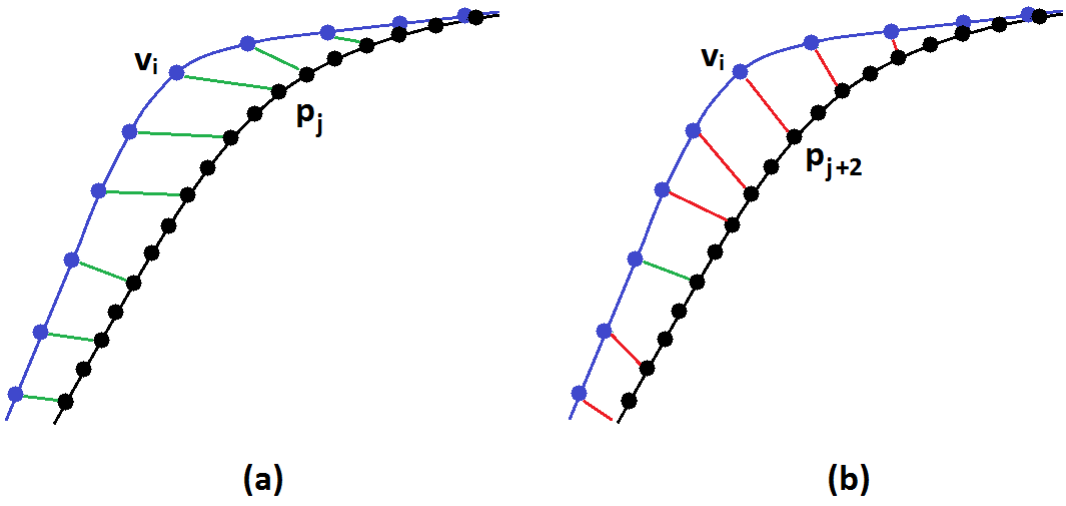

Figure 3.5: Human right shoulder contour matching case. Blue points are anchors of the 3D human template and black points are sampled contour points. (a) Correct contour matching results marked with green lines. (b) Incorrect matching results marked with red lines.

The Euclidean distance between one contour point and the corresponding anchor vertex is an important feature to determine the cost of one match. However, using these distance criteria alone is not good enough to provide an accurate result, especially around the contour part with sharp curves. Take a human shoulder for example (Figure 3.5), the anchor vertex v_i on the shoulder will be matched to the contour point p_{j+2}, since p_{j+2} is the closest point to v_i based upon Euclidean distance. However, p_{j+2} is located on the participant's arm instead of shoulder. The correct matching pair should be v_i and p_j, since it is the connection between the shoulder and the arm. Extra criteria are required to compute the cost.

Kraevoy et. al [48] considered two components to defining the cost metric of matching template model vertex v to contour point p. The first component is *proximity* $d_P = (p^x - v^x)^2 + (p^y - v^y)^2$, which measures position difference between v and p in the image plane, and the second is *normal difference* $d_N = n^p \dot{n}^v$, where n^p is the normal to the contour at p (lifted to 3D using z = 0) and n^v is the mesh normal at v. Since continuous contour points should map to continuous paths on the 3D model, they add one metric of *continiuty* $d_C = \|v_i - V_{i-1}\|^2 / \|p_i - P_{i-1}\|^2$, where v_i and v_{i-1} are the matching vertices of p_i and p_{i-1}.

3.4.2 HMM Model

The goal of HMM is to infer the corresponding hidden states that are most likely to have generated the input observations. The HMM requires *emission probabilities*, the possibility of a given hidden state producing the input observed states, and *transition probabilities*, the possibility from the previous hidden state to the current one. In [48], the contour matching problem was described as a HMM by setting the sorted contour points as observed states and anchor vertices as hidden states.

The emission probability is computed as:

$$P(p_j|v_i) \propto e^{-\frac{1}{2}(\frac{d_P}{\sigma_P})^2} e^{-\frac{1}{2}(\frac{d_N-1}{\sigma_N})^2}, \tag{3.7}$$

where $\sigma_P = 0.5$ and $\sigma_N = 1$. The transition probability is computed as:

$$P(v_i|v_{i-1}) \propto e^{-\frac{1}{2}(\frac{d_C-1}{\sigma_C})^2}, \tag{3.8}$$

where $\sigma_C = 0.05$.

This HMM problem is solved by the Viterbi algorithm, which is a dynamic programming approach to find the most likely sequence of hidden states, called trellis. The result may cause several contour points to match to one mesh vertex, so a post-processing pass guarantees a one to one match, determined by the emission probability, between the contour points and the anchor vertices.

3.5 3D Model Reshaping

3.5.1 Introduction

The 3D model reshaping algorithm in our system uses the Kraevoy's mean-value encoding and decoding approach [47], which is able to generate a 3D human model with skeletons. The approach is simple and robust, so the participant won't feel frustrated during the data collecting process. This approach is composed of two steps, encoding and decoding. The first step can be pre-computed, so we only need to create the model by decoding with the real world data and the pre-saved data from the first step.

In order to deform the template 3D model based upon real-world data, the relationship among

connected vertices of the discrete geometric model needs to be analyzed and this process is done by encoding. In our system, the 3D human prototype is encoded with the Mean-Value Geometry approach [47] and the encoding data is output to an XML file. This step is only needed once for each prototype model, which means that we can reuse the encoding data for any system using the same 3D human prototype, once we have previously generated it. We can then deform the prototype based upon the encoding data and the contour path, which is different from the prototype's contour.

First of all, the 3D template is loaded into the system and form a map to represent the connection between the vertices. We then need to decide the anchor vertices of the template, which will be matched to the real-world data directly and used as the control mechanism. This can be, for example, the vertices on the outline of the initial pose. After we get the adjacency relationship among vertices and the modified anchors, we are able to establish the new position in 3D space for the rest of the vertices with one encoding and decoding of discrete geometry models.

3.5.2 Mean-value Encoding One Vertex

Given a 3D human template with vertices V and edges E, the mean-value encoding for one vertex $v_i \in V$ is computed from the Euclidean coordinates of the vertex and its m neighbor vertices v_j, where $(i, j) \in E$. These neighbor vertices are enumerated counter-clockwise around v_i as v_{j_i},...,v_{j_m}.

For each vertex v_i we define a corresponding local projection plane, $P_i = n_x x + n_y y + n_z z + d_i$, using the normal, $n_i = (n_x, n_y, n_z)$. The normal n_i is computed as

$$n_i = \frac{\sum_{k=1}^{m}(v_{jk+1} - l) \times (v_{jk} - l)}{\| \sum_{k=1}^{m}(v_{jk+1} - l) \times (v_{jk} - l)\|} \tag{3.9}$$

where

$$l = \frac{1}{m} \sum_{(i,j) \in E} v_j. \tag{3.10}$$

We use the averaged normal to a local Laplacian mesh as the normal of the projection plane of the vertex v_i. This enables us to achieve much better results for decoding v_i in terms of stability, speed, and shape preservation. The average distance from origin d_i is computed as:

$$d_i = -\frac{1}{m} \sum_{(i,j) \in E} n_i \cdot v_j. \tag{3.11}$$

Given n_i and d_i, the shape encoding of the vertex coordinates is separated into a tangential component computed in the projection plane and a normal component based on the vertex offset from the plane. First, we project v_i and its neighbors v_j onto the projection plane P_i:

$$v_i{}' = v_i - (d_i + (v_i \cdot n_i))n_i \tag{3.12}$$

$$v_j{}' = v_j - (d_i + (v_j \cdot n_i))n_i \tag{3.13}$$

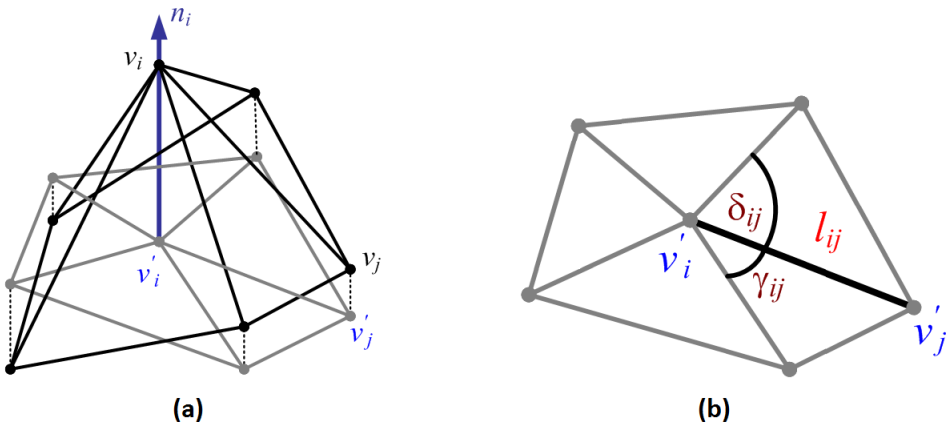

Figure 3.6: Mean-value encoding [47]: (a) The 3D mesh is shown in black, the normal n_i is shown as a vertical vector, the projected mesh in the local projection plane is shown in gray. (b) the values (δ_{ij}, γ_{ij}, l_{ij}) used to compute mean-value weight w_{ij}.

We then compute the mean-value weights of $v_i{}'$ with respect to $v_j{}'$:

$$w_{ij} = \frac{w_{ij}{}'}{\sum_{(i,k)\in E} w_{ij}{}'} \tag{3.14}$$

$$w_{ij}{}' = \frac{tan(\gamma_{ij}/2)+tan(\delta_{ij}/2)}{l_{ij}}$$

The angles γ_{ij}, δ_{ij}, and the lengths l_{ij} are shown in Figure 3.6.

To represent the normal component of v_i with respect to the local frame, we calculate and store the cotangent of the angle between each edge and the normal

$$b_{ij} = \frac{c_{ij}}{\sqrt{1 - {c_{ij}}^2}}, \tag{3.15}$$

where

$$c_{ij} = \frac{(v_i - v_j) \cdot n_i}{\|v_i - v_j\|} \tag{3.16}$$

The encoding of the entire model consists of the set of coefficients w_{ij} and b_{ij} defined for each half-edge (note that $w_{ij} \neq w_{ji}$ and $b_{ij} \neq b_{ji}$).

3.5.3 Mean-value Decoding One Vertex

In this section, we review how to use the encoding to explicitly obtain the 3D coordinates of one vertex from those of the adjacent vertices. The 3D positions of the neighbor vertices v_j and the encoding coefficients w_{ij} and b_{ij} uniquely define the position of the vertex v_i in the Euclidean space. We now sketch the numerical derivations leading to the explicit formulation for v_i (Equation 3.19). Using the mean-value weights w_ij, we obtain ${v_i}'$ (Equation 3.12) from ${v_j}'$

$${v_i}' = \sum_{(i,j)\in E} w_{ij}{v_j}' = \sum_{(i,k)\in E} w_{ij}(v_j - (d_i + (v_j \cdot n_i))n_i) \tag{3.17}$$

where n_i and d_i are calculated using Equation 3.9 and Equation 3.11 respectively. Since the mean-value weights sum, given ${v_i}'$ and using the coefficients b_{ij}, the new v_i is given by

$$v_i = {v_i}' + \sum_{(i,j)\in E} w_{ij}(\|{v_i}' - {v_j}'\|b_{ij} + (v_j - {v_j}') \cdot n_i)n_i. \tag{3.18}$$

We can rewrite this as a function of the rest of the vertices using the encoding coefficients w_{ij}, and b_{ij},

$$v_i = F_i(V) = \sum_{(i,j)\in E} w_{ij}(v_j + \|N_i \sum_{(i,k)\in E} w_{ik}(v_k - v_j)\|b_{ij}n_i) \tag{3.19}$$

where N_i is the 3×3 matrix (column notations)

$$N_i = I_3 - n_i {n_i}^T, \tag{3.20}$$

where I_3 is a 3×3 identity matrix. This formula uniquely defines v_i, for any set of neighbor vertices and any mean-value encoding.

3.5.4 Encoding and Decoding of Models

To uniquely encode 3D models, we must eliminate the degrees of freedom provided by rigid transformation and global scaling. Hence, in addition to the mean-value encoding of each vertex (w_{ij} and b_{ij}), the full encoding must contain the 3D Euclidean coordinates of the anchor vertices V_a. To decode a 3D model from the encoding above, we formulate and solve the following non-linear least squares minimization problem

$$argmin_{V'} G(V') = \frac{1}{2} \sum_{v_i \in V} (v_i - F_i(V))^2 \tag{3.21}$$

where $V' = V \; V_a$, all the non-anchor vertices. Note that while we only need to compute the coordinates of non-anchor vertices, the sum on the right-hand-side of the formula runs over all the vertices in the mesh. If the anchor vertex positions are unchanged, the set of original vertex positions is clearly a solution to the minimization problem.

This problem can be solved by using standard non-linear least-squares minimization techniques. Since the original model is available in our system, the original coordinates can always be used to provide the initial guess for the optimization. The solution requires 9 iterations to converge. In [47], Kraevoy and Sheffer need real-time editing interaction using this solution approach, so they incorporated a multi-resolution structure into the encoding and decoding procedures. This

approach makes the encoding slightly more time consuming, but dramatically speeds up the decoding. In our system, the encoding can be done once as a pre-processing step for one human template, while decoding is performed repeatedly for different participants in the scene. Therefore, the hierarchical approach is very suitable for our human modeling process.

The multi-resolution hierarchical approach removes the non-anchor vertices one by one, until only a base mesh connecting the anchors remains. When selecting the edge to be half-edge collapsed, we use a mixture of volume preservation (see Lindstrom and Turk [54]) and minimal angle maximization metrics in order to preserve the mesh shape and to avoid degenerate configurations throughout the hierarchy. Before each collapsed vertex is removed from the model, the mean-value encoding of the vertex in the current mesh is computed and stored for reconstruction purposes.

At the beginning of the decoding process, anchor vertices are placed at the specified location based upon the real-world data. The subsequent decoding procedure involves two major operations: vertex split and optimization. The former is reversing the simplification order; collapsed vertices are added to the mesh one at a time at the location computed with Equation 3.19. If the anchor positions are unchanged or a rigid transformation of the original position, this placement gives the exact desired position of the vertex in 3D. Otherwise, each split introduces some error; hence $G(V')$ (Equation 3.21) is not optimized. To get minimum $G(V')$, after performing a sequence of vertex splits, we use a Gauss-Newton minimization procedure combined with line-search. The minimum is obtained when the Jacobian of $G(V')$ is zero. The rows of the Jacobian are:

$$\frac{\partial G}{\partial v_i} = v_i - F_i(V) - \sum_{(i,j)\in E} (v_j - F_j(V)) \frac{\partial F_j(V)}{\partial v_i} \tag{3.22}$$

Defining the Jacobian of $G(V')$ as J, the vector of the functions F_i as F, and the matrix of partial

derivatives $\frac{\partial F_j(V)}{\partial v_i}$ as ΔF, the system can be rewritten in matrix format as

$$J(V') = (I' - \Delta F)^T (V' - F) = 0 \tag{3.23}$$

where I' is an $|V| \times |V'|$ sparse matrix with 1's on the diagonal. Using Gauss-Newton method we ignore the second order terms in the Hessian, defining

$$H = (I' - \Delta F)^T (I' - \Delta F) \tag{3.24}$$

Thus at each iteration of the procedure, we solve the linear system

$$H\delta = -J(V') \tag{3.25}$$

and update $V' = V' + \alpha\delta (0 \leq \alpha \leq 1)$. We use standard bisection line-search to compute α. We perform numerical derivation to compute the matrix of partial derivatives ΔF for the three coordinates x, y, and z. We use the conjugate gradient method to solve the linear system (Equation 3.25). From Kraevoy and Sheffer's experience [47], it is sufficient to perform optimization only once for 3% of the vertices and execute 7 iterations for each optimization. For extreme deformations, after the model is fully reconstructed, they applied several Gauss-Seidel iterations, typically four, toward solving Equation 3.23. The Gauss-Seidel procedure uses Equation 3.22 (equating it to zero) to set the value for v_i. This optimization step will remove the clearly visible error of intermediate mesh introduced by performing edge-splits.

CHAPTER 4: ISTRAW

4.1 Introduction

IStraw is a corner finder based upon the ShortStraw algorithm [93]. This algorithm was first introduced by Xiong and LaViola in [95], and later improved in [96]. Corner finding algorithms are used in sketch-based systems as the initial step for gesture recognition.

In our human modeling system, the contour of the participant is segmented before the anchor matching process to improve the matching accuracy. Corners, like armpits, on the human contour are important references for segmentation. We choose IStraw for the corner finding process for two reasons. The first is that IStraw works better for strokes that have noisy data and the human contour is such a noisy object as it is not smooth. The second reason is that IStraw provides the best all-or-nothing accuracy, which means that the result has minimum positive and negative false results. Missing any corner or finding any unnecessary corner will cause false segmentation, so the matching result will be totally wrong and the generated model won't look like the real participant at all. To fit IStraw into our system, we make some changes based upon features of the human contour. The details of these changes are presented in Section [**?**].

In this chapter, We review the ShortStraw algorithm and its limitations, discuss the details of IStraw's improvement based upon ShortStraw, as well as the evaluation results of IStraw and other corner finding algorithms.

4.2 ShortStraw Review

ShortStraw is an accurate polyline corner finder that is easy to understand and implement [93]. After resampling the input data, ShortStraw finds corners using both a bottom-up and top-down approach. In this system, users can draw polylines free-form while achieving a high *total corners* and *all-or-nothing* accuracy rate. Furthermore, the algorithm can be quickly integrated into sketch-based interfaces. However, there is still room to improve its accuracy and to extend the technique to deal with polyline ink strokes containing arcs and curves. In this section, we will discuss the implementation of ShortStraw and its shortcomings.

4.2.1 ShortStraw Implementation

The first pass of ShortStraw involves resampling the input data, an important component for achieving high corner finding accuracy using Wolin et al.'s approach. The resampling algorithm used by ShortStraw is based upon [97], but uses a different interspacing distance between points. The interspacing distance is defined by the diagonal distance of the stroke's bounding box divided by 40.

ShortStraw then finds corners with two steps, one bottom-up and the other top-down. First, ShortStraw defines the concept of "straws" from primitive information. A straw for a point at resampled point p_i is computed as:

$$straw_i = ||p_{i-W}, p_{i+W}|| \tag{4.1}$$

where W is a constant window equal to 3 and $||p_{i-W}, p_{i+W}||$ is the Euclidean distance between the resampled points p_{i-W} and p_{i+W}. The shorter the straw, the more likely the point will be a corner.

The initial corner set is taken from the resampled stroke points whose straw lengths are a local minimum below a threshold t , defined by the median of the computed straw list.

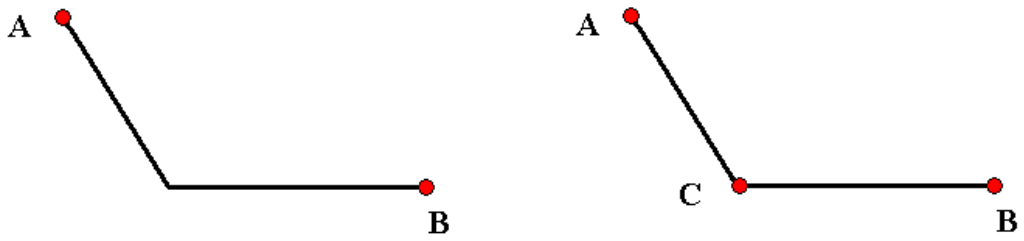

Figure 4.1: An example of a collinear test between two adjacent corners A and B: the initial corners (left) and the corners after the collinear test (right).

After the bottom-up approach, some higher-level processing is used to find missed corners and remove false positives. ShortStraw checks to see whether two adjacent corners pass a collinear test. Take Figure 4.1 for example, corners A and B are not on a line, so there must be additional corners between them. The point with the minimum straw value, C in the figure, will be added to the possible corner set. Then the next collinear test will be between points A and C. The process is repeated until all of the stroke segments between pairs of consecutive corners are lines. Another collinear check is then run on subsets of triplets, consecutive corners like A, B, and C in Figure 4.2. If the two corners A and C are collinear, then B, the possible corner between A and C, is not a real corner and should be removed from the corner set.

Figure 4.2: An example of a triplet collinear test of corner B between three consecutive corners A, B, and C: the initial corners (left) and the corners after the triplet collinear test (right).

