3D FACE MODELING WITH A CONSUMER DEPTH CAMERA

BY

GREGORY PAUL MEYER

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Adviser:

Associate Professor Minh N. Do
ABSTRACT

Modeling the 3D geometry of the face is an important research topic in computer graphics and computer vision. The applications for face models include computer animation and facial analysis. Over the past few decades, several face modeling techniques have been developed using both active and passive sensors. Unlike passive sensors, which reconstruct the geometry of the face using reflected ambient light, active sensors measure the geometry of the face by emitting an external light source onto the surface of the face. As a result, techniques that utilize active sensors often produce higher quality face models, but the active sensors are typically expensive. Recently, low-cost consumer depth cameras have become widely available due to the success of Microsoft’s Kinect camera. The Kinect is an active sensor that provides depth images at video rate; however, the images are often noisy and missing measurements.

In this thesis, we present a method for modeling the geometry of the face using a consumer depth camera. To construct a high quality model, we combine the surface measurements from multiple depth images. By registering and integrating a sequence of depth images, we are able to recover the entire surface of the face and reduce noise. We demonstrate that 3D face models built using our proposed method are comparable to models generated using an expensive, high-resolution 3D scanner.
To my mother
I would like to express my gratitude to my advisor Minh Do for his guidance and support of my research. I would also like to recognize my colleagues Huy Bui, Ben Chidester, and Trong Nguyen for allowing me to use their images in my research. I greatly appreciate the help and equipment provided by the Imaging Technology Group. Finally, I would like to thank my family and friends for all their love and encouragement.
# TABLE OF CONTENTS

| CHAPTER 1  | INTRODUCTION  | 1 |
| 1.1  | Motivation  | 1 |
| 1.2  | Proposed Method  | 2 |
| 1.3  | Thesis Summary  | 3 |

| CHAPTER 2  | BACKGROUND  | 4 |
| 2.1  | Passive Modeling Techniques  | 4 |
| 2.2  | Active Modeling Techniques  | 5 |

| CHAPTER 3  | FACE DETECTION AND SEGMENTATION  | 7 |
| 3.1  | Face Detection  | 7 |
| 3.1.1  | Accelerating Face Detection  | 8 |
| 3.1.2  | Face Detection Performance  | 13 |
| 3.2  | Face Segmentation  | 15 |

| CHAPTER 4  | FACE REGISTRATION AND INTEGRATION  | 18 |
| 4.1  | Surface Alignment  | 18 |
| 4.1.1  | Noise Reduction  | 19 |
| 4.1.2  | Point-Cloud Generation  | 20 |
| 4.1.3  | Iterative Closest Point  | 21 |
| 4.2  | Surface Modeling  | 27 |
| 4.2.1  | Volumetric Integration  | 27 |
| 4.2.2  | Surface Extraction  | 31 |

| CHAPTER 5  | EXPERIMENTAL RESULTS  | 36 |
| 5.1  | Runtime  | 36 |
| 5.2  | Quality  | 37 |

| CHAPTER 6  | CONCLUSION  | 39 |

| REFERENCES  | 40 |
1.1 Motivation

3D face modeling is the process of modeling the geometry of a human face. The ability to generate a 3D model of a person’s face has several uses in computer graphics and computer vision. For computer graphics, 3D face models can be used to construct a personalized avatar for multimedia applications. For computer vision, 3D face models can be used for facial recognition and face analysis.

There are two primary approaches to construct 3D models of human faces. One approach uses active sensors to capture the geometry of a face, and the other uses passive sensors to reconstruct the 3D face model from one or more 2D images [1]. Active sensors emit an external light onto the face being modeled. The light can be a low-intensity laser beam or a structured light projection. The deformations of the applied light are measured to determine the 3D geometry of the face [2]. Passive sensors are ordinary digital cameras that capture reflected ambient light [1]. Multiple 2D images are used to reconstruct the geometry of the face. Techniques that use active sensors typically produce high quality 3D models; however, the sensors are expensive and require a controlled environment. Passive sensors are inexpensive and work under a variety of lighting conditions, but reconstructing the surface from 2D images can produce low quality models [3].

In the past, using an active sensor was restrictive due to its cost. With the success of Microsoft’s Kinect, consumer depth cameras are now widely available. The Kinect is a low-cost active sensor that provides depth images of the scene at video rate. Each depth image consists of a set of pixels, where each pixel contains a depth measurement. Unfortunately, the images are often noisy and contain holes. To reduce noise and fill in holes, a sequence
of depth images can be combined into a 3D model [4].

1.2 Proposed Method

We propose a method for generating a 3D face model using a consumer depth camera. Our system builds a high quality model by incrementally registering and integrating a sequence of depth images in real time. With our method, we enable a person to construct a model of his or her face by simply moving his or her head in front of a depth camera.

In order to simplify the problem, we assume the user’s head and upper torso is visible to the camera, his or her facial expression remains constant throughout the sequence, and the relative motion of his or her head between images is small.

Our system begins by acquiring a depth and color image from a Kinect camera. We use both images to detect and segment the person’s face. We filter the segmented depth image to reduce noise, and we convert the depth measurements into a 3D point cloud. The point clouds are registered using iterative closest point, and they are fused using volumetric integration. Afterwards, we extract the current state of the surface using ray casting and display it to the user. An overview of our system is depicted in Figure 1.1.

![Figure 1.1: An overview of our proposed method.](image)
1.3 Thesis Summary

The remainder of this thesis is organized as follows. In Chapter 2, we discuss a variety of active and passive face modeling techniques that have been proposed over the last few decades, including a few methods that utilize low-cost depth cameras.

Chapter 3 describes our method for detecting and segmenting a person’s face using both depth and color images. We detect the person’s face in the color image using a face detector, and we use the depth information to accelerate the face detection process. Once we know the location of the person’s face, we can remove all depth measurements that do not lie on the face.

Chapter 4 explains the process of registering and integrating the depth images. Registration involves filtering the depth images to reduce noise, converting the depth images to point clouds, and using iterative closest point to compute the rigid body transformation that aligns the point clouds. Integration is performed by incrementally fusing the depth images into a volumetric model. Ray casting is utilized to extract the current state of the volumetric model, and marching cubes is used to convert the volumetric model to a triangular mesh.

In Chapter 5 we discuss our experimental results. We analyze the efficiency of our algorithm, and the quality of our face model. Concluding remarks are given in Chapter 6.
CHAPTER 2

BACKGROUND

3D face modeling has been an active research topic for several decades. A large variety of methods utilizing both active and passive sensors have been developed.

2.1 Passive Modeling Techniques

Passive modeling techniques, such as stereo vision, structure from motion, and morphing, use one or more images to reconstruct a 3D model of a person’s face [2]. The input for these methods could be a single image, a pair of images captured simultaneously by two cameras, or a sequence of images captured by one camera.

Stereo vision reconstructs the 3D surface of a scene using two 2D images. The two images are captured simultaneously by two cameras at slightly different viewpoints [2]. To construct a 3D model, corresponding points must be found in each image. Texture or gradient information is typically used to locate correspondences between the images. Triangulation of corresponding points is used to reconstruct the surface [2]. Finding corresponding points in face images can be challenging due to the lack of texture information in the cheeks, forehead, etc. R. Lengagne et al. [5] uses a priori knowledge of human faces to improve stereo reconstruction. Their system fits a 3D mesh to the reconstructed surface. In areas where the depth information is unreliable, they enforce geometric constraints on the mesh to more accurately model the shape of a human face.

Structure from motion (SfM) uses a sequence of images captured by a single camera to reconstruct the surface. To extract 3D information from the video sequence, features are detected and tracked throughout the frames. The motion of the features is used to determine the motion of the camera, as
well as, the 3D position of the features [2]. 3D face models generated by a SfM algorithm can be low quality. A. Chowdhury et al. [6] use a generic face model to reduce noise in the reconstructed model. The 3D estimates generated by the SfM algorithm are compared to the generic model. When there is a large difference between the reconstructed model and the generic model, the depth measurements around the error are smoothed.

Morphing techniques can generate a face model from a single image. Morphing requires a generic shape of a human face, called a morphable model, to be created beforehand. The morphable model can be obtained by averaging several face models generated by an active sensor [2]. A 3D face model of a specific person’s face can be produced by morphing the generic shape, such that the 2D projections of the morphed shape approximate the 2D image [2]. V. Blanz and T. Vetter [7] constructed a morphable model from 200 face models captured by a laser scanner. The example faces are used to define a vector space, and a new face model can be constructed from a linear combination of these examples. Given a 2D image of a face, a new 3D face model can be built by determining the coefficients of the linear combination that minimize the difference between the image of the face and the 2D projection of the model [7].

2.2 Active Modeling Techniques

Active modeling techniques typically use laser range scanning or structured light projection.

Laser range scanners emit a laser line or spot onto a surface. The laser is moved over the object to create a sequence of images, and from these images, triangulation is used to determine the 3D geometry of the object [2]. L. Chen et al. [8] developed a semi-automatic method to adapt a generic head model to 3D range data captured by a laser scanner. Their method required user input to align the generic model to the range data. By automatically detecting facial features in the range image, Y. Lee et al. [9] were able to fit a generic face mesh to the range data without manually aligning the models.

In a structured light system, a pattern of light is projected onto the object. A camera captures the illuminated surface, and uses the deformation of the light pattern to determine the depth of the object [2]. C. Beumier and M.
Acheroy [10] used structured light projection to generate face models for face recognition. Their system used a standard projector to emit strips of light with varying thicknesses onto a person’s face, and a digital camera is used to capture the deformed light pattern. They matched the strips in the image to the projected pattern and utilized triangulation to estimate the surface.

Recently, low-cost active sensors have become available in the form of consumer depth cameras, such as Microsoft’s Kinect. The Kinect consists of an infrared projector, an infrared camera, and a color camera. The infrared projector emits an invisible pattern onto the scene, and the infrared camera captures the deformed pattern. The Kinect uses the deformations of the structured light projection to measure depth [11]. There have been a few proposed methods that use a Kinect camera to generate 3D face models [12, 13].