4.2.2 ShortStraw Limitations

Although ShortStraw achieves outstanding accuracy compared to other previous corner finding algorithms, there are still some issues left unaddressed by Wolin et al. [93]. The distortion between the resampled stroke and the original stroke causes some real corners to be missed. During the bottom-up approach of ShortStraw, the first three and last three resampled points do not have straw values, given that the window size $W = 3$ is constant. In addition, timing information can be useful for corner finding, since users prefer to slow down on the corner, but ShortStraw does not take advantage of the speed change.

In the top-down step, the triplet collinear check will be unreliable if some corners are missed between these points and may lead to a false deletion of a correct corner. Another issue with the ShortStraw approach is the way in which the threshold is set for a collinear check. The constant threshold used by ShortStraw is not robust in the presence of some shapes. In addition, noise caused by resampling or hooks presents an issue, and sometimes the corner found is not the resampled point closest to the real corner of the input stroke.

Finally, ShortStraw only works well for polyline ink strokes, but not for ink strokes with curves and arcs. Since complex free-hand shapes are needed in most sketch-based interfaces, effective corner finding must involve the removal of false positive corners resulting from curved parts of the sketch.

4.3 IStraw Algorithm

IStraw is a new corner finding approach to improve the accuracy and extend the scope of ShortStraw. We analyzed the issue listed above and developed techniques to address each deficiency.

4.3.1 Straws

The original ShortStraw algorithm uses a window size $W = 3$, resulting in the straws of the first three and last three resampled points to not be computed, but remain a default number, 0. Thus, these resampled points might be selected as corners during a post processing step. We can set values for the straws for these points. Given the indices of the resampled points p_i where i goes from 0 to $N - 1$, the computations are shown as:

$$straws_1 = ||p_0, p_{1+W}|| \times \frac{2W}{(W+1)}$$

$$straws_2 = ||p_0, p_{2+W}|| \times \frac{2W}{(W+2)}$$

$$straws_{N-2} = ||p_{N-1}, p_{N-2-W}|| \times \frac{2W}{(W+1)}$$

$$straws_{N-3} = ||p_{N-1}, p_{N-3-W}|| \times \frac{2W}{(W+2)}$$

where W is the window size and $||p_i, p_j||$ is the Euclidean distance between the resampled points p_i and p_j. Since the definition of $straws_i$ where i goes from W to $N-W-1$ is the Euclidean distance between p_{i-W} and p_{i+W}, there are $2W$ interspacing distances between these two points. We use the Euclidean distance between p_0 and p_{1+W} to compute $straw_1$, but there are only $W+1$ interspacing distances, so we multiply $||p_0, p_{1+W}||$ by 2W and divide by $W + 1$ as the straw associated with p_1. The same approach is applied to the other three straws. Note that having the straws for the start point p_0 and end point p_{N-1} be zero is acceptable, since these two points will always be chosen as corners.

4.3.2 Timing Information

By using timing information we can obtain missing corner candidates due to the observation that users are more likely to slow down while coming to a corner [74]. When resampling the ink stroke, we define the time for each resampled point as the difference between the time stamp of the raw point just prior to the current resampled point, and the time stamp of the raw point just prior to the previous resampled point. Then during the bottom-up step, we look for the maximum time t_{max} between two adjacent corners. If t_{max} is larger than the the threshold $2 \times meanTime$, then that point will be added to the corner list. Note that if this condition holds true, we also change the threshold used during the second pass of the triplet collinear test, explained further in Section 4.3.3.

4.3.3 Dynamic Threshold for the Collinear Test

The collinear test of two points p_a and p_b checks whether the ratio of the Euclidean distance and the path distance between the two points is below a threshold. The equation for the collinear ratio is:

$$r = \frac{||p_a, p_b||}{\sum_{i=a}^{b-1} ||p_i, p_{i+1}||} \tag{4.2}$$

where $0.0 \leq r \leq 1.0$, since the path distance is always greater than the chord distance. We use the ratio of distances in the collinear test, so the interspacing distance of the resampled points will not affect its decision.

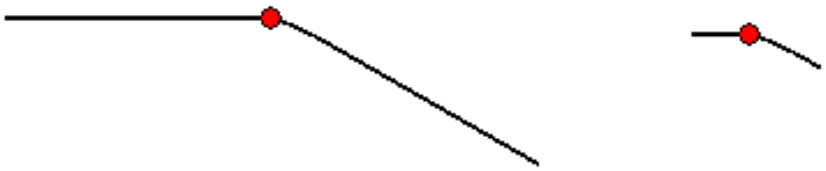

Figure 4.3: The length of a segment will affect the corner decision: a longer segment (left) and a shorter segment (right).

During our initial exploration of ShortStraw, we found, for example, that the point on the left stroke in Figure 4.3 is more likely to be a corner than the point on the right stroke in Figure 4.3, even if the angles between the two line segments for both strokes are equal. Thus, the threshold for the collinear test should change based on line segment length. Another factor in changing the threshold is timing information, since the candidate corner with slower speed is more likely to be a real corner.

During the second collinear pass on any three consecutive corners, we set the threshold based on the length of the segment and timing. Based on empirical observations, if the difference between the first and third corner indices is larger than ten, we increase the threshold by 0.0053. In addition, if the timestamp of this point or its adjacent points is larger than $2 \times meanTime$, then we increase the threshold by 0.0066. For more detail on these thresholds, see Section 4.3.9.

4.3.4 Consecutive False Corners Avoidance

Consecutive false corners is a special case defined as missing a correct corner, caused by failing to detect a corner or falsely removing one, bringing about the false deletion of subsequent corners. This phenomenon occurs because the missing corner decreases the reliability of the triplet collinear

test in the top-down component of ShortStraw.

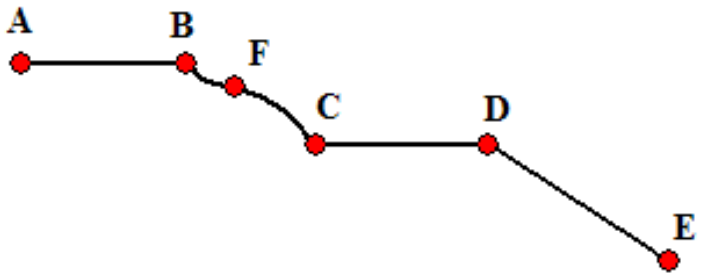

Figure 4.4: An example of consecutive false corner deletion.

Consider Figure 4.4. Points A to E are all the correct corners and point F is a false candidate corner. Since F is close to B, the Euclidean distance and the path distance from A to F have so little difference that B is defined as a wrong corner. The existence of F can lead to the false deletion of point B, then the triplet collinear check of point F will be between A and C instead of B and C. In this case, the system will leave F as a correct corner and go on to the next corner candidate C. Without deleting F, corner C will face the same problem as B and be identified as an unwanted candidate corner.

To avoid this situation, it is necessary to delete F before the triplet collinear check of point B. We solve this problem by passing all the candidate corners through the triplet collinear pass twice. The first pass has a higher threshold, and then we relax the threshold for the second. This dual pass approach will remove a false corner whose collinear ratio is too large (e.g., removing F in Figure 4.4).

4.3.5 Adjusting Corners

Sometimes a smaller straw value does not necessarily mean a resampled point is closer to the corner. From Figure 4.5, we can see that the straw value $|AB|$ of point C_i is larger than $|DE|$, the straw of point C_i', so C_i' will be chosen as the corner instead of C_i, which is closer to the real corner. This will not affect the result for the polyline corner finder using straws and collinear tests. However, our curve detection approach, using angle information (discussed in Section 4.3.7), will often define the point C_i' as a incorrect corner and delete it.

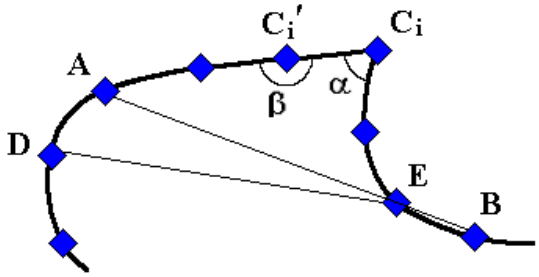

Figure 4.5: An example that the point closer to the real corner has larger a straw value than its adjacent point.

To make sure a corner is the resampled point closest to the real corner, we need to adjust each possible corner to move it to the right point. Normally, the adjustment is made based on whether it is before or after the corner initially found. We use the angle value of the three adjacent points to make the decision, since the point closer to the real corner will have a smaller angle between itself and its two adjacent points. As in Figure 4.5, α is smaller than β, so we should change the corner to point C_i.

4.3.6 Sharp Noise Avoidance

Sharp noise manifests itself in two situations. The first situation occurs in the start or end of the stroke (e.g. Figure 4.6 (left)). As we take the beginning and the end resampled points as corners, the hooks in a stroke normally will cause unwanted corners close to these points. The second situation exists with corners with sharp angles (e.g. Figure 4.6 (right)). This case is induced by the distortion of the stroke after resampling, which might change the shape from one sharp angle to two angles (e.g. Figure 4.7). Both situations can result in incorrect corners.

Figure 4.6: Two examples of sharp noise: caused by a hook (left) and caused by a sharp angle (right).

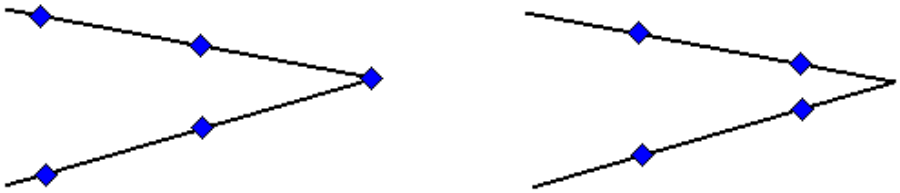

Figure 4.7: A properly resampled sharp angle(left) and an improperly resampled sharp angle (right).

Often, two close resampled points, ones where the difference between their indices in the point array is one or two, are both treated as corners. However, it is impossible for a user to draw a stroke with two corners so close together. Therefore, we can take one of the two resampled points

as the correct corner to avoid sharp noise. In our system, the first or the last point is left as a corner to get rid of hooks, and we choose the one of two adjacent points that has the smaller straw value to handle sharp angles.

4.3.7 Curve Detection

Thus far, we have focused on strategies for improving the ShortStraw algorithm that works well for polyline-based ink strokes. However, these methods do not work well when strokes contain curves and arcs, finding many unnecessary corners on the curve. Therefore, we need an approach to decrease the false positives caused by the curves and arcs.

4.3.7.0.1 General Approach

To remove unwanted corners, it is necessary to be aware of the difference between a real corner and a incorrect one. Ideally, a candidate corner C_i is the vertex of an angle defined by two rays generated from C_i and a resampled point on each side of the vertex starting at the right place. Assuming a correct $shift$ value and a real corner C_i, this angle will not significantly increase by choosing rays using other resampled points closer to the vertex. However, if C_i is on a curve, this angle will get larger. This approach requires finding all possible angles from the resampled point data.

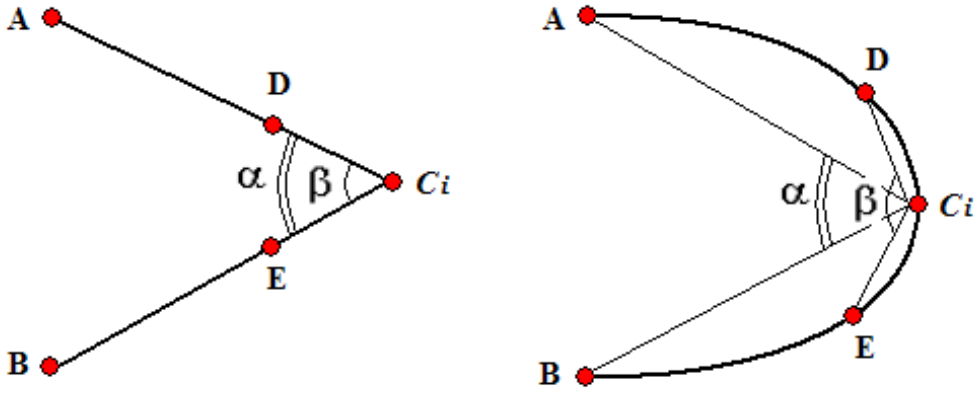

Figure 4.8: The difference between the corner and the curve: the angle does not change with a real corner as the vertex (left) and the angle will increase with a false corner on a curve (right).

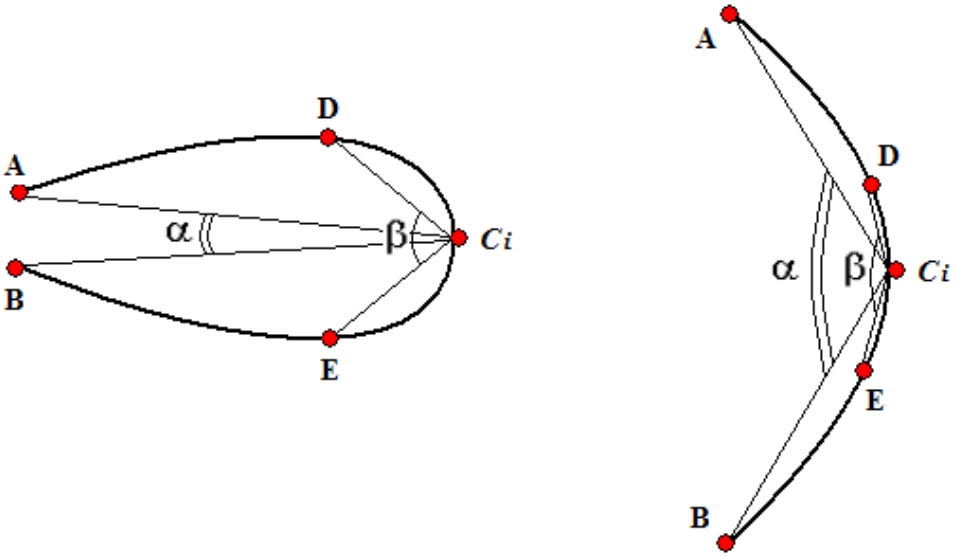

Figure 4.9: The difference between α and β based upon the value of α: α is small (left) and α is large (right).

Instead of comparing all the possible angles, we can pick two representative angles for comparison to enhance efficiency. As in Figure 4.8, the farther angle α is formed by C_i with the two resampled

points A and B, whose indices are equal to the index of C_i plus/minus a shift value. The two points, D and E, for the closer angle β, hav indices equal to the index of C_i plus/minus the shift value divided by 3. If β is below the threshold t_a, then C_i is a correct corner, otherwise it is a point on the curve. In our approach, t_a is set dynamically based on the value of α.

As with handling polyline strokes, falsely deleting a correct corner may cause sequential problems, and we use the approach discussed in Section 4.3.4 by having all possible corners go through the curve detection pipeline twice to avoid consecutive false corner deletion. During the first pass, we set t_a to be $36 + 0.85 \times \alpha$ and $26.1 + 0.93 \times \alpha$ for the second pass. From Figure 4.9, we can see the angle β will be larger if α increases. The details for choosing the thresholds are given in Section 4.3.9.

4.3.7.0.2 Shift Value

The term, $shift$, is defined as the array index difference between corner C_i and point A, and C_i and B. Setting the $shift$ value is crucial to the reliability of this approach. If $shift$ is too small, it is hard to tell the difference between α and β. On the other hand, a large $shift$ value may also cause problems. The left image of Figure 4.10 shows one possible case where a real corner will be deleted since the β is much larger than α with the wrong $shift$ value. Another example is that an incorrect corner on the curve will result in a poorly chosen $shift$ value, as shown in the left image of Figure 4.11. To make the correct decisions, we need to move points, A, B, D and E, closer to the corner C_i.

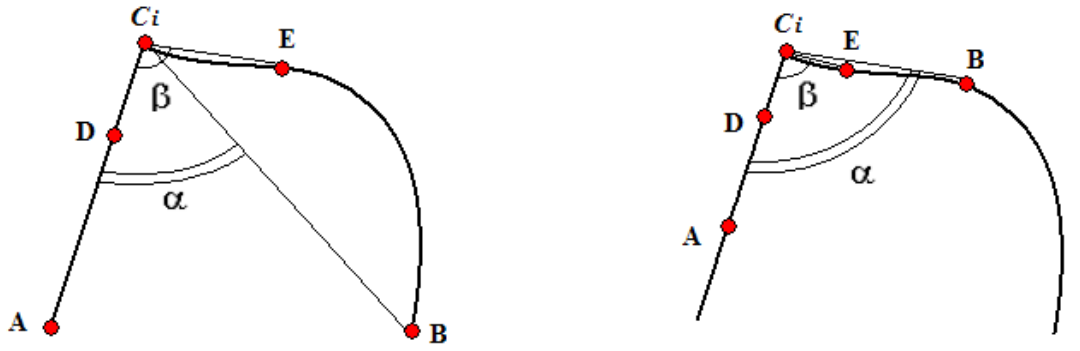

Figure 4.10: Example 1 of unwise shift value: $\beta - \alpha$ is large even though C_i is a correct corner (left) and a smaller shift value which moves point B closer to C_i is needed to make the right decision (right).

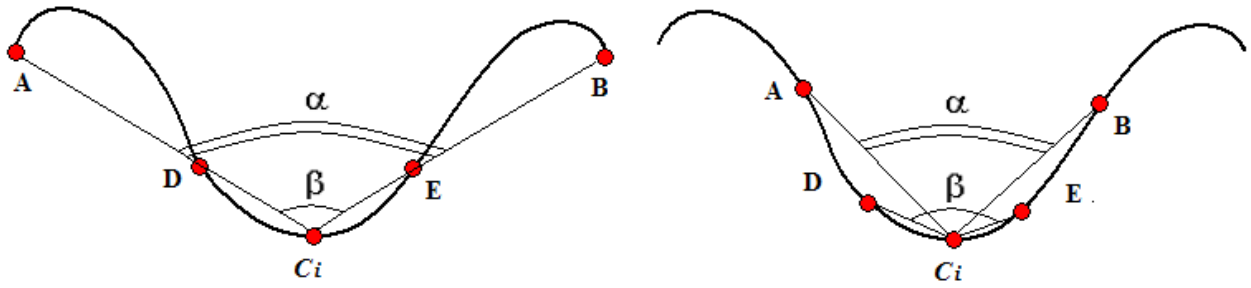

Figure 4.11: Example 2 of unwise shift value: $\beta - \alpha$ is small even though C_i is a incorrect corner (left) and a smaller shift value which moves point B closer to C_i is needed to make the right decision (right).

If the candidate is a real corner, its best $shift$ value should enable α to be the local maximum angle of this candidate (see Figure 4.10). On the other hand, if the candidate is on a curve, its best $shift$

value should enable α to be the local minimum angle (see Figure 4.11). Performing an exhaustive search for these local maxima and minima will sacrifice the simplicity of the algorithm. Thus, we make an approximation by choosing $shift = 12$, determined from a training dataset with the approach described in Section 4.3.9. However, if the previous corner C_{i-1} is too close to C_i, we change $shift$ to the difference between the indices of these two corners and do the same for the corner C_{i+1}.

4.3.7.0.3 Special Cases

On some curves, adjacent possible corners might be too close to tell the difference between the angles α and β. During the first pass of curve detection, we test only the angle of the candidate corner and its adjacent points, if the difference between the indices of this candidate and one of the adjacent corners is less than three.

Another case that will cause problems is the incorrect corner on an S shape curve. As in Figure 4.12, α and β are almost the same for the corner C_i, an incorrect corner. To determine whether it is an S shaped curve, we first need to make sure that the two stroke segments C_iA and C_iB are curves using a collinear test. Second, we test the difference between the direction change from $\overrightarrow{C_iA}$ to $\overrightarrow{C_iD}$ and the change from $\overrightarrow{C_iE}$ to $\overrightarrow{C_iB}$. For example, there is an S shape in the left image of Figure 4.12, so $\overrightarrow{C_iA}$ to $\overrightarrow{C_iD}$ is counterclockwise and $\overrightarrow{C_iE}$ to $\overrightarrow{C_iB}$ is clockwise. On the other hand, if C_i is not on an S shape, then the direction change will be the same, as shown in the right image of Figure 4.12. In this case, $\overrightarrow{C_iA}$ to $\overrightarrow{C_iD}$ and $\overrightarrow{C_iE}$ to $\overrightarrow{C_iB}$ are both clockwise. If the curve is an S shape, the candidate corner will more likely be a incorrect corner, so we check whether the angle defined by the candidate corner and its adjacent points is larger than 135 degrees. If so, it is considered to be a incorrect corner. We made the choice of 135 degrees after examining all the incorrect corners meeting the S shape requirement from our second training dataset (strokes with

curves) and taking the minimum angle of these points as the threshold. If C_i is not on a S shape curve, we use the general curve detection approach to test it.