M. Hernandez et al. [12] proposed a system that generates a 3D face model from a sequence of depth images. They capture depth images from the Kinect camera and use a face detection algorithm to segment the user’s head from the depth image. The segmented depth images are registered and integrated into a 3D model. Their method accumulates the registered depth images in an unwrapped cylindrical 2D image, which allows them to use 2D spatial filters to remove noise.

M. Zollhöfer et al. [13] use the Kinect camera’s color sensor to detect facial features such as the eyes and nose. They use these features to align a generic face model to a depth image. Afterwards, they deform the generic model to fit the depth image.
CHAPTER 3

FACE DETECTION AND SEGMENTATION

The depth images acquired from the camera contain depth measurements for the entire scene within view of the sensor. Since we are modeling the geometry of a person’s face, we are only interested in the pixels that lie on the face. Consequently, we must detect and segment the user’s head from the depth images.

3.1 Face Detection

The Kinect captures both depth and color images (Figure 3.1). We can leverage the color information to identify faces within the scene. Detecting faces is a preprocessing step for many vision applications. In addition to face modeling, applications that typically require face detection include face analysis and biometrics, face recognition, human-computer interaction, and surveillance [14]. As a result, there are effective techniques, such as the Viola-Jones method [15], for detecting faces.

Our objective is to model the surface of the face in real time. Consequently, we require a face detection algorithm that can locate faces faster than real time in order to have enough post-processing time to reconstruct the face.

In most situations, the size of a face is unknown; therefore, a face detector must examine different sizes of sub-images at every position within the image. Although the Viola-Jones face detector was designed to efficiently classify sub-images [15], the exhaustive search over the entire image at multiple scales is computationally expensive.

If we were able to remove sub-images that are unlikely to contain faces, we could accelerate the face detection process. Based on the depth information, we can approximate the size of each sub-image, and remove those that are much smaller or larger than an average person’s face. In addition, we
Figure 3.1: Color and depth image captured by a Kinect camera. Each pixel in the color image (a) contains the amount of light reflected off the surface, and each pixel in the depth image (b) contains the distance between the camera and the surface.

We eliminate all sub-images whose boundaries are not aligned to depth discontinuities. These modifications to the Viola-Jones face detector significantly reduce the time to detect faces, as well as decrease false detections.

3.1.1 Accelerating Face Detection

We use the depth information to identify a set of sub-images that may contain faces. We begin by approximating the size of a face at every pixel location. Afterwards, we find regions that may contain faces based on depth boundaries. Combining this information, we generate a set of sub-images to be classified by the Viola-Jones face detection algorithm.

**Estimating face dimensions**

If we assume each pixel lies on a face, we can use the pixel’s depth measurement to estimate a range of possible face dimensions. For the \((i,j)\)th pixel in the image, we compute the minimum and maximum widths of the sub-images,

\[
w_{i,j}^{\text{min}} = \frac{f \cdot W_{\text{min}}}{Z_{i,j}}, \quad w_{i,j}^{\text{max}} = \frac{f \cdot W_{\text{max}}}{Z_{i,j}}
\]

where \(f\) is the depth sensor’s focal length, \(Z_{i,j}\) is the pixel’s depth measurement in millimeters, and \(W_{\text{min}}\) and \(W_{\text{max}}\) are our approximate minimum and maximum sizes for a human face in millimeters.
By constraining our search to only sub-images that are within this range, we eliminate all sub-images that are much smaller or larger than the average person’s face. However, many of these sub-images will not contain faces.

Identifying possible face regions

We can further limit our search by determining which pixels are likely to be on a face. Usually, there is a depth discontinuity between a person’s head and the background; therefore, a sub-image will not contain a face if its edges are not near depth boundaries. For each pixel, we have a range of widths for a set of sub-images. If there is not a depth boundary near any of these sub-images, we conclude that the pixel is not on a face, and all sub-images centered on this pixel are discarded.

We identify the boundaries within the depth image by comparing neighboring depth measurements. The \((i, j)^{th}\) pixel is labeled as a boundary if the difference between its depth measurement and a neighboring pixel’s depth measurement is greater than a threshold,

\[
 b_{i,j} = \begin{cases} 
 1, & \text{if } |Z_{i,j} - Z_{k,l}| > \theta_{\text{max}} \\ 
 0, & \text{otherwise} 
\end{cases} 
\]  

(3.2)

where \((k,l) \in \{(i-1,j), (i,j-1)\}\), and \(\theta_{\text{max}}\) is the difference threshold. To rapidly evaluate the distance from any point to the nearest depth discontinuity, we perform a two-pass distance transformation on the boundary map [16]. The two-pass distance transformation is split into a forward pass and a backward pass, and the initial distance at each pixel is set to the following:

\[
 d_{i,j} = \begin{cases} 
 0, & \text{if } b_{i,j} = 1 \\ 
 \infty, & \text{otherwise.} 
\end{cases} 
\]  

(3.3)

During the forward pass, the distances are updated from left to right and top to bottom. The distance at the \((i, j)^{th}\) pixel is changed based on its neighboring pixels to its left and above,

\[
 d_{i,j} \leftarrow \min_{(k,l) \in \mathcal{N}_1} d_{k,l} + \left[ \sqrt{(k-i)^2 + (l-j)^2} \right] 
\]  

(3.4)

9
Figure 3.2: Illustration of the two-pass distance transformation of the image (a). During the forward pass (b) the distances are updated pixel by pixel starting with the top-left corner and proceeding from left to right and top to bottom. During the backward pass (c) the distances are updated starting with the bottom-right corner and proceeding from right to left and bottom to top.

\[
\begin{array}{ccc}
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\end{array}
\quad
\begin{array}{ccc}
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\infty & \infty & \infty \\
\end{array}
\quad
\begin{array}{ccc}
3 & 2 & 1 \\
3 & 2 & 1 \\
3 & 2 & 1 \\
2 & 2 & 1 \\
2 & 1 & 1 \\
1 & 1 & 1 \\
1 & 0 & 1 \\
1 & 0 & 1 \\
1 & 0 & 1 \\
\end{array}
\]

(a) Initial distances (b) Forward pass (c) Backward pass

where \( N_1 = \{ (i, j), (i - 1, j), (i + 1, j - 1), (i, j - 1), (i - 1, j - 1) \} \). During the backward pass, the distances are updated from right to left and bottom to top. Similarly, the distance at the \( (i, j)^{th} \) pixel is changed based on its neighboring pixels to its right and below,

\[
d_{i,j} \leftarrow \min_{(k,l) \in N_2} \left[ d_{k,l} + \sqrt{(k-i)^2 + (l-j)^2} \right] \tag{3.5}
\]

where \( N_2 = \{ (i, j), (i + 1, j), (i - 1, j + 1), (i, j + 1), (i + 1, j + 1) \} \). An illustration of the two-pass distance transformation is shown in Figure 3.2.

After executing the two passes, we have an approximation of the distance from each pixel to the nearest depth discontinuity. The boundary map and

Figure 3.3: Depth boundary map (a) and corresponding distance transformation (b). The distance to a depth boundary is used to identify possible face regions.
Figure 3.4: The mask of possible face regions (b) is constructed by examining sub-images (a) centered at each pixel. Three points (crosses in (a)) along the mean sub-image’s border (dash line in (a)) are checked to determine if the pixel is likely to be on a face based on the distance from the point to a depth boundary.

distance transform, for the depth image in Figure 3.1b, are shown in Figure 3.3.

To ensure that a pixel could lie on a face, we examine three points along the border of the mean sub-image. The mean sub-image at the \((i, j)^{th}\) pixel has the width,

\[
w_{avg}^{i,j} = \frac{w_{min}^{i,j} + w_{max}^{i,j}}{2}
\]

(3.6)

and we inspect a point centered on its right \((i + w_{avg}^{i,j} / 2, j)\), left \((i - w_{avg}^{i,j} / 2, j)\), and top \((i, j - w_{avg}^{i,j} / 2)\) edges, as shown in Figure 3.4a. If the pixel is on a face, then these three points should be near a depth discontinuity. For that reason, if the distance to the nearest depth boundary, at each of the three points, is less than \(w_{max}^{i,j} - w_{avg}^{i,j}\), then the \((i, j)^{th}\) pixel may lie on a face. We construct a mask of all possible face pixels (Figure 3.4b), and for each pixel within the mask, we pass all of its sub-images to the face detector to be classified.

Figure 3.5 shows the possible face regions, as well as the estimated face dimensions. We are able to remove a large part of the image that does not contain faces.
Figure 3.5: Each pixel in (b) and (d), which is in a possible face region, contains an approximate width for a face in the corresponding color images, (a) and (c). These results are used to determine which sub-images are classified by the face detector.
3.1.2 Face Detection Performance

We compared the runtime and accuracy of our accelerated face detector method against OpenCV’s standard Viola-Jones face detector. We experimented with several videos captured by the Kinect. Each sequence contains a variety of subjects at different distances from the camera.

We profiled both methods on an Intel Core i7 CPU, and the average runtime per frame across all the test sequences is listed in Table 3.1. The per frame overhead for our method (which is included in the runtime) is only 1.86 milliseconds. For both VGA (640×480) and QVGA (320×240) image resolutions, we achieve a 2.5 times speedup over the standard Viola-Jones face detector. Although both methods are fairly efficient, our method allows for additional post-processing, which is essential for modeling the face in real time.

Table 3.1: Average runtime per frame.