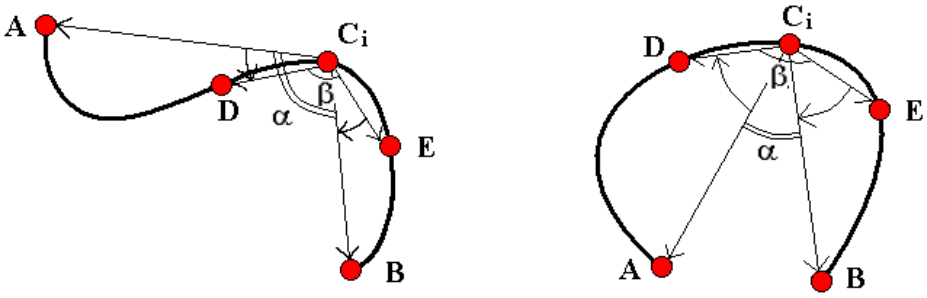

Figure 4.12: An example of S shape (left) and a normal curve (right).

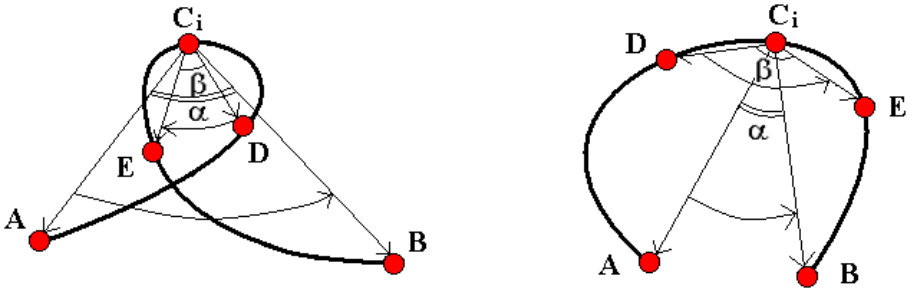

Figure 4.13: An example of a stroke that has a self-intersection (left) and a normal curve (right).

Strokes with self-intersections, as shown in the left image of Figure 4.13, also need special attention. Although C_i is a incorrect corner, the angle α is larger than β, since points D and E are inside the loop but A and B are outside the loop. To check for the existence of a self-intersection in a stroke, we can test for direction change. If the rotation direction from $\overrightarrow{C_iA}$ to $\overrightarrow{C_iB}$ and from $\overrightarrow{C_iD}$ to $\overrightarrow{C_iE}$ is opposite then a self-intersection exists. If they have the same direction then no self-intersection exists as shown in the right image of Figure 4.13. Once there is a self-intersection, we reduce the $shift$ value to 4 instead of 12 in the second pass of curve detection. If the self-intersection still exists, we keep this corner, otherwise we use the general approach to test it.

4.3.8 Shifting Resampled Points

Resampling the input stroke is necessary to find the corners in our system, but it will distort the original stroke, especially at some corners. The worst case is when the real corner is located between two resampled points, like in Figure 4.14, making the corner look like a curve. One way to alleviate this problem is shifting the resampled points to move one of them closer to the corner.

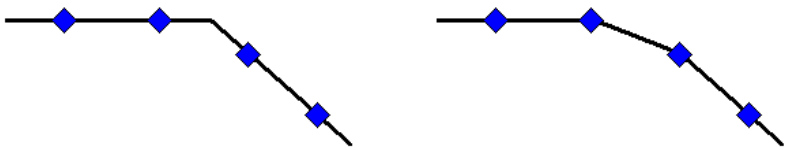

Figure 4.14: An example of distortion with a resampled stroke: the initial stroke (left) and the resampled stroke (right).

In our corner finder, we resample the input stroke twice: the first time setting the first raw point as the first resampled point, and the second time shifting all the resampled points backward half the interspacing distance. Next we find all the corners in these two resampled strokes and merge the two corner sets. The distance between a resampled point to a real corner is, at worst, one fourth the interspacing length. Based on the sampling rate from our data, we found that two time shifts were sufficient for our approach. However, more shifts might be needed with lower sampling rates, a possibility that we plan to investigate as part of our future work.

With this approach, we will see an increase in false positives during the bottom-up part of the algorithm (i.e., initial corner set calculation using straws). However, the top-down component (i.e., the collinear test and curve detection) is not sensitive to the resampling rate, since the collinear test utilizes ratios rather than Euclidean distances and curve detection examines angles relative to each other.

4.3.9 Choosing Thresholds

Our corner finder depends heavily on the parameters used in the algorithm. Optimizing these parameters is essential to the accuracy of the system. Different parts of the algorithm have different thresholding requirements, and we describe how to find and set values for the most important parameters in this section.

The parameters used in interspacing between resampled points, the window size for computing straws, and the threshold for finding initial corners come from the original ShortStraw paper [93], and they work well in our algorithm. Thus, we chose to utilize these values in our work.

There are several threshold values used in the collinear testing procedures to find missing corners and remove false positives. The general strategy for computing these thresholds is to find the appropriate collinear ratio r using equation 4.2 on the training dataset. For the collinear test used to find missing corners, we find the maximum r for all of the missing corners in the training dataset. As in Figure 4.1, in order to add the missing corner C, we must make sure the stroke segment from the previous corner A to the next corner B is not a line, which means the ratio must be smaller than the chosen threshold. Based on our training data, the ratio of 0.975 was set as the threshold to ensure all these missing corners are added to the corner list.

The first pass of the triplet collinear test is used to remove false corners that specifically stem from the consecutive false corner problem (e.g., F in Figure 4.4). To find the appropriate threshold for this test, we find all of the consecutive false negatives in the polyline training data using the ShortStraw algorithm with 0.975 as the threshold for the first collinear check. We then compute the collinear ratio r for each stroke segment between the adjacent missing corners and take the minimum as the threshold. Using our training data, we set the threshold equal to 0.988.

The second pass of the triplet collinear test is used to try to minimize the number of false positives

without adding any false negatives using the polyline training data. This condition is important because we focus on only false positives in the curve detection component of IStraw. To find the appropriate threshold for this test, we

1. find all the false negatives in the strokes using the IStraw algorithm with a triplet collinear test threshold of 0.95 (taken from the original ShortStraw algorithm) in pass 2 and without curve detection
2. compute the collinear ratio r for each stroke segment starting from the corner before the false negative and ending at the corner after the false negative
3. separate corners into four groups:
   (a) corners far from their adjacent corners (index difference larger than ten) and drawn slowly (time of the corner point is larger than $2 \times meanTime$)
   (b) corners that are far from their adjacent corners but are drawn fast
   (c) corners that are drawn slowly but not far from their adjacent corners
   (d) the remaining corners
4. find the maximum ratio for each group and keep the remaining corner's ratio as the basis threshold. The remaining thresholds are used when corners fall into the first three groups, defined by stroke speed and corner index differences.

For our implementation and training data, the ratios for each group are 0.9826, 0.98, 0.9813, and 0.9747 respectively.

For the curve detection part of our algorithm, we need to choose good "shift" values for each of the two passes. Recall that a $shift$ value is an integer representing the index difference between a

corner and a resampled point. This value is used to find the resampled points to the left and right of a candidate corner, which points are then used to compute α. In our case, the best $shift$ value for each candidate was computed by searching for the minimum α value if the candidate is a false corner and the maximum α value if it is a real corner. Based on our training dataset, we found that the mean of the observed best $shift$ values was 12, and thus have chosen this as the $shift$ we use in computing α.

The best β, as defined in Section 4.3.7.0.1, is the local maximum angle defined about a false corner and the local minimum angle for a real corner. To find β we need to know what the relationship is between the shift value used in calculating α and the index difference of β to the candidate. We define the "shift" value used in calculating α as the variable X and the index difference for calculating β as variable Y. Since the relationship between X and Y is $Y = kX$, we can use least squares to find k. In our case, $k = 1/3$ and the indices to get β are the testing corner index plus/minus the shift value used in calculating α divided by three.

At this point, we have initial shift values for calculating α and β. However, as discussed in Section 4.3.7.0.1, β will be greater than α for a false corner on a curve if the $shift$ value is chosen wisely and the candidate is none of the special cases. Unfortunately, a real corner whose β is also larger than α is more likely to be incorrectly deleted. To get a reasonable threshold for β in the first pass of the curve detection process, we need to analyze these real corners to delete as many false positives as possible without creating any false negatives. We can find these correct corners, as shown in the left image of Figure 4.15, using the constraint $\beta > \alpha$. After obtaining the set (α, β), points from all the training data with or without curves, our goal is to find the upper boundary $\beta = a + b\alpha$ of all the real corners to guarantee there will be no false negative after our first curve detection pass. We use the least squares method to get a line $\beta = a' + b'\alpha$ that best fits the data. We then vertically shift this line upward to the point farthest away from the line to get the boundary we need. For the training data we used, we found $\beta = 36 + 0.85\alpha$ to be the threshold needed to

ensure all the correct corners are left after the first pass.

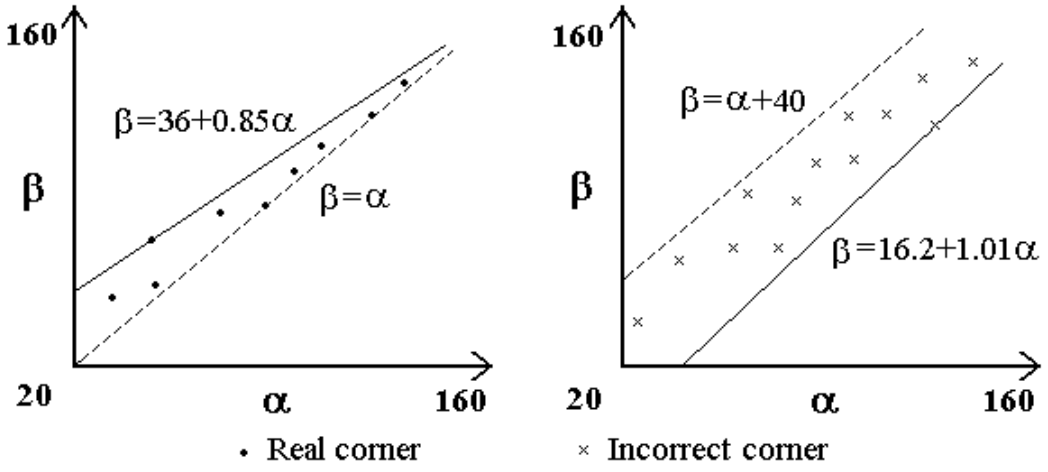

Figure 4.15: Our method for choosing the upper boundary of all the real corners (left) and the lower boundary of all the incorrect corners (right).

In the second curve detection pass, we are interested in minimizing both false positives and false negatives to obtain the highest all-or-nothing accuracy possible. Some of the candidates on the curve have such a subtle increase from α to β that they may be left as a real corner, so we need to find a lower boundary for all these false candidate corners in order to eliminate them. To do so, we employ the same approach as with the first curve detection pass. Since all the real corners are below the line $\beta = 36 + 0.85\alpha$, the incorrect corners under this line may be left as correct corners. In this case, we enlarge the boundary from $\beta < 36 + 0.85\alpha$ to $\beta < 40 + \alpha$ to find all the false positives from our training set that may be decided as real corners during this pass (see right image of Figure 4.15). Again, based on the training data, we obtain a second line to fit the data of all these incorrect corners and shift it down slightly to $\beta = 16.2 + 1.01\alpha$, to provide a lower boundary of all the observed false corners. This lower boundary can guarantee no false positives. Finally, we got $\beta = 26.1 + 0.93\alpha$, a line having the mean slope of the two boundaries and going through the intersection of them, as the threshold needs to minimize both false positives and negatives. These

lines are graphically depicted in Figure 4.16. The real corners above the line, $\beta = 16.2 + 1.01\alpha$, will be false negatives and the incorrect corners below this line will be false positives.

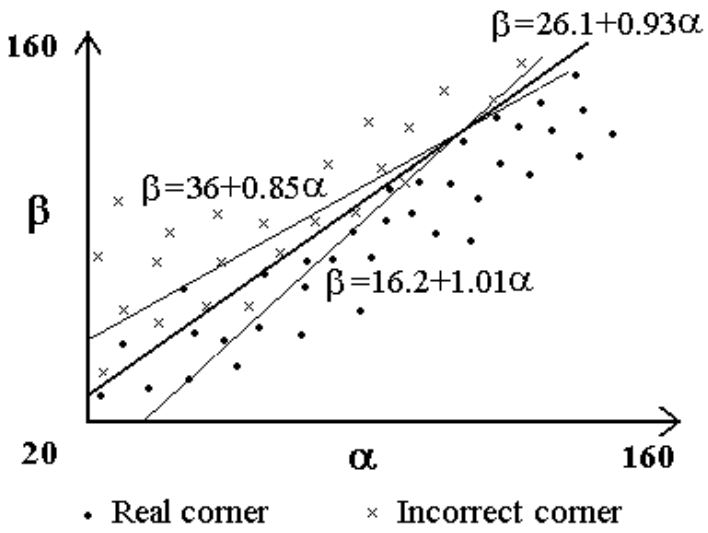

Figure 4.16: Our method for choosing the threshold in the second pass.

4.4 IStraw Evaluation

To evaluate IStraw, we conducted several experiments comparing its corner finding accuracy to the original ShortStraw algorithm as well as to MergeCF and Sezgin's scale space algorithm. Note that both MergeCF and Sezgin's algorithm directly support corner finding in strokes with arcs and curves. In addition we analyzed IStraw's computational complexity to determine if its running time was on par with ShortStraw.

4.4.1 Evaluation Tests

As in [93], we use two different measures to determine the accuracy of IStraw. The first one, ”Correct Corners Accuracy”, described in [74], is equal to the number of correct corners found divided by the total number of correct corners a human would perceive. The second one, ”All-or-Nothing Accuracy”, defined in [93], takes false corners into account, which means a correct stroke should have no false positives or negatives. This accuracy metric is calculated by dividing the number of correctly segmented strokes by the total number of strokes.

We used the test data, 244 polyline strokes in [93] to configure the polyline ink stroke part of our algorithm. This set of data, consisting of the 11 shapes shown in Figure 4.17, were drawn by six users. In addition, we used data gathered from six students, 120 stokes in total, using the shapes shown in Figure 4.18 to configure the curve detection component of our algorithm.

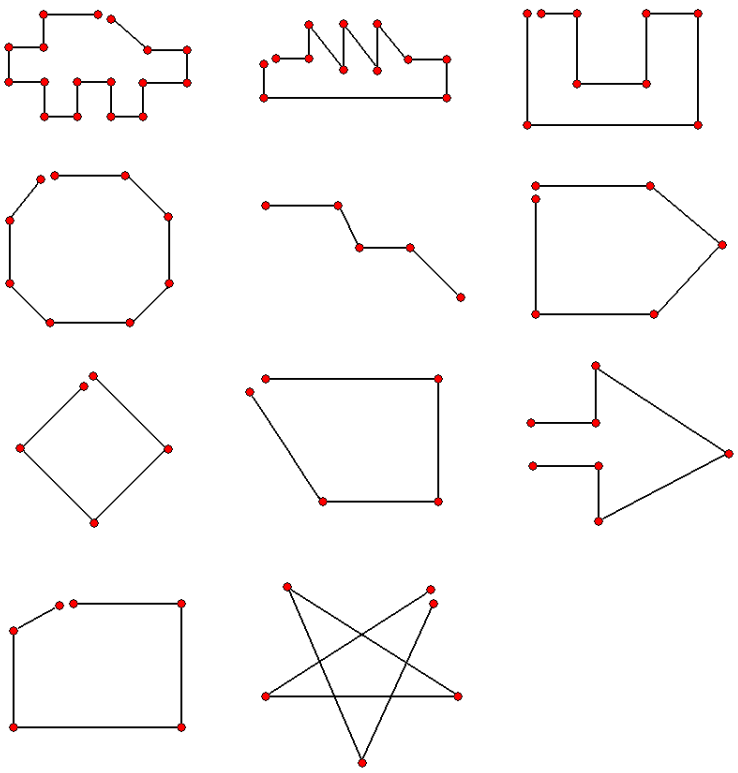

Figure 4.17: The 11 polyline shapes used for corner finding testing from the original ShortStraw dataset. There are 87 corners in total, including the start and end points, which are marked with red points.

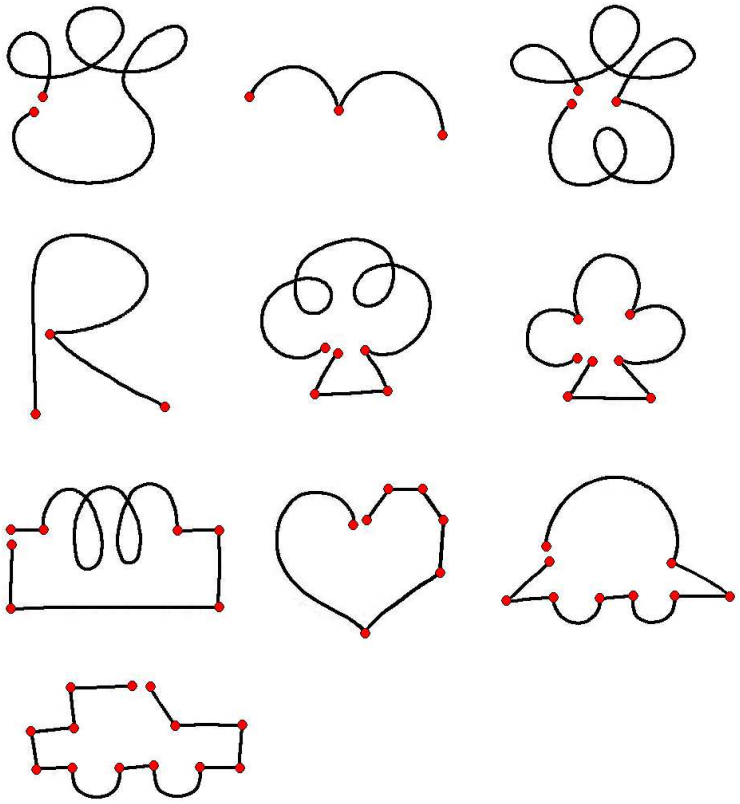

Figure 4.18: 10 new shapes with curves used for corner finding testing. There are 59 corners in total, including the start and end points, which are marked with red points. This data was used for both training and testing.

To test IStraw, we collected two separate datasets using a Compaq TC4400 tablet computer with a 1.83 GHz Intel Core2 processor and 2 GB of memory. The first dataset used strokes from 15 users (6 females and 9 males) from the computer science, electrical engineering and mechanical engineering fields. Nine out of the fifteen users had tablet PC experience. Users wrote samples for 21 shapes used for testing including the 11 found in Figure 4.17 and the 10 in Figure 4.18. After getting familiar with the system, each user was asked to draw each shape four times. 1260 strokes were collected but 13, examples of which are shown in Figure 4.19, were removed because they

were not properly drawn like the required shapes in Figures 4.17 and 4.18. Thus, our first test set contained 1247 strokes, 652 from polyline ink strokes and 595 from curve ink strokes.

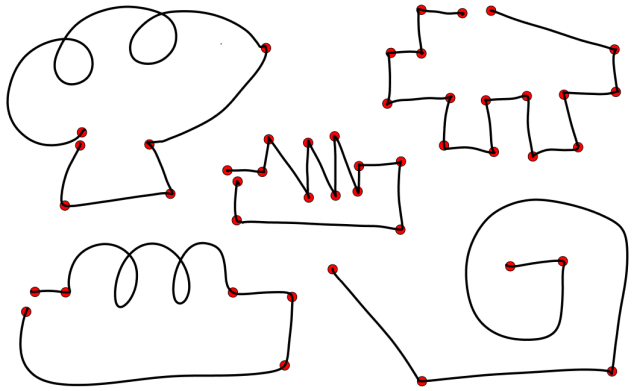

Figure 4.19: Some examples of removed strokes, but we can see they are all correctly segmented with IStraw.

Our second dataset was gathered to have strokes across a wider range of users and shapes. This dataset had strokes from 10 users (5 female and 5 male) using the 10 additional shapes shown in Figure 4.20. Note that only two of the ten users had participated in the previous data collection task. The testing process was the same as the previous one. Each shape was drawn 4 times by each user, so 400 strokes were collected. Since 5 strokes were removed (Figure 4.19), the second data set contained 395 strokes in total. None of these strokes were used in the training process for our corner finding algorithm.

In addition to ShortStraw and IStraw, we tested two other algorithm variations. The first one is ShortStraw+C (ShortStraw combined with our curve detection approach) and the second is IStraw-C (our algorithm without curve detection). To make a thorough comparison, we also tested two other, state-of-the-art corner finding algorithms: MergeCF [94] and Sezgin's scale space algorithm [75], both of which are able to handle strokes with curves.

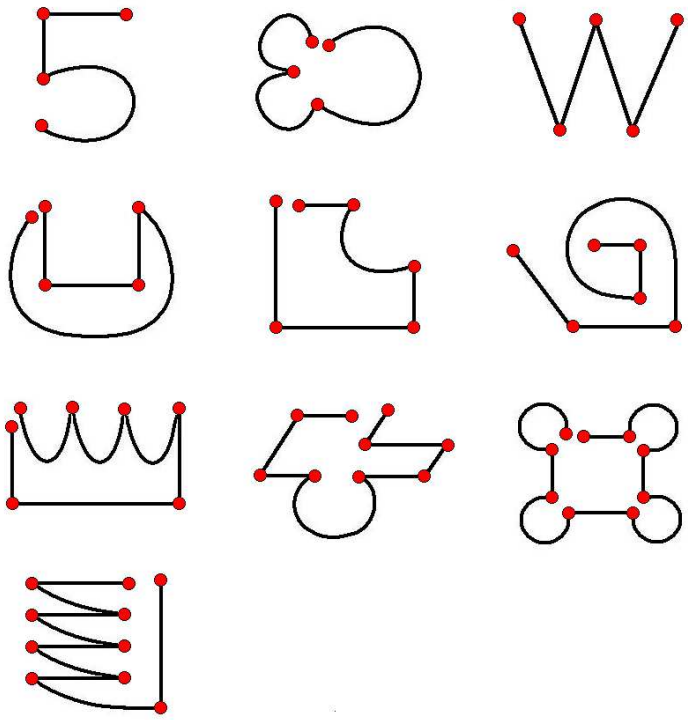

Figure 4.20: 10 additional new shapes used for corner finding testing. There are 65 corners in total, including the start and end points, which are marked with red points. This data was used for testing only.

4.4.1.0.4 Original ShortStraw Data

The results in Table 4.1 are based on the test set used in the ShortStraw paper; we also used it to help set the thresholds for IStraw-C. We chose to use this data to ensure our implementation of the original ShortStraw algorithm had the same results as Wolin et al. [93]. The results show that our ShortStraw implementation did indeed give us the same results as [93] and that IStraw-C obtains very high correct corners and all-or-nothing accuracy with optimized parameters.

Table 4.1: Accuracy results for ShortStraw and IStraw-C, our algorithm without curve detection. The results are for the data used in the original ShortStraw paper.

	ShortStraw	IStraw-C
False Positives	32	2
False Negatives	38	1
False Strokes	63	3
Correct Corners Accuracy	0.979	0.999
All-or-Nothing Accuracy	0.741	0.998

Table 4.2: Accuracy results for ShortStraw, IStraw-C, ShortStraw+C, IStraw, MergeCF and the Sezgin corner finding algorithms (652 polyline strokes).

	ShortStraw	IStraw-C	ShortStraw+C	IStraw	MergeCF	Sezgin
False Positives	32	2	21	2	10	23
False Negatives	93	12	94	0	20	461
False Strokes	106	14	99	2	23	286
Correct Corners Accuracy	0.982	0.997	0.982	1.000	0.996	0.911
All-or-Nothing Accuracy	0.837	0.979	0.848	0.997	0.965	0.561

4.4.1.0.5 Polyline Ink Stroke Test – Dataset One

For the polyline ink stroke test, we wanted to examine IStraw with and without our curve detection extension to determine if it would cause any accuracy degradation for polyline ink strokes. We were also interested in the accuracy of other corner finding algorithms. Table 4.2 shows the results of testing these six algorithms on the polyline stroke part of our first dataset.

4.4.1.0.6 Curve Detection Tests – Dataset One

To test whether IStraw works better for strokes containing curves, we conducted experiments with the stroke data from dataset one with curves. Note that shapes we tested were the same as the

Table 4.3: Accuracy results for ShortStraw, IStraw-C, ShortStraw+C, IStraw, MergeCF and the Sezgin corner finding algorithms (595 strokes with curves).

	ShortStraw	IStraw-C	ShortStraw+C	IStraw	MergeCF	Sezgin
False Positives	8351	8613	39	5	663	1090
False Negatives	35	10	92	5	32	196
False Strokes	595	595	103	9	294	424
Correct Corners Accuracy	0.990	0.997	0.974	0.999	0.991	0.944
All-or-Nothing Accuracy	0	0	0.827	0.985	0.506	0.287

training shapes (see Figure 4.18). Table 4.3 shows the results of testing these six algorithms for the strokes with curves.

4.4.1.0.7 Tests on Dataset Two

Table 4.4: Accuracy results for IStraw, MergeCF and the Sezgin corner finding algorithms (395 strokes of new shapes).

	IStraw	MergeCF	Sezgin
False Positives	13	139	172
False Negatives	7	4	116
False Strokes	19	90	184
Correct Corners Accuracy	0.997	0.998	0.955
All-or-Nothing Accuracy	0.952	0.773	0.534

In our previous tests, we showed IStraw has higher accuracy over the other corner finding algorithms. However, the shapes we tested were also used in training. To explore corner finding accuracy on strokes that were not used in tuning the corner finders, we used a new testing dataset from the shapes in Figure 4.20. Table 4.4 shows the results of testing the three curve-stroke recognizers: IStraw, MergeCF and the Sezgin algorithm on dataset two.

4.4.2 Analysis of Computational Complexity

In order to get higher corner finding accuracy, we developed IStraw by making several refinements and adding new components to the ShortStraw algorithm. The question arises as to how these changes might increase the computational complexity of our approach compared with ShortStraw. To investigate this, we examine each change made and compare the computational complexity between our algorithm and ShortStraw. First, we set the number of raw points to M and number of resampled points to N. We further assume that the number of corners is C.

During resampling, we use the same algorithm as the one in [93] and the runtime is O$(M+N)$. For the polyline corner finding component, we did not modify the bottom-up component of the O(N) algorithm, but use the speed data to add more potential corners. We have many enhancements in the top-down approach, but all these will not affect the computational complexity, so this part runs in time O(CN) and the running time for the worst scenario is O(N^2). To avoid consecutive false corners, we need one more loop, whose iteration time is C, the number of corners.

The last part of our algorithm is curve detection, and the algorithm contains two loops to remove the unnecessary corners. These two loops are similar. The iteration number of each one is C, so the computational complexity is O(C). In conclusion, the computational complexity of our algorithm is O$(M + N^2 + C)$, exactly the same as ShortStraw.

CHAPTER 5: METHODOLOGY

5.1 Introduction

The goal of this dissertation is to create a 3D human model that looks like a specific participant, which means that the 3D model has accurate geometry based upon the collected real world data and a correctly mapped image texture. The hardware setting of our system is simple, and it is composed of one Microsoft Kinect and a regular personal computer. With the exception of the algorithms that we implemented in our system mentioned in Chapter [**?**], all components of what we introduce in this chapter are novel algorithms and processes or involve significant improvements to existing techniques.

In order to generate 3D human models for the participants, we use one human template, which is a triangle mesh attached to a skeleton. Our modeling technique is similar to the approach created by Kraevoy et al. [48] in 2009. The advantage of this approach is that the relationship between the vertices and the skeleton won't change, so the deformed model is still attached to the original skeleton. Under these conditions, we can control the generated human model with human tracking techniques or other kinds of input data. One major difference between our method and Kraevoy's is that we use the Kinect to get the contour of the real person automatically instead of drawing a contour. We also improved some existing contour matching technique with modified IStraw algorithm and the process to generate a texture map for the 3D human model.

In this chapter, we will emphasize our unique contribution to the human modeling and related research areas. The first section is an overview of our method and of each stage of the system pipeline. We then will present the detail of the main challenges addressed here, including contour extraction, contour segmentation, and texture mapping.

5.2 Main Pipelines

This section presents the main pipeline of our human modeling system and each modules's functionality. Some of the modules use existing work mentioned in the previous chapter; the rest have our own unique contributions, which we will describe in this section. The main pipeline is divided into two parts based upon whether the computation can be pre-executed. One advantage of mean-value encoding [47] is that the encoding result will be unchanged if one focuses on a single t3D model. Since we only use one template in our system, we only need to encode this model once and save the result. Whenever a new session starts, our system will load the saved data for the decoding process.

Figure 5.1 (a) displays the three pre-processed steps: anchor deciding, model encoding, and data saving. The first step is to determine the anchors of the human template and save these points to a XML file. The properties of the non-anchor vertices are then calculated with the mean-value encoding algorithm, see Section 3.5. In order to get rid of the expensive encoding process for each test, we save the encoding to another XML file.

Figure 5.1 (b) shows the remaining processes for human modeling. These processes are required for each sessioin. First of all, the system needs to load pre-computed data, including the human template, the anchors information, and the encoding data. The participant then stands in front of the Kinect with the initial pose, which is the same as the default pose of the template (see Figure 3.1). Once the participant's pose matches the requirement, the color image and depth image from the Kinect are used to extract the contour of the participant. The contour is segmented into eight pieces, the details of which process is presented in Section 5.3. Points on each contour segment are matched to one anchor vertex and then the model decoding starts by following the mean-value decoding approach mentioned in Section 3.5. After we get the geometry of the participant, we can generate the texture map based upon the color image from the Kinect and compute the texture

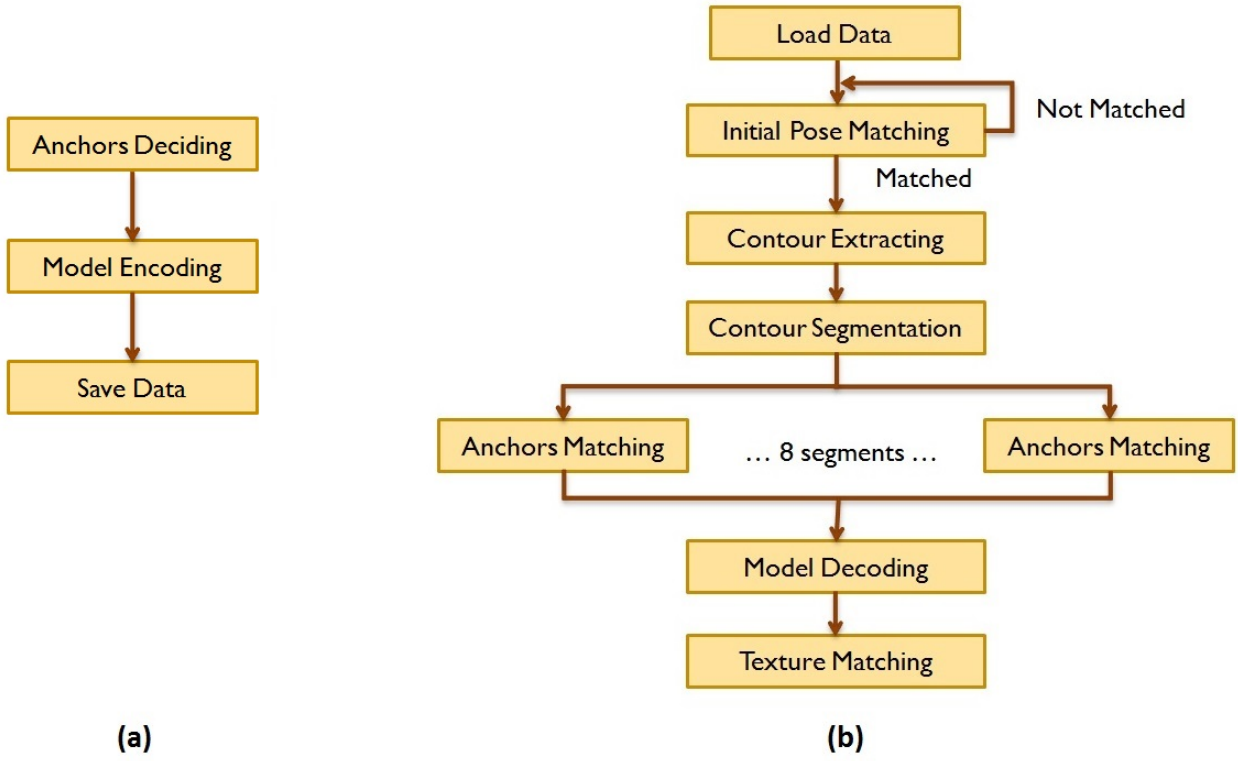

Figure 5.1: Main pipeline of our human modeling system. (a) Pre-computed part. (b) Steps for each test.

coordinate of each vertex.

5.2.1 Anchors Deciding

Anchors are those vertices that won't be affected by the reshaping algorithm. During the 3D model reshaping process, these vertices are either fixed or set by the user or real world data input. The remaining vertices in the 3D model are changed based upon the new location of these anchors. In our system, there are two kinds of anchors: one kind includes vertices that will be matched to the real world data directly; the other includes vertices that won't change shape. In order to better

distinguish these two kinds of anchor vertices, we call the first kind edge anchors and the second kind end anchors.

When the camera is put in front of the 3D human template and facing it, the edge anchors will be located on the silhouette of the model and this is the reason for naming these vertices edge anchors. Instead of manually marking these vertices, we introduce an edge anchor finding algorithm. The human template is rendered as white on a black background, so the 2D contour can be easily identified. The projected position of each vertex is calculated with the matrix, which is the same as the camera matrix for the rendering. The vertex v_i, whose projected position p_i is close enough to the contour, could be an edge anchor. What we mean by close enough is that the minimum Euclidean distance d_i between p_i and the contour is below a threshold of one pixel. Ideally, p_i is located on the contour and d_i is equal to zero. However, the rendering output is rasterized, so we set our threshold to tolerate the error introduced by rasterization and the calculation accuracy. So far we can get all the anchor candidates, but some vertices, like vertex D in Figure 5.2, must be removed from the list. Vertex D is close enough to the contour and it will be an anchor if there is no B, which is closer to the contour. One false anchor can be detected when it meets the following conditions. Suppose this anchor candidate is D, and more than two of its neighboring vertices, A, B, and C, are anchor candidates. Furthermore, vertices A and C are not connected, and B is closer to the contour pixels. In this case, B is an anchor and D is rejexcted as a false anchor candidate.

The end anchors are vertices belonging to the 3D human template's head, hands, and feet. These end body parts cannot be deformed based upon contour information. The head shape is not related to the facial features. For example, a person with big head may have a small nose, but a person with a small head may have a big nose. The head contour can also be affected by the hair style. We tried deforming the template's head based on a female head contour, and the result was unacceptable. The whole face became contorted, because the left canthus (corner of the eye where the upper and lower eyelids meet) is stretched to the upper-left and the nose is shifted to the right. Therefore, we

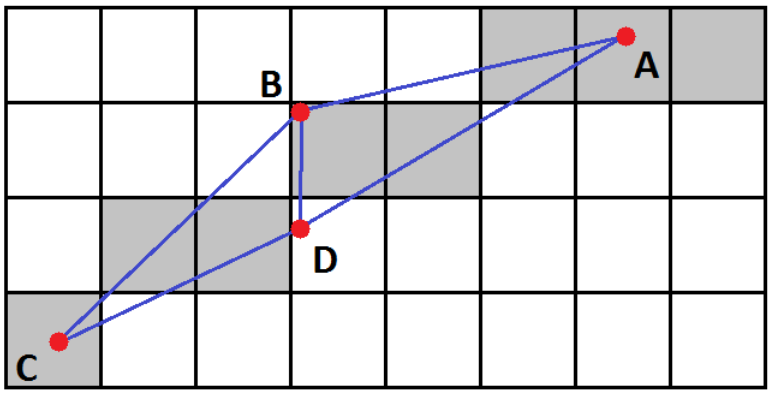

Figure 5.2: An example of false anchors. Gray squares are the contour pixels, Red points are the projected positions of anchor candidates, vertex A to D, and blue lines are connections among these vertices. Vertex D is a false positive.

won't deform the face but only transfer, scale, and rotate it based upon the face tracking results from the Kinect. Although the face features are incorrect, the result looks fine with the texture. For the hands and feet, we transfer them to the right location without any reshaping. The contours of the hands and feet are related to the movement of fingers and toes, and the participant will wear shoes unlike the human prototype, who is bare foot. Furthermore, it is impossible to get a precise contour of these parts, since they have signifant details, are lost due to the low resolution of the color image.

5.2.2 Contour Extracting

In order to get the contour of the user automatically, we took advantage of the depth camera provided by the Kinect SDK and its human tracking technique. Before running the MR system, we pre-compute the angle range of each tracked bone by fitting it to the initial pose. When we start the system, the Kinect tracks the participant in the scene and compares his/her tracked bones

to the pre-computed angles. After the pose is matched, the system will load the depth image and corresponding color image from the Kinect as the input to the contour generating function.

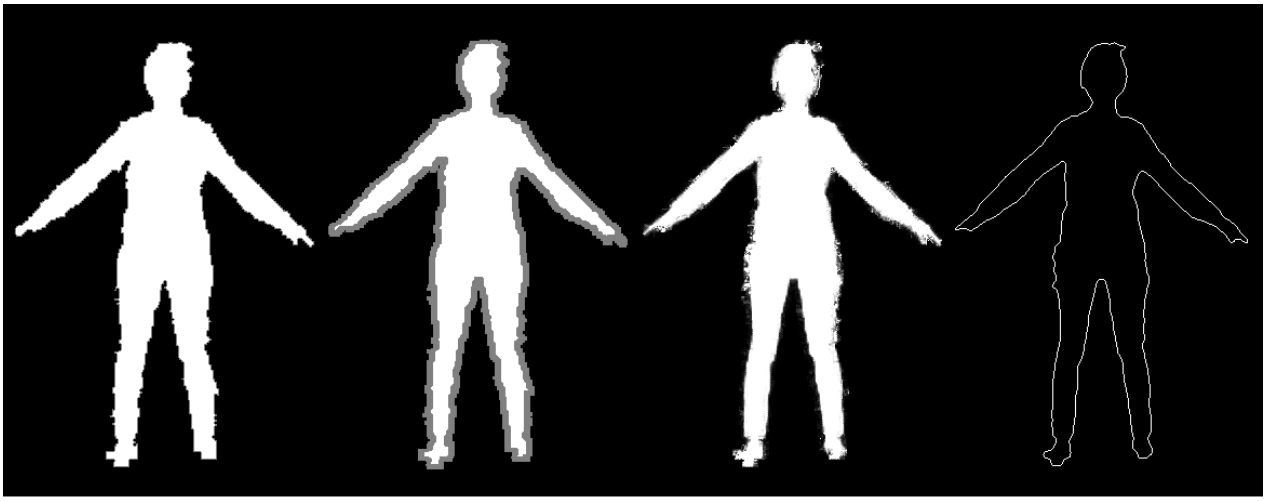

Figure 5.3: Contour extracting pipeline: (a) raw participant edge map; (b) automatically generated trimap; (c) generated alpha map; (d) contour pixels.

Figure 5.3 shows the result of the automated contour extracting process. The depth image and the human tracking information can provide a rough outline of the tracked participant, as shown in Figure 5.3 (a); however, this raw data is not accurate due to errors in the depth image. This raw outline will be used to determine the trimap of the participant, as seen in Figure 5.3 (b). With the trimap, depth information and the color image, our matting algorithm is able to generate a precise matte of the participant, as displayed in Figure 5.3 (c). The generated matte determines the contour of the participant with a threshold setting, as depicted in Figure 5.3 (d).