<table>
<thead>
<tr>
<th>Image Resolution</th>
<th>Viola-Jones (ms)</th>
<th>Our Method (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>640×480</td>
<td>100.74</td>
<td>39.62</td>
</tr>
<tr>
<td>320×240</td>
<td>32.64</td>
<td>13.01</td>
</tr>
</tbody>
</table>

Additionally, we compared the accuracy of both methods. Table 3.2 lists the percentage of detections that were false positives, and the percentage of faces that were not detected. The majority of the false detections are in sub-images much smaller or larger than a person’s face or outside our possible face regions. Since we eliminate such sub-images, we are able to greatly reduce the amount of false alarms. Figure 3.6 contains a few examples of false detections removed by our proposed method. When a false alarm is roughly the size of a face and near a possible face region, like in Figure 3.6h, we are unable to eliminate it. Fortunately, these situations seldomly occur.

Table 3.2: Percentage of false and miss detections.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Viola-Jones (%)</th>
<th>Our Method (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>False Detection</td>
<td>9.86</td>
<td>0.30</td>
</tr>
<tr>
<td>Miss Detection</td>
<td>17.70</td>
<td>18.37</td>
</tr>
</tbody>
</table>

Most of the miss detections in our test sequences are caused by faces in a profile orientation. This is not surprising because the classifier was trained to detect frontal faces. The slight increase of miss detections with our method
Figure 3.6: Example results from the standard Viola-Jones face detector (a, c, e, g), and our proposed method (b, d, f, h). In addition to accelerating the Viola-Jones face detector, our method also significantly reduces false detections.
was caused by a person leaving the operational range of the depth sensor. Specifically, the user moved to within 40 centimeters from the camera, as shown in Figure 3.7. Since the depth measurements on the face are invalid and ineffectual for face modeling, it is unnecessary to be able to detect the face in this situation.

3.2 Face Segmentation

After identifying the location of the face, it is necessary to segment the depth image by removing all depth pixels that do not measure the surface of the face. Our goal is to model all facial features; unfortunately, the face detector only recognizes a part of the face. The area of the face identified by the detector often omits part of the forehead and chin, as well as the ears. In addition, the region detected can fluctuate between images. We are able to extract the entirety of the face by analyzing the depth data.

Using face detection, we establish a set of depth pixels, $R$, that lie on the face. We use a region growing algorithm to find the remainder of the face. By recursively examining depth pixels on the boundary of the set, we can expand the set to contain the entire face. The pseudocode for this method is shown in Algorithm 1.
Algorithm 1: Region growing algorithm

The initial set of face pixels $\mathcal{R} = \{(i,j)\}_{n=1}^{N}$

foreach $(i,j) \in \mathcal{R}$ do
    foreach $(k,l) \in \text{Neighbor}(i,j)$ do
        if $(k,l) \notin \mathcal{R}$ and $|Z_{i,j} - Z_{k,l}| < \delta_{\text{max}}$ then
            $\mathcal{R} \leftarrow \mathcal{R} \cup (k,l)$
        end
    end
end

The region growing method will often extend the set of depth pixels beyond the face to include the entire body. In this case, we need to determine the horizontal line, $y$, that separates the head from the torso. The depth measurements above the line are considered part of the head region,

$$\mathcal{R}_H = \{(i,j) \mid (i,j) \in \mathcal{R} \text{ and } j \leq y\} \quad (3.7)$$

and everything below the line is included in the torso region,

$$\mathcal{R}_T = \{(i,j) \mid (i,j) \in \mathcal{R} \text{ and } j > y\}. \quad (3.8)$$

If the width of the head is significantly different than the width of the torso, then we can calculate the scanline that maximizes the difference between the mean width of the head region,

$$\mu_H = \frac{\text{# of pixels in } \mathcal{R}_H}{\text{# of rows that contain a pixel in } \mathcal{R}_H} \quad (3.9)$$

and the mean width of the torso region,

$$\mu_T = \frac{\text{# of pixels in } \mathcal{R}_T}{\text{# of rows that contain a pixel in } \mathcal{R}_T}. \quad (3.10)$$

Specifically, we compute,

$$y = \arg\max_y w_H w_T (\mu_H - \mu_T)^2 \quad (3.11)$$
Figure 3.8: Example segmentation result. Our method takes the input depth image (a), identifies the person using face detection (box in (b)), extracts the person using the region growing algorithm (b), calculates the horizontal line that separates the head from the torso (line in (c)), and generates the segmented depth image (d).

where,

\[ w_H = \frac{\# \text{ of rows that contain a pixel in } R_H}{\# \text{ of rows that contain a pixel in } R} \]  \hspace{1cm} (3.12)

and,

\[ w_T = \frac{\# \text{ of rows that contain a pixel in } R_T}{\# \text{ of rows that contain a pixel in } R} \]  \hspace{1cm} (3.13)

The horizontal line evaluated in this fashion will typically contain the neck, which can be removed by simply shifting the line upwards. Afterwards, we can generate the segmented depth image by removing all depth measurements that are outside the head region. If \((i,j) \notin R_H\), then we set \(Z_{i,j} = 0\). The segmentation process is summarized in Figure 3.8
CHAPTER 4
FACE REGISTRATION AND INTEGRATION

The depth pixels in the segmented depth image give us a rough model of the face. Unfortunately, the depth images captured by a consumer depth camera are contaminated with noise, and parts of the image may not contain valid depth measurements. Our method improves the model of the face geometry by registering and integrating a sequence of depth images. By fusing the surface measurements from a sequence of depth images, we are able to reduce noise and fill in holes.

4.1 Surface Alignment

To construct the 3D face model, we capture a sequence of depth images. Each segmented depth image contains surface measurements of the face in a particular pose. The measurements are represented by depth pixels which correspond to 3D points on the surface of the face. To combine the sequence of images, the point clouds need to be transformed into a reference orientation. We define the reference orientation as the pose of the face in the initial depth image, and all subsequent images are registered to this orientation.

We register two point clouds by computing the rigid body transformation, \( T \), that aligns the surfaces,

\[
T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}.
\] (4.1)

The transformation matrix, \( T \), is composed of a rotation matrix,

\[
R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix} \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}.
\] (4.2)
where $\alpha$, $\beta$, and $\gamma$ are the rotation around the $x$, $y$, and $z$ axes, respectively, and a translation vector,

$$
t = \begin{bmatrix}
    t_x \\
    t_y \\
    t_z
\end{bmatrix}
$$

(4.3)

where $t_x$, $t_y$, and $t_z$ are the translation amounts along the $x$, $y$, and $z$ axes.

We can register the entire sequence by incrementally aligning two consecutive point clouds. By multiplying the series of transformations, we can align the last point cloud to the initial point cloud. Therefore, we can register each new depth image as it is captured by the camera.

### 4.1.1 Noise Reduction

Before calculating the rigid body transformation, it is beneficial to filter the depth images, as noise can degrade our ability to align the surfaces. We use a bilateral filter [17] because it reduces noise while preserving depth discontinuities. The output of a bilateral filter is determined by a nonlinear combination of neighboring pixels,

$$
Z_{i,j}' = \frac{\sum_{(k,l) \in \mathcal{N}(i,j)} c(i, j, k, l) s(Z_{i,j}, Z_{k,l}) Z_{k,l}}{\sum_{(k,l) \in \mathcal{N}(i,j)} c(i, j, k, l) s(Z_{i,j}, Z_{k,l})}
$$

(4.4)

where $\mathcal{N}(i,j)$ is the neighborhood of the $(i,j)^{th}$ pixel. The weight of a neighboring pixel is based on the closeness of the pixels,

$$
c(i,j,k,l) = \exp \left( -\frac{(i-k)^2 + (j-l)^2}{2\sigma_c^2} \right)
$$

(4.5)

and the similarity of their depth measurements,

$$
s(Z_{i,j}, Z_{k,l}) = \exp \left( -\frac{(Z_{i,j} - Z_{k,l})^2}{2\sigma_s^2} \right).
$$

(4.6)

Figure 4.1 compares the surface measurements before and after applying the bilateral filter.
Figure 4.1: Comparison between the noisy surface measurements and the filtered surface measurements. To visualize the differences, we display the normal maps corresponding to the raw measurements (a) and the filtered measurements (b).

4.1.2 Point-Cloud Generation

Each depth pixel in an image represents a 3D point on the surface of the face. The camera’s intrinsic matrix,

\[
K = \begin{bmatrix}
    f_x & 0 & c_x \\
    0 & f_y & c_y \\
    0 & 0 & 1
\end{bmatrix}
\]  \hspace{1cm} (4.7)

where \(f_x\) and \(f_y\) are the camera’s focal lengths, and \(c_x\) and \(c_y\) denote the camera’s principal point, defines the projective mapping from the 3D point to the 2D pixel coordinate. The 3D point, \([X, Y, Z]^T\), is mapped to the pixel coordinate, \([i, j]^T\), by projecting the point onto the image plane,

\[
\begin{bmatrix}
    i \\
    j \\
    1
\end{bmatrix} = \begin{bmatrix}
    X * f_x + Z * c_x \\
    Y * f_y + Z * c_y \\
    Z
\end{bmatrix} = K \begin{bmatrix}
    X \\
    Y \\
    Z
\end{bmatrix}
\]  \hspace{1cm} (4.8)

therefore, \(i = (X * f_x) / Z + c_x\) and \(j = (Y * f_y) / Z + c_y\). From the depth image, we know the pixel coordinate \(i\) and \(j\), as well as, the depth \(Z\).
result, we can reverse the projection to recover \( X \) and \( Y \),

\[
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} =
\begin{bmatrix}
Z \ast \frac{(i-c_x)}{f_x} \\
Z \ast \frac{(j-c_y)}{f_y} \\
Z
\end{bmatrix}
= K^{-1}Z
\begin{bmatrix}
i \\
j \\
1
\end{bmatrix}.
\tag{4.9}
\]

Reversing the projection is demonstrated in Figure 4.2. We generate a 3D point cloud for each of the depth images by back-projecting all of the pixel within the image. The vertex corresponding to the \((i, j)\)th pixel is \( v_{i,j} = [X_{i,j}, Y_{i,j}, Z_{i,j}]^T \), and its surface normal is estimated by taking the cross product of neighboring vertices,

\[
n_{i,j} = (v_{i+1,j} - v_{i,j}) \times (v_{i,j+1} - v_{i,j}).
\tag{4.10}
\]

Figure 4.3 shows an example of a point cloud constructed by un-projecting a depth image.