5.2.2.1 Rough Contour Shift

The depth map and human tracking information from Microsoft Kinect SDK 1.5 can determine the pixels of the depth map belonging to the participant. These pixels will be mapped to the corresponding pixels of the color image with the mapping function provided by the Kinect SDK and a coordinate transfer function. From Figure 5.4 (a), we can see that the raw edge marked with red is not well-aligned to the color image, with the major issue being pixel shifted to the right of the image borders.

Figure 5.4: Raw edge from Kinect depth data and human tracking information: (a) red pixels transferred from depth image space to the color image space only with the Kinect SDK mapping function. (b) green pixels shifted to left to be closer to the image borders than the red pixels in (a).

For our process to start, we ask the participant to stand at a specified target location to get an addi-

tional transfer function to better map the depth data to the color image. We suppose the minimum column index is j_{min}, and the maximum column index is j_{max}. After getting the mean value of shiftness of the pixels, whose column index is j on the left, where $j < j_{min} + 5$, and pixels on the right, where $j > j_{min} - 5$, we can estimate the linear relationship between the shiftness j_{shift} and the column index of each pixel. The linear equation to compute the shiftness is:

$$j_{shift} = r * (j - j_{min} + 0.5 * j_{diff})/j_{diff}, \tag{5.1}$$

where $j_{diff} = j_{max} - j_{min}$, and r is 4. This transfer function will shift each pixels of the participant to the left based upon the column information of each pixel of the participant. As shown in Figure 5.4 (b), the result marked with green after the shifting function is more accurate and able to provide a more reasonable trimap. In order to configure this equation, we compared the shifted rough contour with the manually marked real contour on the color image of the training data. Current parameters provides the minimum mean Euclidean distance between the shifted and the real contours.

5.2.2.2 Trimap Generation

The shifted pixels from previous step can generate a rough contour of the participant, like Figure 5.5, in which the pixels outside the human body are marked with black; the pixels on the edge of the participant are grey; and the pixels inside the body are white.

The grey edge will be expanded to form the unknown area of the trimap; the left black area will be the known background; and the left white area is the foreground. The easiest approach to generate the unknown area is to set a constant width w, and expand each grey pixel to a $w \times w$ window. The initial contour pixel is in the center of the window. Suppose the pixel,p_{ij}, is located at the

Figure 5.5: Raw contour map of the upper body of the participant.

ith row and jth column of the color image, then any pixel p_{xy} with $i - w/2 \leq x \leq i + w/2$ and $j - w/2 \leq y \leq j + w/2$ will be unknown pixel in the trimap.

The window size w is chosen carefully. If the window size is too small, some pixels belonging to the foreground may be marked as background on the trimap, and vice verse. However, if the window size is too large, the increasing unknown area will reduce the accuracy of the matting result and increase the execution time of the algorithm. In our approach, we set w to 7, which will produce the best result.

5.2.2.3 Matting Result Postprocess

The result of the matting process can provide a precise contour of the participant with the trimap from the previous step along with the color image and depth image from the Kinect. Our matting

approach is based upon the global sampling method of the natural image matting proposed in [38]. However, we use an automatically generated trimap from our previous step instead of a user specified trimap to get the background and foreground samples. The details of the single image matting will be discussed in Section 3.3.

The result of the mapping algorithm is an alpha map, in which each pixel is represented by an integer from 0 to 255. The larger the alpha value is, the more likely the pixel belongs to the foreground, which is the participant in our system. The generated alpha map will provide an accurate matte of the participant from the background by setting a threshold to 127. The contour pixels are the pixels of the participant but having neighbor pixel(s) that belong to the background, and the rough normal of each contour pixel is computed as well.

The generated alpha map has many errors caused by an unreliable depth map and the mapping between the depth and color image. In Figure 5.3 (c), we can see that the alpha map is not continuous on some neighboring pixels. These errors will falsely mark some background pixels as foreground or vice verse. The reason is that the colors of some foreground and neighboring background pixels are close. The resulting contour of the original alpha map will have significantf noise.

In order to get a more reasonable contour, we use an algorithm to smooth the alpha map. The new alpha value of one pixel p_i is the weighted sum of neighboring the pixels' original alpha value in a limited radius of 5 pixels. The weight of each neighboring pixel p_j is based upon a 2D Gaussian function:

$$W(p_i, p_j) = A \times exp(-\frac{(p_j.x - p_i.x)^2}{2\sigma^2} - \frac{(p_j.y - p_i.y)^2}{2\sigma^2}), \tag{5.2}$$

where $A = 0.0522$ and $\sigma = 1.7607$. The parameter σ determines the decay speed of the Gaussian function. The larger σ is, the more smooth the resulting alpha map will be, since the new alpha value weighs more neighboring pixels' value. The best σ should be able to remove noise and preserve the curvature feature of the contour. The other parameter A is based upon σ to make sure

that the sum of all the weights is 1. We compared the mean Euclidean distance d between the extracted contour and the real contour of training data to configure the parameters for the Gaussian function. This parameters can provide the best result, minimum mean d of all the training data.

5.3 Contour Segmentation

In order to improve the accuracy of the matching between the contour of the participant and the anchors of the 3D prototype, we use the Kinect skeleton tracking data and IStraw-based algorithm to segment the contour. Unlike the contour in [48], which preserves the curvature feature of the template model, the contour of the participant in our system can be quite different from the prototype model. These differences are caused by different body shapes and clothes of the participants. Therefore, simply copying Kraevoy et. al's matching algorithm will result in many mismatches. We segment the contour into several parts and match these segments with anchor sub-paths to increase the reliability of the matching criteria.

5.3.1 Contour Segmentation Algorithm

IStraw is a general corner finding algorithm for pen-based interfaces, but our contour segmentation problem only needs to determine which body part each contour point is corresponding to. Unlike sketch-based interface, our system does not have time information of points on the contour. However, we have extra information not available in the sketch-based case, like the rough shape of the contour, and the tracked skeleton of the participant. We need to take advantage of these extra data and create a more robust and more accurate contour segmentation algorithm for a specific human pose (Figure 3.1).

Our contour segmentation process includes three steps:

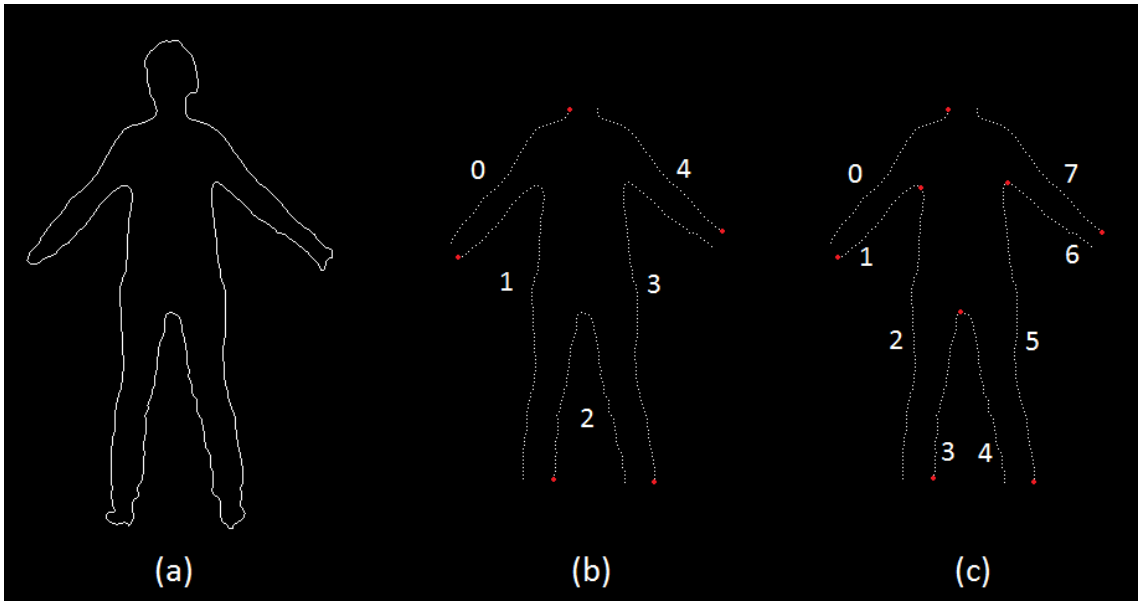

Figure 5.6: Contour segmentation process. (a) Initial contour path. (b) After skeleton based segmentation. (c) After corner based segmentation.

Resampling is the process to resample the initial contour path, as Figure 5.6 (a).

Skeleton based segmentation is the process to remove some contour points and group the rest into five contour segments. As shown in Figure 5.6 (b), these five segments start from the five red dots.

Corner based segmentation is the process to find some specific corners and further segment the contour into eight paths, as Figure 5.6 (c).

5.3.1.1 Resampling

Before segmenting the contour into different body parts, the sorted contour pixels is resampled into points with the same interval. This process is necessary for two reasons: the first is to remove the noise of the path for the accuracy of the corner finding algorithm, and the second is to decrease

the density of the contour points for the efficiency of the matching algorithm.

The interval distance between two adjacent resample points is chosen carefully based upon the requirement of both corner finding and matching algorithms. If the interval is too small, the corner finder may detect some noise as corners, but if the interval is too big, some corner may be missing from the result. The smaller the interval is, the slower the matching process will be, because the computational complexity of the matching algorithm is proportional to the number of resample points. Furthermore, each anchor must be mapped to at least one resample point, so the number of resample points should be more than the number of anchors. Unlike IStraw, which should works for strokes of different size, our human modeling system take participants' contour with similar length as input, since human beings' contour size won't be quite different. Therefore, we set the interval between sample points to a constant number of pixels. After testing different parameters, we determined that three is the best trade-off. If the interval distance is 4 pixels, there might be more anchors than the resample points. If the interval distance is 2 pixels, it is easier to introduce false positive into the corner finding step.

5.3.1.2 Skeleton Based Segmentation

After resampling the contour path, we can start the segmentation with the skeleton data of the participant. In this step, we will remove the contour points of the *end body parts*, including head, hands, and feet, and segment the contour into five parts, including neck to left wrist, left wrist to left ankle, left ankle to right ankle, right ankle to right wrist, and right wrist to neck (Figure 5.6 (b)).

The skeleton data from the Kinect can provide the 3D positions of the head, neck, hands, wrists, feet, and ankles. These points are projected to depth map space and then mapped to color image space. We also compute the direction from neck to head d_{head}, left wrist to left hand d_{lhand}, as well

as the same process for d_{rhand}, d_{lfoot}, and d_{rfoot}. Each resampled contour point will be checked whether it belongs to one of the five body parts. Take the head for example, one contour point p_i is on the head and will be removed from the contour path, when the distance between p_i and the neck point on the image space p_{neck} is less than 80 pixels and p_i is on the head direction $(p_i - p_{neck})\dot{d}_{head} > 0$. The same process will be executed for hands and feet, except the distance limitations are 50 pixels for hands and feet instead of 80. These limitations are determined by the observed maximum size of head, hands, and feet.

The contour points continuously belonging to one body part and two neighboring contour points cannot be on two different end body parts. Suppose the folling are three consecutive contour points ,p_i, p_{i+1}, and p_{i+2}, then it is impossible for p_i and p_{i+2} to be on one body part while p_{i+1} belongs to another. The end body parts are not connected to each other as, for instance, there are shoulder and arms between head and hands. Therefore, if p_i is on the head, p_{i+1} cannot be on the hands or feet.

Previous features enable us to remove these end body parts and segment the contour into five paths simultaneously. The initial contour path always starts from the head, so we will check whether contour points belong to the head one by one. If the point is not on the head, it will be added to the first contour segment. After this, the following points will be tested for being on the left hand. If these continuous points are not, they will be added to the first segment, and once one of them belongs to the left hand, the first following point not on the left hand will be the first point in the second segment. The same process will be executed for the left foot, right foot, right hand, and head in sequence.

5.3.1.3 Corner Based Segmentation

After this, a skeleton-based segmentation algorithm based upon IStraw [96] is used to further divide the contour into more segments. In 5.6 (b), we can see that there are three obvious corners on the contour path 1, 2, and 3, including armpits and crotch. These points are hard to be identified correctly by the tracked skeleton. Take left armpit for example, it is close to the left shoulder joint, but it is impossible to get the precise searching direction. In order to find these corners, we need to take advantage of the contour shape of the fixed initial pose.

The initial IStraw algorithm is proposed as the initial step for gesture recognition without any knowledge of the stroke. The testing stroke of IStraw can be any shape and any size, but the contour for segmentation in our system has similar shape and size. By analyzing the contour paths, we notice the difference between them and the ambiguous strokes tested in the IStraw algorithm.

• There will be only one most noticeable corner on the contour path from 1 to 3.

• These corners may be located on a short curve instead of a obvious peak.

• These corners have sharp angles between themselves and two adjacent corners. The angles of armbits are less than $90°$, and the angle of the crotch will be smaller than $60°$.

• There is some noise caused by cloth folds of the participant.

These new features require new thresholds and process adjustments for IStraw to provide accurate and robust results.

After adjusting corners (see Section 4.3.5), we run multiple triple collinear tests for any three adjacent corners (including the start and the end points) until there is only one corner left. In order to remove noise caused by cloth folds instead of a real corner, we initialize the threshold for collinear test as 0.975, and then decrease it during each iteration. We change the threshold gradually instead of setting a small threshold at the beginning, because small thresholds may result in deleting the right corner. A false corner after the right corner may reduce the reliability of the

collinear test – the detailed explanation is located in Section 4.3.4. The curve detection process will be removed, since it may delete the corners that we are looking for.

5.3.2 Analysis

The parameters for corner based contour segmentation is different from IStraw. IStraw may find corners around the cloth folds, but we only need corners around armpits and crotch of the participant. In order to configure better parameters for segmentation, we used new training data of the real person instead of the strokes in [96].

Segmenting the contour path before matching to anchor vertices can improve the matching result. The human prototype is modeled with a tight T-shirt and shorts, but the participant may wear a jacket or pants. The different clothes and cloth folds may introduce extra curves, which can cause mismatch or shiftiness, since Kraevoy et. al's algorithm [48] uses normal difference as a matching criterion (Section 3.4.1). Furthermore, more contour segments mean the initial alignment will be more accurate. This improvement will enhance the credibility of the proximity of the matching criteria (Section 3.4.1). These matching errors will bring further problems in the generated 3D human model.

Figure 5.7 shows the generated 3D models with different contour segmentation. The first 3D human model is the best and it comes from the full segmentation algorithm to generate eight contour paths. In Figure 5.7 (b), we only use the skeleton-based segmentation to divide the contour into five segments and remove the corner segmentation step. The second model is okay, but there is an obvious artifact around the right armpit, which overlaps the contour and misclassifies some background pixels as part of the human texture. We also test the result without any contour segmentation. The last human model (Figure 5.7 (c)) is the worst, especially the wrists, hands, and the right armpit. The wrists are too slim, and the hands are shifted to the wrong location. From

Figure 5.7: The generated 3D model of different segmentations. (a) Skeleton-based and corner-based segmentation. (b) Skeleton-based segmentation only. (c) No segmentation.

the comparison, we notice that the mismatch on a flat contour area won't cause many problems. However, any trivial mismatch will lead to noticeable errors in the generated human model.

The end body part removal will cause contour path disconnects on the wrists and ankles. If the mismatch happens around the disconnect points, it will result in the wrist or ankle of the generated human model being twisted. Figure 5.8 is the demo of the simplified left wrist area, and the real prototype is more complicated. The correct matching is v_j to p_i and v_k to p_{i+1} and the correct modeling result is shown in Figure 5.8 (b). However, one mismatch (v_{j-1}, p_i) will cause further wrong match pairs (v_j, p_{i+1}). This error will make the 3D model become slim and twist as seen in Figure 5.8(c).

Even when the contour points are continuous, any mismatch around the contour corners, like armpits and crotch, may make the generated 3D models be bloated. In Figure 5.9 (b), the ver-

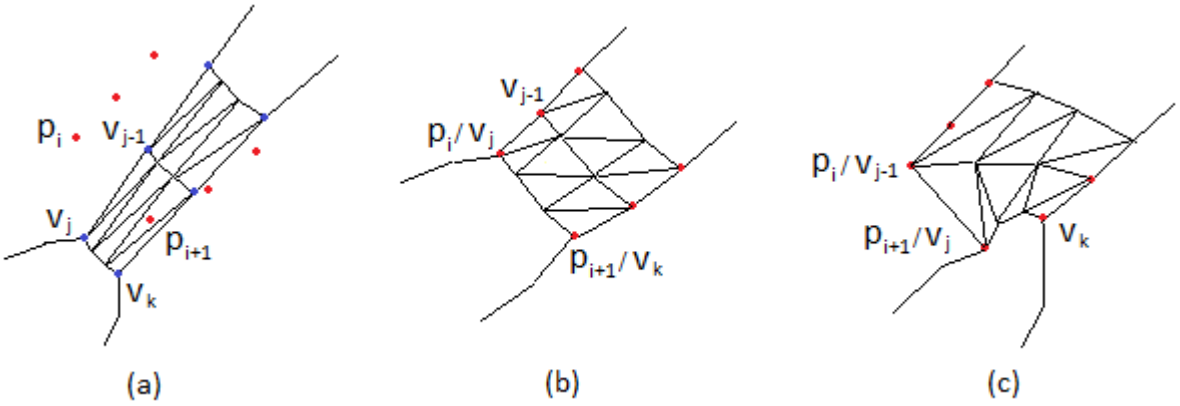

Figure 5.8: Left wrist area modeling demo. Contour points are marked with red dots, anchor vertices are marked with blue dots, and 3D model is the black triangular mesh. (a) Contour points and 3D prototype. (b) Generated model with correct matching. (c) Generated model with incorrect matching.

tex v_j is mismatched to p_{i+2} instead of p_i, which is not matched to any anchor point, and the generated 3D model will be over the contour.

False corners will cause false contour segmentation, and result in serious errors in the generated model. In Figure 5.10, we use the original IStraw algorithm for corner finding, and it will output the red dot on the left image as a corner, because of a cloth fold on the pants. The following step takes the red dot as the right armpit and segments the contour sub-path there. Therefore, the anchor vertices from the right ankle to right wrist will match to the wrong contour point, and the corresponding body parts are improperly modeled. As shown in the right image of Figure 5.10, the generated right leg and arm do not match the input contour at all.

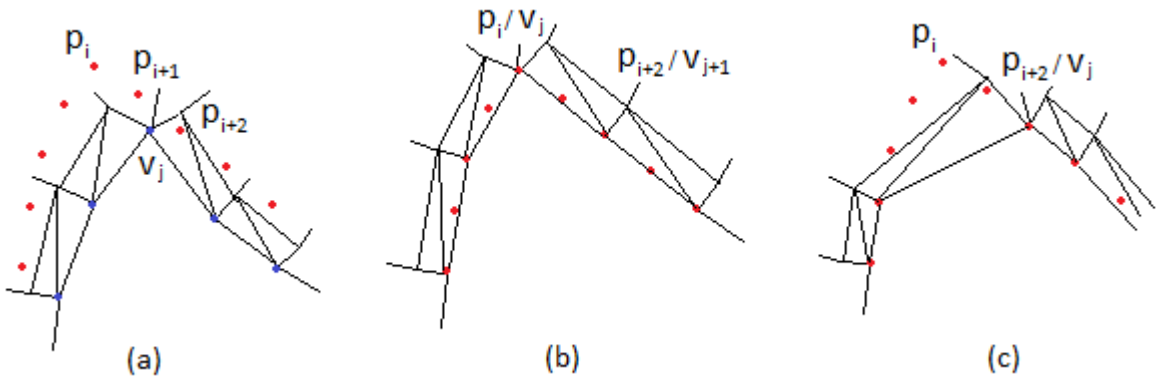

Figure 5.9: Left wrist area modeling demo. Contour points are marked with red dots, anchor vertices are marked with blue dots, and 3D model is the black triangular mesh. (a) Contour points and 3D prototype. (b) Generated model with correct matching. (c) Generated model with incorrect matching.