### 4.1.3 Iterative Closest Point

Iterative closest point (ICP) [18] is a common way of computing the rigid body transformation that aligns two 3D point clouds. There are many variants of ICP, but all consist of the following steps [18]:

1. Determine corresponding points between point clouds.
2. Compute the rigid body transformation by minimizing an error metric.
Figure 4.3: The point cloud (b) was generated by back-projecting the depth pixels in image (a).

Figure 4.4: The purpose of registration is to align two surfaces (a). Iterative closest point (ICP) computes the rigid body transformation that registers two roughly aligned surfaces (b). ICP iteratively refines the transformation by identifying point correspondences and calculating the incremental transformation that minimizes an error metric. ICP repeats this process until the surfaces are aligned (c).

3. Apply the transformation to a point cloud.

4. Repeat for a number of iterations.

A high-speed variant of ICP, suggested by [19], uses projective data association and the point-to-place error metric. Projective data association [20] allows point correspondences to be computed directly by finding corresponding points along camera rays. The point-to-plane error metric [21] minimizes the distance between a point and the corresponding point’s tangent plane, which has been shown to have improved convergence rates over other error metrics [19]. Figure 4.4 depicts the registration of two point clouds using ICP.
ICP requires an initial estimate for the rigid body transformation, $T$, that aligns the source point cloud, $S$, to the destination point cloud, $D$. We know that both point clouds consist of 3D points on the surface of the face, so our initial alignment could be the transformation that aligns their centroids,

$$T = \begin{bmatrix} I & (\mu_D - \mu_S) \\ 0 & 1 \end{bmatrix}$$ (4.11)

where $I$ is a $3 \times 3$ identity matrix, and $\mu_S$ and $\mu_D$ are the 3D centers of the source and destination point clouds, respectively.

Identifying corresponding points

To register surfaces, we need to identify a set of corresponding points between the source and destination point clouds. As long as the relative motion of the face between source and destination is minimal, we can use projective data association to find corresponding points along camera rays.

Projective data association begins by using our current estimate of the transformation, $T$, to roughly align the two point clouds. Afterwards, the vertices in the source point cloud, $S$, are projected into the image plane using the camera’s intrinsic matrix, $K$, and the pixel coordinates are used to identify the corresponding points in destination point cloud, $D$. The projective data association technique is illustrated in Figure 4.5, and the pseudocode is listed in Algorithm 2.
Algorithm 2: Projective data association

Source point cloud $\mathcal{S} = \{(v_{i,j}, n_{i,j})_n\}_{n=1}^N$ where $(i, j)$ is the pixel coordinate corresponding to the vertex and normal.

Destination point cloud $\mathcal{D} = \{(v'_{i,j}, n'_{i,j})_m\}_{m=1}^M$ where $(i, j)$ is the pixel coordinate corresponding to the vertex and normal.

Initialize the set of point correspondences $\mathcal{C} = \{\emptyset\}$.

\begin{verbatim}
foreach $(v_{i,j}, n_{i,j}) \in \mathcal{S}$ do
    \[ \hat{v}_{i,j}, 1\]^T = T \[ v_{i,j}, 1\]^T \]
    \[ \hat{n}_{i,j}, 0\]^T = T \[ n_{i,j}, 0\]^T \]
    \[ k, l, 1\]^T = \text{K} \hat{v}_{i,j} \]
    if $(v'_{k,l}, n'_{k,l}) \in \mathcal{D}$ then
        if $\|\hat{v}_{i,j} - v'_{k,l}\| < \delta_{\text{max}}$ and $\cos^{-1}(\hat{n}_{i,j} \cdot n'_{k,l}) < \theta_{\text{max}}$ then
            $\mathcal{C} \leftarrow \mathcal{C} \cup (\hat{v}_{i,j}, v'_{k,l}, n'_{k,l})$
        end
    end
end
\end{verbatim}

Computing the incremental transformation

We iteratively refine our estimation of the rigid body transformation, $T$, by computing the incremental transformation, $\tilde{T}$, that minimizes the point-to-
plane error metric \[21\],

\[
\tilde{T}^* = \arg\min_{T} \sum_i \left[ (\tilde{T}s_i - d_i)^T n_i \right]^2
\]  

(4.12)

where, \(s_i = [s_{xi}, s_{yi}, s_{zi}, 1]^T\) is the source point transformed by our estimate of the transformation, \(T\), \(d_i = [d_{xi}, d_{yi}, d_{zi}, 1]^T\) is the corresponding destination point, and \(n_i = [n_{xi}, n_{yi}, n_{zi}, 0]^T\) is the normal vector of \(d_i\), all of which are in homogeneous coordinates. The point-to-plane error metric is displayed in Figure 4.6. Once we have solved for incremental transformation \(\tilde{T}\), we can refine our estimate of the rigid body transformation,

\[
T \leftarrow \tilde{T} T.
\]  