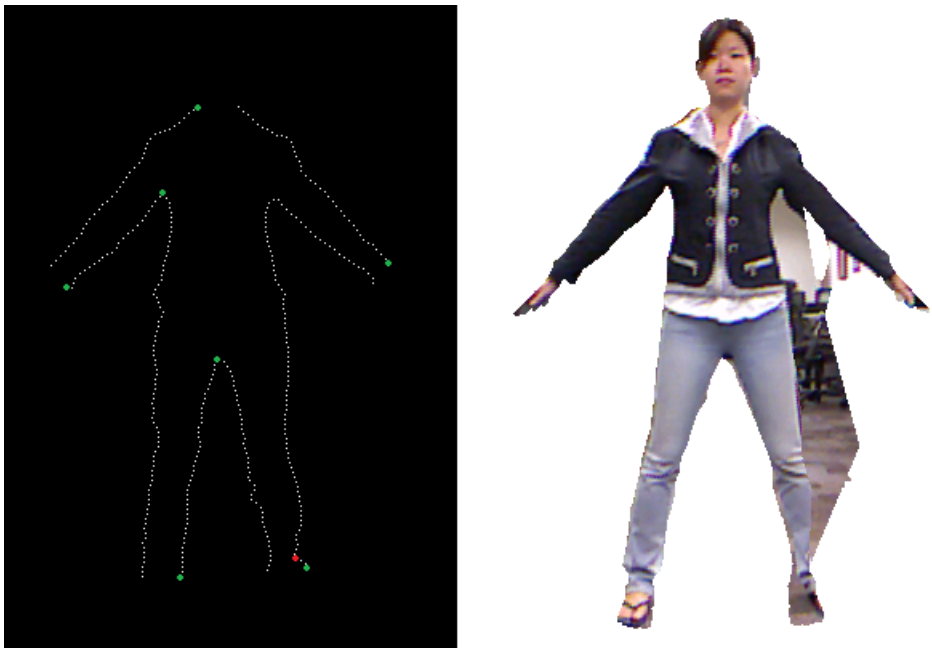

Figure 5.10: Finding corner with original IStraw algorithm. Left: False corner marked with red dot. Right: 3D human model generated from the contour segmentation on the left.

CHAPTER 6: EVALUATION

6.1 Interface Design

In order to trigger the human modeling process automatically, the participant must stand in an initial pose. When the system starts, the color video streaming from the Kinect will be displayed, so the participant can see his/her movement. However, this is not enough to guide the participant to the right pose, especially for those who are not familiar with the system. We decided to semi-transparently render the human template in the initial pose at the location where the participant is supposed to stand. The rendering color of the model will change based upon the status of the data collecting process. Figure 6.1 shows the work flow of the interface. The first state is to help the participant to adjust his/her position or pose. Once the position and pose are matched, the interface

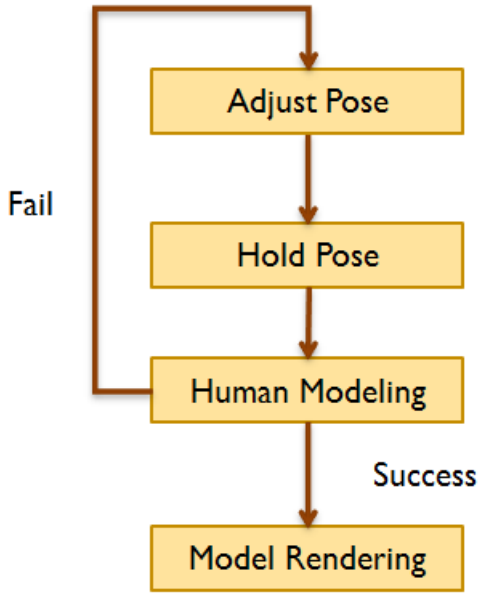

Figure 6.1: The work flow of the interface for human modeling.

will notify the participant to hold the pose for three seconds and then will save the data required for the modeling process. The next state is human modeling, and the participant does not need to hold the pose any longer. If the human model is created successfully, the human template will be deformed to the generated model. If not, the system will return back to the initial state.

Figure 6.2: The initial state of the interface for our human modeling system.

The initial color of the 3D human template is dark yellow, (60, 60, 0), and the output video is the sum of the model and the Kinect color streaming. As shown in Figure 6.2, the area with the human template is lighter and more yellowish. Therefore, the participant is able to adjust his/her position and pose based upon the template's. The human prototype is semitransparent, so the participant can see the prototype as reference and his/her own movement in the video. The rendering scheme

and the color of the human template is chosen carefully. We tried wireframe rendering, but the model looks scary, especially the face. The human template is only used as a reference and does not need to look real, so we decided to render it as a flat surface without shading and shadows. The color yellow is chosen because it is noticeable and is different from the colors used in the following steps. Since the color is added onto related pixels of each frame, it will be hard for the participant to see his/her movement inside the overlapped area if the chosen color is too bright. However, if the color is too dark, it is hard to see the template as a reference. In both situations, the participant cannot see the movement and the prototype's pose clearly at the same time, so he/she may feel frustrated when it takes a long time to pose correctly.

Figure 6.3: The second state of the interface for our human modeling system.

Figure 6.4: The third state of the interface for our human modeling system.

Once the pose of the participant matches the initial pose, the human prototype will be rendered with red, as in Figure 6.3. Then the participant must freeze until the prototype turns to green, as in Figure 6.4. During these two states, the template is rendered with a bright color, because the participant no longer needs both the template and the color streaming as references to adjust his/her pose. The interface must provide a strong signal to notify the participant what to do. Generally, the color red means stop and green means free to go, as with a traffic light. Therefore, we take advantage of this common sense to help the participant to quickly understand the interface.

The system requires the participant to hold the initial pose for several seconds. This is essential

to the consistency of the color image, depth data, and the skeleton data from the Kinect. When the participant adjusts the pose, he/she will keep moving. If the system takes the available data from Kinect immediately after the pose is matched, the participant may have moved, resulting in saving different data. The Kinect has different latency and processing time for different types of data. The color image is the fastest, because it is the raw data from the color camera and the only latency is due to data transmission. The depth map will be slower, since the Kinect SDK needs to analyze the input infrared data and corresponding pattern to generate the depth information. We use the skeleton information from the Kinect SDK for pose matching. Suppose that the Kinect SDK used the depth data at time t_d for human tracking and generated the skeleton data after a delay of t_s. If the matching process requires time t_m and the delay to get the first color image after the matching is t_dc, then the color image used for human modeling is taken at $t_c = t_d + t_s + t_{dc}$. If the delay for the first available depth data is t_{dt}, the tracking result and depth data used is taken at $t_t = t_d + t_s + t_{dt}$. The latencies between availability of these data will cause problems during the contour extraction and contour segmentation processes. The generated trimap, which is based upon the tracking result, won't match the color image and can introduce significant errors to the matting result. The alignment of the skeleton and depth data is essential to the reliability of the contour segmentation.

Figure 6.5 shows the latency between the human tracking result, color image and depth data when we save the first available data after the pose is matched. It is easy to notice the pose differences of the arms, since the participant is moving her arm up and down to figure out the right pose. Our solution is simple but effective. We change the prototype's color to warn the participant to stop moving, and employ a 3 seconds latency between the color changing and the input data saving. When the prototype turns to green, it means the required data is saved and the participant can move freely. If the participant is familiar with the system, one second is generally long enough for the system to change the interface and the participant to react to the change. However, partic-

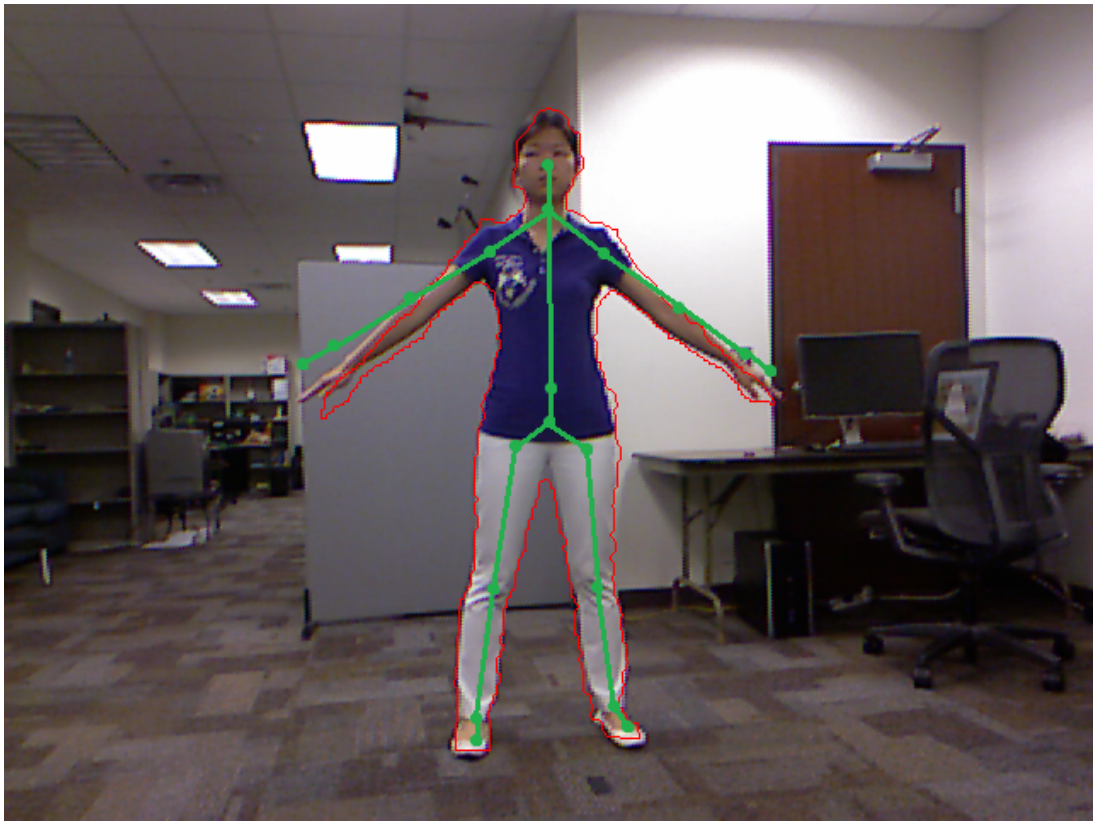

Figure 6.5: Latency between different Kinect data: (Background) the color image; (Red) the outline of the participant's tracking result; (Green) the saved skeleton information.

ipants, who are not familiar with the system, commonly need more time to notice the change and understand what is happened. Therefore, we set the time interval to 3-seconds, which is safe for the different levels of participants. The color used in the interface is chosen carefully as well. Red is a common sign for stopping, while green means free to move, like the traffic lights.

6.2 Test Process

Before running the formal test, I collected the test data of myself four times with the test system as the training data. Each time, I am wearing different clothes to get rid of the ambiguity introduced by clothes. The training data is used to debug the system and configure the parameters used in the system. All the ground truth is manually marked, such as the real contour and the real corners. The scheme used for choosing the parameters is discussed in the previous chapter along with the algorithms.

The goal of the system test is to evaluate the accuracy of the human modeling results of using our system. The usability of the system interface is not addressed in this evaluation process. The test was held inside the Synthetic Reality Laboratory (SREAL) at UCF, and all the participants were faculty members and students working in the lab. Therefore, most are familiar with the idea of human modeling and have experienceof using the Kinect. There were ten participants in total, 2 females and 8 males. Figure 6.6 shows the silhouettes of the ten participants, and it is obvious that their body shapes are quite different. For example, some of them have large heads, some have wider shoulders, and some have long legs. The participants also wear different clothes, like short sleeve T-shirts, long sleeve shirts, and flared leg pants. The body shape and clothes variance help to put high demands on the reliability of the evaluation process, allowing us to test whether our system works for most people or just a small group of people having specific body shapes.

At the beginning of each test, I tell the participant to stand at a specific location in front of a Kinect and follow the initial pose of the human template displayed on the screen. Once the pose is matched, the participant must hold the pose until the template on the screen becomes green. Then the participant's job is done. Our test system saves all the data into binary files used for the human modeling process. Afterward, the system compares the results of different algorithms using the same test input. The saved data include color image, depth image, color-depth matching informa-

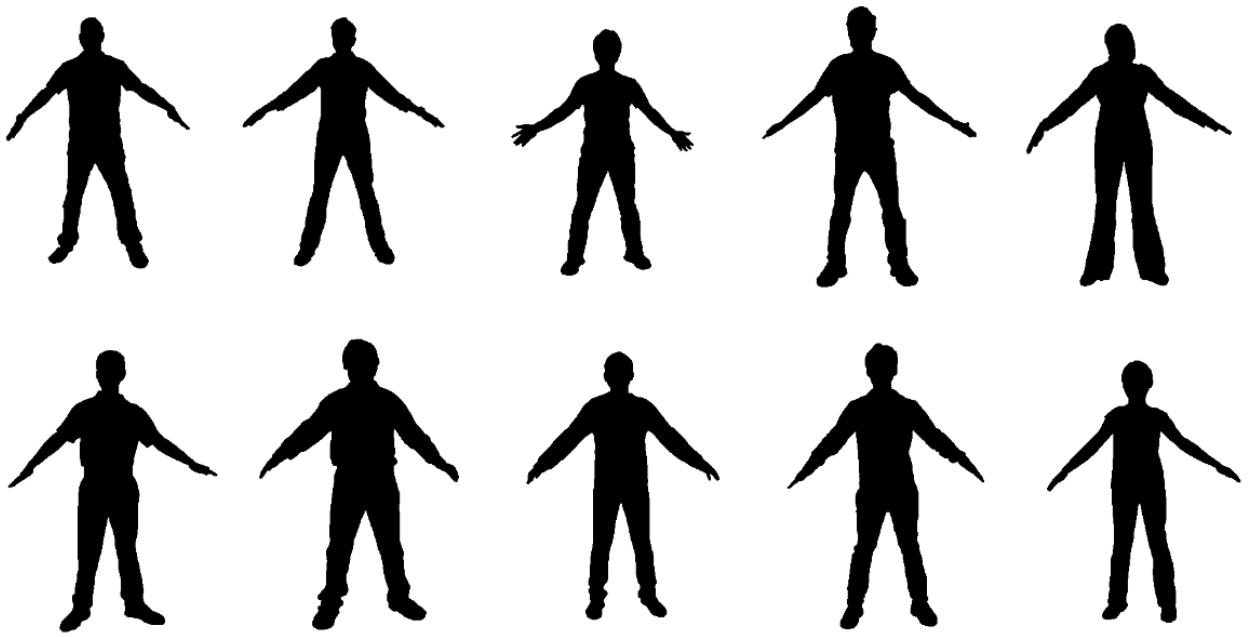

Figure 6.6: Silhouettes of the ten participants.

tion, human pixels, skeleton positions, and model face parameters.

• The color image is the 640×480 rgb image from the color camera of the Kinect.

• The depth image is the 640×480 depth data from the Kinect sdk, which will process the data collected by the infrared camera.

• The color-depth matching information is the data of the coordinate correspondance between the color image space and the depth image space. The Kinect sdk is able to provide the information based upon the parameters of the color camera and the infrared camera.

• Human pixels are depth image pixels associated with the participant, and the Kinect sdk can isolate these pixels by analyzing the depth image.

• Skeleton positions are the skeleton information of the participant from the result of the Kinect sdk's human recognition process.

• Model face parameters are the position, size of the the participant's face. The Microsoft Face Tracking sdk for Kinect enables this real-time face tracking.

After collecting the data, we are able to run different human modeling systems on the same input and compare the test results. The first system is our full algorithm. The second one is our algorithm using manual techniques to mark the real contour instead of automatically generating this contour, so we can see the errors introduced by the matting algorithm. The next system is developed by Kraevoy [48] with our generated contour as the drawing contour. The drawing skill of the user will affect the reliability of the drawing contours and we do not want human factors to affect the evaluation. Therefore, using the same contour can better compare the reshaping algorithm used in [48] and ours.

6.3 Results

None of the evaluation results take the head, hands, and feet into account. The human template used in our system is bald and bare foot, so the generated human model does not match the hair outline and shoes of the participant. The current Kinect SDK does not support finger tracking, so the contour of the participant's fingers cannot match to the corresponding contour path of the template.

6.3.1 Contours

Figure 6.7 shows the real contour (green) and the generated contour (red) of one participant. The black pixels are the overlap part of the real and the generated contours. Real Contour (RC) of the participant in the color image space consists of manually marked pixels, which are on the silhou-

ette of the participant based upon our subjective judgment. After getting the color image of the participant, we manually marked his or her Real Contour for future evaluation purpose. The generated contour is the contour created by our system with the input data from the Kinect. Since there are lots of non-overlapped pixels between the real and the generated contours, the automatically generated contours have errors introduced by depth map input and the matting algorithm.

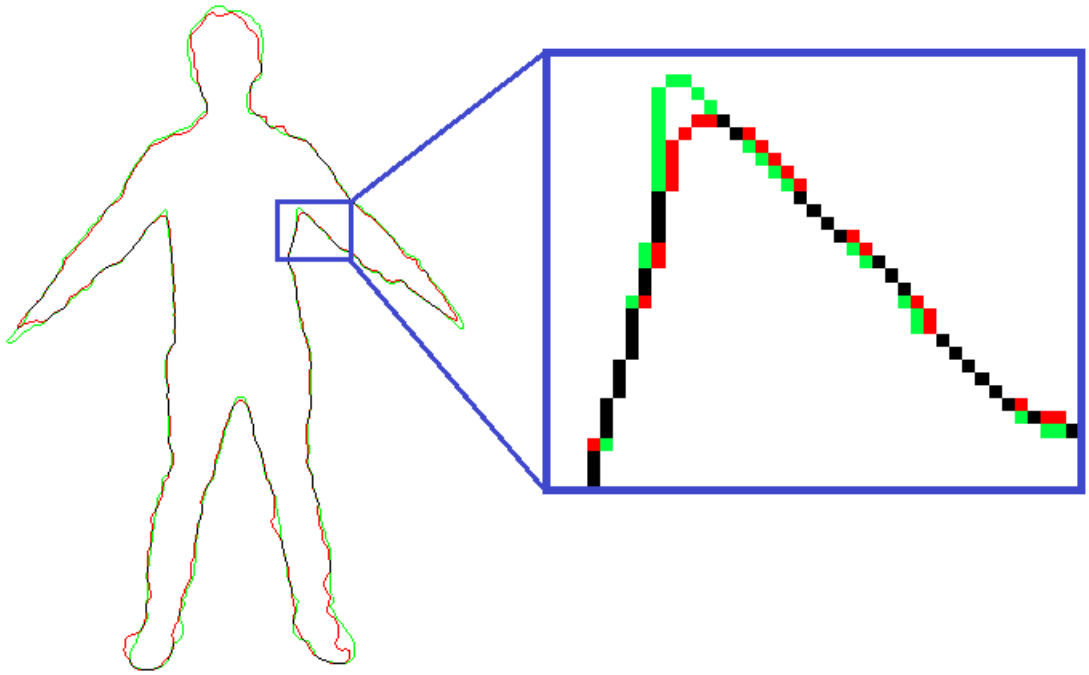

Figure 6.7: Real and generated contours of one participant.

In order to evaluate the error introduced in the contour generation process, suppose there are n pixels on the generated contour. For each pixel p_{gc}^{i} of the generated contour, we find the nearest pixel p_{rc}^{j} on the real contour, which has the minimum Euclidean distance between p_{gc}^{i} and any pixels on the real contour. Then the Euclidean distance d_{gc}^{i} between p_{gc}^{i} and p_{rc}^{j} is the projected distance from p_{gc}^{i} to the real contour, and the difference between the real and the generated contour is the mean value of the project distance $Diff_{gc} = 1/n \sum_{i=1}^{n} p_{gc}^{i}$ of all the pixels on the generated

contour.

Table 6.1: Geometry evaluation.

	Generated Contour
Participant #1	1.2286
Participant #2	1.2836
Participant #3	1.1875
Participant #4	1.2265
Participant #5	1.1576
Participant #6	0.9397
Participant #7	1.1126
Participant #8	1.1085
Participant #9	1.1249
Participant #10	1.1303
Mean	1.1500

Table 6.1 shows the difference between the real and the generated contours of the ten participants. The average difference is 1.1500 pixels, which means that the mean projected distance between the generated contour is 1.1500 pixels away from the real contour. From Figure 6.7, we can see that most of the errors occur around the contour areas having sharp angles and around the head and feet areas. The depth maps are not precise around some small area, like the armpits, and have more errors around the feet, since it is hard to distinguish the floor and the participant from the depth information. The matting algorithm may also introduce some errors, like those seen in the hair part and other noisy areas.

6.3.2 Geometry Evaluation

The errors introduced during the contour extracting process will be propagated and infect the final 3D modeling result. In order to evaluate the geometry of generated human models, we compare the contour of the 3D model with the manually marked real contour. After getting the 3D human

model, we render it to a black background, and then the contour of the model can be extracted easily. The points on the contour are the non-black pixels that have black neighboring pixels. The difference between the 3D model contour and the real contour $Diff_{mc}$ is computed in the same manner as the difference between the generated contour and the real contour. The result represents the pixel distance between the model contour and the real contour.

Table 6.2: Geometry evaluation.

	Xiong's	Kraevoy's	Xiong's + RC
Participant #1	1.2713	1.7150	0.8902
Participant #2	1.3209	1.9263	1.0764
Participant #3	1.2178	1.3592	0.9855
Participant #4	1.2609	1.3334	0.8443
Participant #5	1.6235	1.9352	1.3305
Participant #6	1.3323	1.6202	1.0448
Participant #7	1.3928	1.5060	1.1197
Participant #8	1.4839	1.5271	1.0117
Participant #9	1.4658	1.5473	1.0777
Participant #10	1.4128	2.4855	0.9174
Mean	1.3515	1.6955	1.0298

Table 6.2 shows the comparison for those results produced by our system, Kraevoy's [48] results with our generated contour as the input strokes, and our system with the real contour as input. Our system can generate better 3D human geometry than Kraevoy's with the same contour input, since the introduction of the contour segmentation process can improve the accuracy of the anchors and contour matching result, and then the model deformation is based upon more accurate real world data. When we use the real contour as input, the generated model is the best, because errors are only caused by the anchor matching and model deformation process.

Figure 6.8 is the output model portions of participant #7, and the visual feedback matches the evaluation result. When we use the real contour to generate the model, we get the best result, 1.1197 pixel distance, and the model in image (c) of Figure 6.8 does not include many background

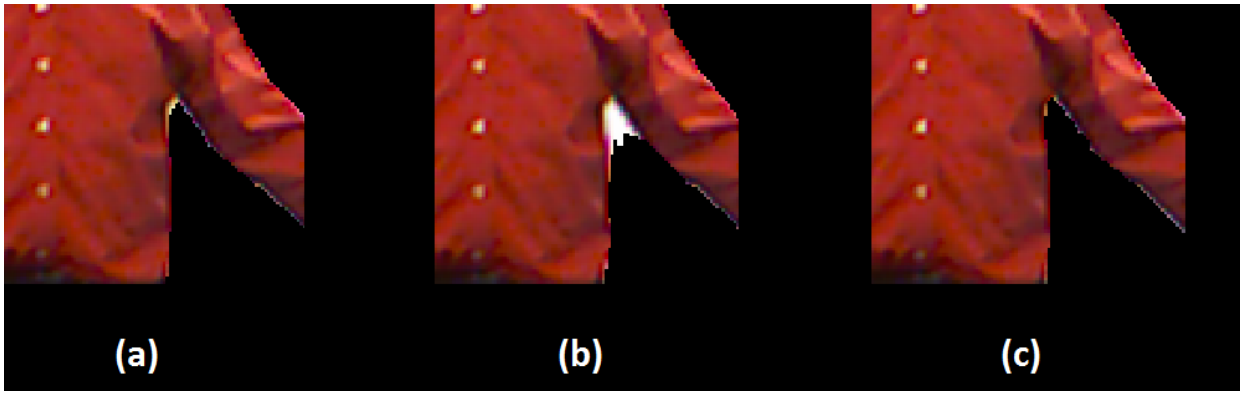

Figure 6.8: An example of generated 3D human model portions with different systems or input: (a) Xiong's system using generated contour without texture retouching process; (b) Kraevoy's system using generated contour; (c) Xiong's system using real contour.

pixels, represented by the whitish pixels around the armpit. The left model in Figure 6.8 is the result of using our full system, and there are more background pixels showing on the model. The worst result in Figure 6.8 (b) is produced with Kraevoy's system. Although the evaluation result, 1.5060 pixels, seems not that bad, the corresponding model is quite different from the real participant around the armpit area.

Evaluating the generated 3D model by comparing contours is able to tell which model is closer to the ground truth. However, the evaluation result is not sufficient to visualize how much difference is introduced. For example, the geometry evaluation result of models in image (a), (b), and (c) of Figure 6.8 are 1.3928, 1.5060, and 1.1197 pixels. The distance between models in (a) and (c) is 0.2731 pixels, while the distance between models in (b) and (C) is 0.3864 pixels. However, the model in image (b) looks much worse than the one in image (a). There are two reasons that the evaluation result value cannot represent the real differences between the models. First of all, we use the minimum distance between the model contour and the real contour as the projected distance, but the real projected distance is much larger than the minimum distance. Take participant #7's

model generated with Kraevoy's system for example (see the zoom in Figure 6.9), the minimum distance is only 6 pixels, but the real projected distance should be 14.8661 pixels. The second reason is that we take the mean distance as the evaluation result. If most part of the generated model A is exactly the same as the real contour and only some small portions are really bad, the mean difference distance is still small. On the other hand, if most part of the generated model B is one or two pixels away from the real contour, this model will be visually closer to the participant, but the evaluation result of this model won't be much better than model A, and may even be worse.

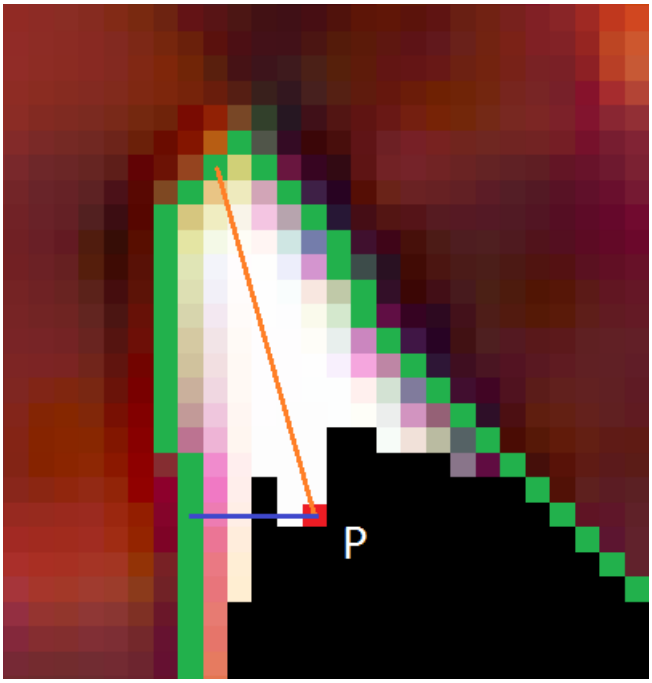

Figure 6.9: Zoom in of participant #7's model generated with Kraevoy's system: green line is the real contour; red pixel P is one pixel on the model contour; blue line shows the minimum distance between P and the real contour; orange line shows the real projected distance.

6.3.3 Texture Evaluation

Since the geometry evaluation is not sufficient to evaluate the visual perception of the output human models, we add the texture evaluation to measure how the model looks with the texture. We use both false positive rate and false negative rate to determine the accuracy of the texture. False positive rate is the number of background pixels on the 3D model divided by the total number of texture pixels n_t, which is the number of pixels inside the real contour. False negative rate is the number of texture pixels not showing on the model divided by n_t. False positive pixels are more visually disturbing than the false negative pixels. Since our final goal is to merge our human modeling system with other Mixed Reality systems, the background of the participant will change, and then the false positive pixels become quite noticeable. On the other hand, the visual result assicated with false negatives causes the texture to be shifted toward the outside, and it is hard for a person to notice that.

Table 6.3: Texture evaluation of our system.

	Xiong's System		
	false positive rate	false negative rate	total false rate
Participant #1	1.7382%	4.6503%	6.3885%
Participant #2	1.8419%	4.2585%	6.1004%
Participant #3	1.6650%	4.1342%	4.1342%
Participant #4	1.8875%	3.8642%	5.7517%
Participant #5	1.7882%	4.9196%	6.7078%
Participant #6	1.7119%	3.9050%	5.6169%
Participant #7	0.5535%	4.8184%	5.3719%
Participant #8	0.7333%	4.7232%	5.4565%
Participant #9	0.7794%	5.5873%	6.3667%
Participant #10	1.8002%	4.3713%	6.1715%
Mean	1.4499	4.5232%	5.8066%

Table 6.3 and Table 6.4 show the texture evaluation result of our system and Kraevoy's system. From the result, we can see that our system cause less false rate, especially false positive rate. The

Table 6.4: Texture evaluation of Kraevoy's system.

	Kraevoy's		
	false positive rate	false negative rate	total false rate
Participant #1	4.8399%	4.5363%	9.3762%
Participant #2	5.3532%	6.0743%	11.4275%
Participant #3	2.2346%	4.5986%	6.8332%
Participant #4	2.0107%	4.3317%	6.3424%
Participant #5	4.8399%	4.5363%	9.3762%
Participant #6	2.5654%	4.7527%	7.3181%
Participant #7	0.6415%	5.2473%	5.8888%
Participant #8	1.2916%	5.9072%	7.1988%
Participant #9	0.9408%	6.4228%	7.3636%
Participant #10	2.8422%	4.4066%	7.2488%
Mean	2.7560%	5.0814%	7.8374%

mean total false rate of our system is 5.8066%, which is better than Kraevoy's 7.8374%. There is little difference in the mean false negative rate between our system (4.5232%) and Kraevoy's (5.0814%). For the first and the fifth participant, Kraevoy's system even produced the model with a little bit lower false negative rate. However, our system is doing much better for reducing the false positives, which is essential to the visual appearance of the generated 3D model. Kraevoy's system works fine for some participants, but not for participants #1, #2, and #5.

Figure 6.10 shows the color image and generated 3D models of the second participant. Most parts of the model created by Kraevoy's system look good; some area are even a little bit better than our system, such as the left armpit. However, the red circled areas are really bad. One portion of the participant's left shoulder is missing, and we can easily notice the sharp hole on that body part. The other problem is from the right armpit to the right hand. The 3D model shifted a lot and the background wall is shown on the human body.

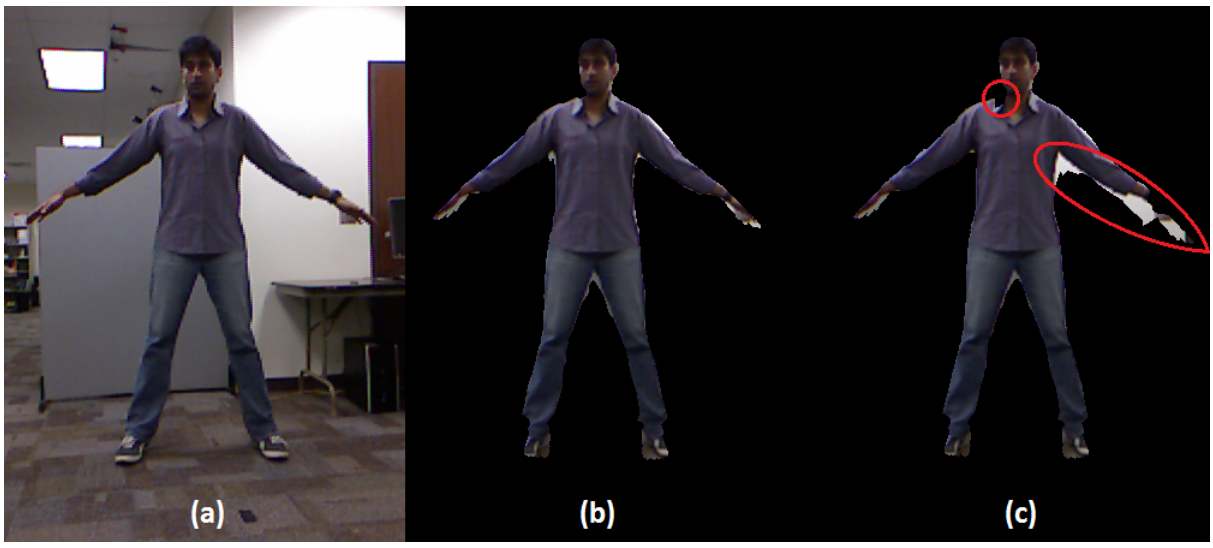

Figure 6.10: Comparing participant #2's model generated with our system and Kraevoy's system: (a) input color image; (b) 3D human model generated with our system; (c) 3D human model generated with Kraevoy's system.

CHAPTER 7: CONCLUSIONS

7.1 Achievements

We have presented a framework for a novel human modeling system that is able to create a 3D human model automatically without any user retouching of the input data. Our system is based upon the 3D model reshaping from 2D contour technique. We use a mean-value geometry encoding algorithm [47] to find the relationship among the vertices in the 3D human template and save the encoding data into a file for the decoding process. Using this approach, we eliminate the encoding time during the human modeling process by only needing to load the encoding information from a pre-saved file. The techniques can generate 3D models rapidly while retaining an initial template's features.

As an initial step in our process, we took advantage of the Microsoft Kinect's ability to automatically extract an approximate contour of the participant instead of our drawing the contour manually. The human tracking result from the Kinect SDK is able to identify when the participant is in our predetermined initial pose and trigger the human modeling process automatically once that pose is matched. We then generate the trimap based upon the depth data and human tracking information. Empolying an automatically created trimap, a matting algorithm is used to get the contour of the participant. This whole pipeline is fully automatic.

A significant achievement in our work is the manner in which our contour segmentation process can improve the matching accuracy between the anchor vertices and the contour points. We use skeleton information from the Kinect SDK and IStraw corner finding algorithm to segment the contour. The IStraw algorithm, which was initially developed for sketch-based interface as an initial step for gesture recognition, works as the basis for our contour segmentation. Our adaptation

of IStraw addresses false negatives and false positives introduced by that algorithm. The resulting segmentation process eliminates many of the mismatches, thereby improving the accuracy of the final 3D model.

In order to evaluate our system, we collected data of ten participants and ran a number of different human modeling systems with this same input data. Since the contour generation process introduces errors, we compared the real contour and the generated contour by computing the mean distance between them. Our primary basis for evaluation is a comparison to Kraevoy's system [48], which we consider the standard among existing systems. In order to not introduce bias, we tested the Kraevoy system with our automatically generated contour path. This differs from the norm used in that approach, which takes a user drawing as its contour input. Furthermore, we compared the 3D model produced from the real contour (ground truth) and the one generated by our algorithm. Our human modeling process with the real contour creates the best 3D human model for all ten participants, and the test results from our full system represents a significant improvement over Kraevoy's. The mean projected distance between the 3D model contour and the real contour is used to compare the output models, as this metric provides reliable, objective results. However, this metric does not address all errors perceived by the human visual system. As such, we introduced another metric to measure the false positive rate and the false negative rate. Our system provides significantly better results on the false positive rate, which is more important to human visual perception of the 3D human model than is the false negative rate.

7.2 Future Plan

Our current system uses only one Kinect to capture the participant's data from one point of view, so the generated 3D model is limited to the front side texture. We can set up a camera to capture the back side of the participant after the initial pose is matched. This image and the Kinect color

image can be appended to form the whole texture map.

The motivation of this human modeling research is to improve the interaction between the real participant and the virtual world. One possible implementation of our system is to improve the sense of presence users feel in collaboration systems. Take the TeachLivE^{TM}project [63] for example; it is a remote teacher training project and the teachers interact with five virtual students, in part controlled by a human-in-the-loop, called the interactor. The teacher is standing in front of a big screen, which shows a projected virtual classroom and the students. The camera of the virtual world is determined by the movement of the teacher. Remotely, the interactor can take direct control of one of the five students based upon the teacher's movement; other virtual characters are controlled through computer agency. When the teacher walks close to one student and looks at him/her, the controller's workstation will present the virtual world as displayed on the teacher's screen, so the interactor knows whom the teacher is addressing and can act through that student's virtual presence, employing the character's unique personality. There is another window on the workstation for the controller to observe the teacher's movements, actions and expressions (body and facial non-verbal messages).

Adding our human modeling technique to TeachLivE^{TM}can help the teacher and the interactor to be more embedded in the virtual world. The teacher is the participant in our modeling system and a 3D model of the teacher can be created at the beginning of running TeachLivE^{TM}. This new setting enables the teacher to interact with the virtual world naturally. On the interactor side, the teacher's 3D model is displayed inside the virtual scene instead of in a separate window and this 3D model is controlled by the teacher's tracking data. Furthermore, the interactor can observe the virtual world from the active student's view point instead of the teachers. This is a more natural view and makes it easier for the interactor or others (e.g., trainers, co-learners or raters) to observe the teacher's behavior.

There are many issues that need to be solved in this new collaboration system design. The most challenging problem is the interaction paradigm between the teacher and the virtual world. If the virtual objects are movable, the teacher should be able to manipulate them. For example, the teacher may show some virtual teaching props (manipulables) to the students. However, the teacher should not be able to move some heavy virtual objects easily, such as the student desks. Moreover, some virtual objects, such as walls, are static obstacles that can never be moved. When there is a conflict between the real and the virtual world, there is a challenge as to how one should display the virtual scene on both the teacher's and the interactor's monitors. We do not want to see the virtual teacher model embedded in a desk or wall on the interactor side. One possible solution is to constrain the movement of the virtual teacher model to be influenced but not absolutely tied to the real teacher, so the model can follow the rules of physics within the virtual world and stand beside, not inside, the virtual desk or wall.

If, on the other hand, we disengage the model from the real person, how do we notify the teacher that his/her avatar in the virtual world is no longer synchronized with him/her? It will be an interesting topic to analyze the different approaches to tell the teacher that his/her movement conflicts with the virtual world. Sometimes the conflict is trivial, such as when there is some minimal intersection between the teacher and the student's desk. The system may not even need to warn the teacher as the disparity is too small to be noticed and is certainly not worth interrupting ongoing teaching tasks. However, when the conflict is critical, the system must make the teacher realize what happened immediately. For example, the teacher's body could go through the virtual student's body. The system may use visual, auditory, and/or haptic feedback, or it might just use a form of redirection, keeping the virtual camera and virtual teacher counterpart always in areas that can be traversed, even if that means an internal recalibration of the mapping between the teacher's position in the real world and that in the virtual setting. This is a topic for further human-centered computing research.

LIST OF REFERENCES

[1] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz, *Keyframe-Based Tracking for Rotoscoping and Animation*. In Proceedings of ACM SIGGRAPH '04, pp. 584-591, 2004

[2] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen, *Interactive Digital Photomontage*. In Proceedings of ACM SIGGRAPH '04, pp. 294-302, 2004

[3] E. de Aguiar, C. Theobalt, M. Magnor, H. Theisel, H.-P. Seidel, *M3: Marker-free Model Reconstruction and Motion Tracking from 3D Voxel Data*. In Pacific Conference on Computer Graphics and Applications 2004, pp. 101-110, Oct. 2004

[4] E. Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H.-P. Seidel, and S. Thrun, *Performance Capture from Sparse Multi-View Video*. In SIGGRAPH '08, pp: 98:1-98:10, 2008

[5] B. Allen, B. Curless, and Z. Popovic *The Space of Human Body Shapes: Reconstruction and Parameterization from Range Scans*. ACM Trans. on Graphics, vol. 22, no. 3, pp. 587-594, 2003

[6] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, *SCAPE: Shape Completion and Animation of People*. In SIGGRAPH '05, pp. 408-416, 2005

[7] N. E. Apostoloff and A.W. Fitzgibbon, *Bayesian Video Matting Using Learnt Image Priors*. In IEEE Conference on Computer Vision and Pattern Recognition 2004, pp. 407-414, 2004

[8] S. Asteriadis, A. Chatzitofis, D. Zarpalas, D. S. Alexiadis, and P. Daras, *Estimating Human Motion from Multiple Kinect Sensors*. In Proceedings of the 6th International Conference on Computer Vision / Computer Graphics Collaboration Techniques and Applications (MIRAGE '13), 2013

[9] X. Bai and G. Sapiro, *A Geodestic Framework for Fast Interactive Image and Video Segmentation and Matting*. In Proceedings of IEEE ICCV '07, pp. 1-8, Oct. 2007

[10] A.O. Balan, L. Sigal, M.J. Black, J.E. Davis, H.W. Haussecker, *Detailed Human Shape and Pose from Images*. In IEEE Conference on Computer Vision and Pattern Recognition 2007 (CVPR '07), pp. 1-8, 2007

[11] A. Balan, and M. Black, *The Naked Truth: Estimating Body Shape Under Clothing*. In ECCV'08, 2008

[12] B.G. Baumgart, *Geometric Modeling for Computer Vision*. PhD dissertation, Stanford University, 1974

[13] N. Beato, R. Pillat, and C. E. Hughes, *Real-time Video Matting for Mixed Reality using Depth Generated Trimaps*. In Proceedings of GRAPP/IVAPP 2012, pp. 280-288, Feb. 2012

[14] N. Beato, *Towards Real-time Mixed Reality Matting in Natural Scenes*. PhD dissertation, University of Central Florida, 2012

[15] T. Beeler, B. Bickel, P. Beardsley, B. Summer, and M. Gross, *High-Quality Single-Shot Capture of Facial Geometry*. In Proceedings of ACM SIGGRAPH 2010, pp. 40:1-40:9, 2010

[16] K. Berger, K. Ruhl, Y. Schroeder, C. Bruemmer, A. Scholz, and M. A. Magnor, *Markerless Motion Capture Using Multiple Color-depth Sensors*. In Proceedings of Vision, Modeling and Visualization 2011 (VMV'11), pp. 317-324, 2011

[17] A. Berman, P. Vlahos, and A. Dadourian, *Comprehensive Method for Removing from an Image the Background Surrounding a Selected Object*. U.S. Patent 6,135,345, 2000.

[18] M.J. Black and P. Anandan, *The Robust Estimation of Multiple Motions: Parametric and Piecewise-smooth flow fields*. Computer Vision and Image Understanding, vol. 64, no. 1, pp. 75-104, 1996

[19] V. Blanz and T. Vetter, *A Morphable Model for the Synthesis of 3D Faces*. In SIGGRAPH '99, pp. 187-194, 1999

[20] G. Borshukov, D. Piponi, O. Larsen, J. P. Lewis, and C. Tempelaar-Lietz, *Universal Capture - Image-Based Facial Animation for "the Matrix Reloaded"*. In SIGGRAPH 2005 courses, 2005

[21] D. Bradley, W. Heidrich, T. Popa, and A. Sheffer, *High Resolution Passive Facial Performance Capture*. In ACM SIGGRAPH 2010, pp. 41:1-41:10, 2010

[22] V. Buchmann, S. Violich, M. Billinghurst, and A. Cockburn, *FingARtips: Gesture Based Direct Manipulation in Augmented Reality*. In Proceedings of the 2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia (GRAPHITE '04), pp. 212-221, 2004

[23] M. Caputo, K. Denker, B. Dums, and G. Umlauf, *3D Hand Gesture Recognition Based on Sensor Fusion of Commodity Hardware*. In Proceedings of Conference Mensch & Computer 2012, pp. 293-302, 2012

[24] L. Chen, H. Lin, and S. Li, *Depth Image Enhancement for Kinect Using Region Growing and Bilateral Filter*. In Proceedings of 21st International Conference on Patter Recognition (ICPR '12), pp. 3070-3073, Nov. 2012

[25] K. Cheung, S. Baker, and T. Kanade, *Shape-From-Silhouette Across Time Part I: Theory and Algorithms*. International Journal of Computer Vision, vol. 62, issue. 3, pp. 221-247, May 2005

[26] J.-H. Cho, R. Ziegler, M. Gross, and K. H. Lee, *Improving Alpha Matte with Depth Information*. IEICE Electronics Express, vol. 6, no. 22, pp. 1602-1607, 2009

[27] J.-H. Cho, T. Yamasaki, K. Aizawa, K. H. Lee, *Depth Video Camera Based Temporal Alpha Matting for Natural 3D Scene Generation*. In Proceedings of IEEE 3DTV-CON, pp. 1-4, 2011

[28] Y. Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski, *A Bayesian Approach to Digital Matting*. In IEEE Conference on Computer Vision and Pattern Recognition, 2001

[29] Y. Y. Chuang, A. Agarwala, B. Curless, D. H. Salesin, and R. Szeliski, *Video Matting of Complex Scenes*. ACM Trans. Graphics, vol. 21, no. 3, pp. 243-348, 2002

[30] Y. Cui, W. Chang, T. Noll, and D. Stricker, *KinectAvatar: Fully Automatic Body Capture Using a Single Kinect*. Computer Vision - ACCV 2012 Workshops, pp. 133-147, November 2012

[31] J. A. Ferwerda, *Three varieties of realism in computer graphics*. In B.E. Rogowitz and T. N. Pappas, editors, Proceedings of Human Vision and Electronic Image VIII, voliume 5007 of SPIE Proceedings Series, pp. 290-297, 2003

[32] E. S. L. Gastal and M. M. Oliveira, *Shared Sampling for Real-Time Alpha Matting*. Computer Graphics Forum, vol. 29, no. 2, pp. 575-584, May. 2010

[33] L. Grady, T. Schiwietz, S. Aharon, and R. Westermann, . In Proc. Fifth Int'l Conf. Visualization, Imaging, and Image Processing (VIIP '05), pp. 423-429, 2005

[34] Y. Guan, X. Liang, Z. Ding, Y. Fan, W. Chen, and Q. Peng, *Energy Matting*. In Edutainment'06, 2006

[35] P. Guan, A. Weiss, A. O. Balan, and M. J. Black, *Estimating Human Shape and Pose from a Single Image*. In IEEE 12th International Conference on Computer Vision (ICCV '09), pp. 1381-1388, 2009

[36] B. Guenter, C. Grimm, D. Wood, H. Malvar, and F. Pighin, *Making Faces*. In Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98), pp. 55-66, 1998

[37] N. Hasler, Bodo. Rosenhahn, T. Thormahlen, M. Wand, Juergen Gall, and H.-P. Seidel, *Markerless Motion Capture with Unsynchronized Moving Cameras*. In IEEE Conference on Computer Vision and Pattern Recognition 2009 (CVPR '09), pp. 224-231, June 2009

[38] K. He, C. Rhemann, C. Rother, X. Tang, and J. Sun, *A Global Sampling Method for Alpha Matting*. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR '11), pp. 2049-2056, 2011

[39] M. Hernandez, J. Choi, and G. Medioni, *Laser Scan Quality 3-D Face Modeling using a Low-Cost Depth Camera*. In 20th European Signal Processing Conference (EUSIPCO '12), pp. 1995-1999, August 2012

[40] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davision, and A. Fitzgibbon, . In ACM Symposium on User Interface Software and Technology, October 2011

[41] A. Jain, T. Thormahlen, H.-P. Seidel, and C. Theobalt, *MovieReshape: Tracking and Reshaping of Humans in Videos*. In ACM SIGGRAPH Asia 2010 (SIGGRAPH ASIA '10), pp. 148:1-148:9, 2010

[42] J. Jia, J. Sun, C.-K. Tang, and H.-Y. Shum, *Drag-and-Drop Pasting*. In Proceedings of ACM SIGGRAPH '06, pp. 631-637, 2006

[43] N. Joshi, W. Matusik, and S. Avidan, *Natural Video Matting using Camera Arrays*. In Proceedings of ACM SIGGRAPH, pp. 779-786, 2006

[44] I. Kemelmacher-Shlizerman and R. Basri, *3D Face Reconstruction from a Single Image Using a Single Reference Face Shape*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 2, pp. 394-405, Feb. 2011

[45] I. Kakadiaris and D.Metaxas, *3D Human model Acquisition from Multiple Views*. In Fifth International Conference on Computer Vision (ICCV '95), pp. 618-623, Jun. 1995

[46] S.-Y. Kim, J.-H. Cho, and A. Koschan, *3D Video Generation and Service based on a TOP Depth Sensor in MPEG-4 Multimedia Framework*. In IEEE Transactions on Consumer Electronics, vol. 56, no. 3, pp. 1730-1738, Aug. 2010

[47] V. Kraevoy, and A. Sheffer, *Mean-Value Geometry Encoding*. Intl. Journal of Shape Modeling, vol 12, issue 1, pp. 29-46, 2006

[48] V. Kraevoy, A. Sheffer, and M. Van De Panne, *Modeling from Contour Drawings*. In SBIM '09, pp. 37-44, 2009

[49] A. Levin, A. Rav-Acha, and D. Lischinski, *Spectral Matting*. In Proceedings of IEEE CVPR '07, pp. 1-8, Jun. 2007

[50] A. Levin, D. Lischinski, and Y. Weiss, *A Closed Form Solution to Natural Image Matting*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 2, pp. 228-242, Feb. 2008

[51] Y. Li, J. Sun, and H.-Y. Shum, *Video Object Cut and Paste*. In Proceedings of ACM SIGGRAPH '05, pp. 595-600, 2005

[52] H. Li, E. Vouga, A. Gudym, L. Luo, J. T. Barron, and G. Gusev, *3D Self-Portraits*. In Proceedings of ACM SIGGRAPH '05, pp. 595-600, 2005

[53] I.-C. Lin, and M. Ouhyoung, *Mirror MoCap: Automatic and Efficient Capture of Dense 3D Facial Motion Parameters from Video*. The Visual Computer, vol. 21, no. 6, pp. 355-372, 2005

[54] P. Lindstrom, and G. Turk, *Fast and Memory Efficient Polygonal Simplification*. In SIGGRAPH Asia 2013, 2013

[55] T. Lu and S. Li, *Image Matting with Color and Depth Information*. In Proceedings of 21st International Conference on Patter Recognition (ICPR '12), pp. 3787-3790, Nov. 2012

[56] W.-C. Ma, T. Hawkins, P. Peers, C.-F. Chabert, M. Weiss, and P. Debevec, *Rapid Acquisition of Specular and Diffuse Normal Maps from Polarized Spherical Gradient Illumination*. In EUROGRAPHICS Symposium on Rendering, 2007

[57] W.-C. Ma, A. Jones, J.-Y. Chiang, T. Hawkins, S. Frederiksen, P. Peers, M. Vukovic, M. Ouhyoung, and P. Debevec, *Facial Performance Synthesis Using Deformation-Driven Polynomial Displacement Maps*. In ACM SIGGRAPH Asia 2008 (SIGGRAPH Asia '08), pp. 121:1-121:10, 2008

[58] M. McGuire, W. Matusik, H. Pfister, J. F. Hughes, and F. Durand, *Defocus Video Matting*. In Proceedings of ACM SIGGRAPH '05, pp. 567-576, 2005

[59] P. Milgram, and F. Kishino, *A Taxonomy of Mixed Reality Visual Display*. IEICE Transactions on Information Systems, vol. E77-D, no. 12, Dec. 1994

[60] Y. Mishima, *Soft Edge Chroma-Key Generation Based upon Hexoctahedral Color Space*. U.S. Patent 5,355,174, 1993.

[61] T. B. Moeslund and E. Granum, *A Survey of Computer Vision-Based Human Motion Capture*. Computer Vision and Image Understanding, vol. 81, no. 3, pp. 231-268, Mar. 2001

[62] T. B. Moeslund, A. Hilton, and V. Kruger, *A Survey of Advances in Vision-Based Human Motion Capture and Analysis*. Computer Vision and Image Understanding, vol. 104, no. 2-3, pp. 90-126, Nov.-Dec. 2006

[63] A. Nagendran, R. Pillat, A. Kavanaugh, G. F. Welch, and C. E. Hughes, *AMITIES: Avatar-mediated Interactive Training and Individualized Experience System*. In Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology, pp. 143-152, Oct. 2013

[64] F.I. Parke, *Computer Generated Animation of Faces*. In ACM SIGGRAPH '72, pp. 451-457, 1972

[65] F. Pighin, R. Szeliski, and D. Salesin, *Modeling and Animating Realistic Faces from Images*. Int'l J. Computer Vision, vol. 50, no. 2, pp. 143-169, 2002

[66] R. Plankers and P. Fua, *Articulated Soft Objects for Video-Based Body Modeling*. In Eighth IEEE International Conference on Computer Vision (ICCV '01), pp. 394-401, 2001

[67] T. Porter and T. Duff, *Compositing Digital Images*. In SIGGRAPH 1984, pp. 253-259, Jul. 1984.

[68] R.J. Qian and M.I. Sezan, *Video Background Replacement Without a Blue Screen*. In ICIP 1999, pp. 143-146, Oct. 1999

[69] B. Robertson, *Facing the Future*. Computer Graphics World, vol. 36, issue 6, Oct. 2013

[70] C. Rocchini, P. Cignoni, C. Montani, P. Pingi, and R. Scopigno, *A Low Cost 3D Scanner based on Structured Light*. Computer Graphics Forum, vol. 20, issue 3, pp. 299-308, September 2001

[71] K. Rohr, *Human Movement Analysis Based on Explicit Motion Models*. Motion-Based Recognition, M. Shah and R. Jain (Eds), chap. 8, pp. 171-198, 1997

[72] M. A. Ruzon and C. Tomasi, *Alpha Estimation in Natural Image*. In Proc. IEEE Conference on Computer Vision and Pattern Recognition 2000, pp. 18-25, Jun. 2000

[73] Y. Schroder, A. Scholz, K. Berger, K. Ruhl, S. Guthe, and M. Magnor, *Multiple Kinect Studies*. Technical Report no. 09-15, ICG, Oct. 2011

[74] T. Sezgin, T. Stahovich, and R. Davis, *Sketch based Interface: Early Processing for Sketch Understanding*. In Workshop on Perceptive User Interface, 2001

[75] T. Sezgin, and R. Devis, *Scale-Space Based Feature Point Detection for Digital Ink*. In SIGGRAPH '06 Courses, pp. 29, 2006

[76] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, *Real-Time Human Pose Recognition in Parts from Single Depth Images*. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR '11), pp. 1297-1304, 2011

[77] C. Sminchisescu and A. Telea, *Human Pose Estimation from Silhouettes. A Consistent Approach Using Distance Level Sets*. In 10th International Conference on Computer Graphics, Visualization and Computer Vision (WSCG '02), 2002

[78] A.R. Smith and J.F. Blinn, *Blue Screen Matting*. In SIGGRAPH '96, pp. 259-268, Aug. 1996

[79] J. Sun, J. Jia, C.-K. Tang, and H.-Y. Shum, *Poisson Matting*. In Proceedings of ACM SIGGRAPH '04, pp. 315-321, 2004

[80] J. Sun, Y. Li, S.-B. Kang, and H.-Y. Shum, *Flash matting*. In Proceedings of ACM SIGGRAPH '06, pp. 772-778, 2006

[81] N. Magnenat-Thalmann, and D. Thalmann, *The Direction of Synthetic Actors in the Film Rendez-vous a Montreal*. IEEE Computer Graphics and Applications, vol. 7, no. 12, 1987, pp. 9-19, 1987

[82] J. Tong, J. Zhou, L. Lin, Z. Pan, and H. Yan *Scanning 3D Full Human Bodies Using Kinects*. Visualization and Computer Graphics, vol. 18, no. 4, pp. 643-650, 2012

[83] D. Vlasic, I. Baran, W. Matusik, and J. Popovic, *Articulated Mesh Animation from Multi-view Silhouettes*. In Proceedings of ACM SIGGRAPH 2008, pp. 97:1-97:10, 2008

[84] J. Wang and M. Cohen, *An Iterative Optimization Approach for Unified Image Segmentation and Matting*. In Proc. 10th IEEE Int'l Conf. Computer Vision (ICCV '05), vol. 2, pp. 936-943, Oct. 2005.

[85] J. Wang, P. Bhat, A. Colburn, M. Agrawala, and M. Cohen, *Interactive video cutout*. In Proceedings of ACM SIGGRAPH '05, pp. 585-594, 2005

[86] J. Wang and M.F. Cohen, *Optimized Color Sampling for Robust Matting*. In Proceedings of IEEE CVPR '07, pp. 1-8, Jun. 2007

[87] J. Wang, M. Agrawala, and M.F. Cohen, *Soft Scissors: an Interactive Tool for Realtime High Quality Matting*. In SIGGRAPH 2007, pp. 9:1-9:6, 2007

[88] J. Wang and M. Cohen, *Simultaneous Matting and Compositing*. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR '07), pp. 1-8 2007

[89] J. Wang and M.F. Cohen, *Image and Video Matting: a Survey*. Foundation and Trends in Computer Graphics and Vision, vol. 3, no. 2, pp. 97-175, 2008

[90] O. Wang, J. Finger, and Q. Yang, *Automatic Natural Video Matting with Depth*. In Proceedings of 15th Pacific Conference on Computer Graphics and Applications, pp. 469-472, 2007

[91] T. Weise, S. Bouaziz, H. Li, and M. Pauly, *Realtime Performance-Based Facial Animation*. In SIGGRAPH '11, pp. 77:1-77:10, 2011

[92] A. Weiss, D. Hirshberg, and M. J. Black, *Home 3D Body Scans from Noisy Image and Range Data*. In 13th International Conference on Computer Vision, pp. 1951-1958, 2011

[93] A. Wolin, B. Eoff, and T. Hammond, *ShortStraw: A Simple and Effective Corner Finder for Polylines*. In EUROGRAPHICS Fifth Annual Workshop on Sketch-based Interfaces and Modeling (SBIM'08), pp. 33-40, 2008

[94] A. Wolin, B. Paulson, and T. Hammond, *Sort, Merge, Repeat: An Algorithm for Effectively Finding Corners in Hand-Sketched Strokes*. In Proceedings of the sixth EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling, SBIM '09, pp. 93-99, 2009

[95] Y. Xiong, and J. LaViola, *Revisiting ShortStraw - Improving Corner Finding in Sketch-Based Interfaces*. In Proceedings of EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling, SBIM '09, pp. 101-108, 2009

[96] Y. Xiong, and J. LaViola, *A ShortStraw-based Algorithm for Corner Finding in Sketch-based Interfaces*. Computers and Graphics, vol. 34, no. 5, pp. 513-527, 2010

[97] J. Wobbrock, A. Wilson, and Y. Li, *Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes*. In Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04), pp. 159-168, 2004

[98] P. Vlahos, *Electroic Composite Photography*. U.S. Patent 3,595,987, July 27, 1971, Expired.

[99] P. Vlahos *Comprehensive Electronic Compositing System*. U.S. Patent 4,100,569, July 11, 1978, Expired.

[100] G. Ye, Y. Liu, N. Hasler, X. Ji, Q. Dai, and C. Theobalt, *Performance Capture of Interacting Characters with Handheld Kinects*. In Proceedings of the 12th European Conference on Computer Vision (ECCV), 2012

[101] S. You, U. Neumann, and R. Azuma, *Hybrid Inertial and Vision Tracking for Augmented Reality Registration*. In Proceedings of IEEE Virtual Reality, pp. 260-267, Mar. 1999

[102] L. Zhang, N. Snavely, B. Curless, and S.M. Seitz *Spacetime faces: High-Resolution Capture for Modeling and Animation*. ACM Transactions on Graphics, vol. 23, no. 3, pp. 548-558, 2004

[103] S. Zhang, and P. Huang, *High-Resolution, Real-Time 3D Shape Acquisition*. In Conference on Computer Vision and Pattern Recognition Workshop (CVPR '04), Jun. 2004

[104] Y. Zheng, C. Kambhamettu, J. Yu, T. Bauer, and K. Steiner, *Fuzzymatte: A Computationally Efficient Scheme for Interactive Matting*. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition 2008 (CVPR '08), pp. 1-8, Jun. 2008

[105] S. Zhou, H. Fu, L. Liu, D. Cohen, and X. Han, *Parametric Reshaping of Human Bodies in Images*. In SIGGRAPH '10, 2012

[106] J. Zhu, M. Liao, R. Yang, and Z. Pan, *Joint Depth and Alpha Matte Optimization via Fusion of Stereo and Time-of-Flight Sensor*. In Proceedings of IEEE CVPR '09, pp. 453-460, 2009

[107] M. Zollhofer, M. Martinek, G. Greiner, M. Stamminger, and J. Sbmuth, *Automatic Reconstruction of Personalized Avatars from 3D Face Scans*. Computer Animation and Virtual Worlds, vol. 22, issue 2-3, pp. 195-202, 2011