(4.13)

The parameters of \(\tilde{T}\) are rotation angles \(\alpha\), \(\beta\), and \(\gamma\), and the translation amounts \(t_x\), \(t_y\), and \(t_z\). The rotation angles are arguments to nonlinear trigonometric functions (Equation 4.2), as a result, the point-to-plane error metric, in this form, cannot be minimized using efficient linear least squares methods. If the rotation of the face does not change significantly between the source and destination, we can use the small-angle approximation to linearize the rotation matrix,

\[
\tilde{R} \approx \begin{bmatrix}
1 & -\gamma & \beta \\
\gamma & 1 & -\alpha \\
-\beta & \alpha & 1
\end{bmatrix}
\]  

(4.14)

where \(\sin\theta\) is replaced by \(\theta\) and \(\cos\theta\) is replaced by 1. With this approxima-
tion we can rewrite the point-to-plane error metric [22],

$$\sum_i \left[ (\bar{T}s_i - d_i)^T n_i \right]^2 = \|Ax - b\|^2 \quad (4.15)$$

where,

$$x = \begin{bmatrix} \alpha \beta \gamma \ t_x \ t_y \ t_z \end{bmatrix}^T \quad (4.16)$$

and the $i^{th}$ row of matrix $A$ equals,

$$A_i = \begin{bmatrix} s_y i n_x i - s_z i n_y i \ s_z i n_x i - s_x i n_z i \ s_x i n_y i - s_y i n_x i \ n_x i \ n_y i \ n_z i \end{bmatrix} \quad (4.17)$$

and the $i^{th}$ row of vector $b$ equals,

$$b_i = \begin{bmatrix} d_x i n_x i + d_y i n_y i + d_z i n_z i - s_x i n_x i - s_y i n_y i - s_z i n_z i \end{bmatrix}. \quad (4.18)$$

With the small-angle approximation, we can update the Equation 4.15,

$$x^* = \arg\min_x \|Ax - b\|^2$$

$$= \arg\min_x (Ax - b)^T (Ax - b) \quad (4.19)$$

$$= \arg\min_x x^T A^T Ax - 2x^T A^T b + b^T b.$$ 

By taking the derivative of the point-to-plane error metric and setting it equal to zero, we obtain a linear system of equations,

$$A^T Ax^* = A^T b \quad (4.20)$$

which can be solved efficiently using linear least squares techniques.

Since $A^T A$ is a symmetric positive-definite matrix, we can solve for $x^*$ using Cholesky decomposition [4]. Cholesky decomposition factorizes a symmetric positive-definite matrix, $M$, into the product of a lower triangular matrix and its transpose, $M = LL^T$ [23]. Afterwards, we can efficiently solve the system of equations $Mx = y$ by solving two triangular systems $Lz = y$ and $L^T x = z$ [23].
Iteration

For several iterations, we continue to refine the transformation, $T$, by updating the point correspondences and computing the incremental transformation that minimizes the point-to-plane error metric. We stop iterating when the error,

$$e = \sum_i \left( (\tilde{T}^* s_i - d_i)^T n_i \right)^2$$

converges, or after a fixed number of iterations. If we are unable to find the transformation that aligns a depth image to the reference orientation, the surface measurements from the image are discarded.

4.2 Surface Modeling

After registering the point clouds, we need to fuse the depth measurements into a single 3D face model. The depth images are incrementally fused using volumetric integration, which allows us to continually update the surface model as the camera captures new depth images. While constructing the 3D model of the face, we ray cast the volume to extract the current approximation of the surface. Once the entire sequence of depth images has been integrated, we use marching cubes to convert the volumetric model to a triangular mesh.

4.2.1 Volumetric Integration

We use an implicit function, $D(x)$, to describe the surface of the face. At each point, $x$, the function represents the weighted signed distance to the nearest surface along camera rays [24]. Each depth image has a corresponding signed distance function, $d(x)$, as well as a weight function, $w(x)$. We construct $D(x)$ by incrementally combining each image’s signed distance function and weight function,

$$D(x) \leftarrow \frac{W(x)D(x) + w(x)d(x)}{W(x) + w(x)}$$

$$W(x) \leftarrow W(x) + w(x)$$

where $W(X)$ is the accumulated weight function [24].

27
Ideally, the distance functions would represent the true signed distance to the nearest surface; however, it would be computationally expensive to compute this function for every depth image. As a result, the signed distance function, $d(x)$, is only accurate near the surface. To prevent surfaces from interfering with each other, the weight function, $w(x)$, should only be nonzero close to the surface. The surface measurements captured by the commodity depth camera are contaminated by noise, so the actual location of the surface will be within some range of the measurement. In order to integrate measurements from multiple depth images, the weight function, $w(x)$, should be nonzero within this range of uncertainty [24].

**Computing the weighted signed distance function**

The functions are represented on a discrete volume containing $256^3$ voxels. To encapsulate the entire surface of the face, the dimensions of the volume were chosen to be larger than the average human head diameter. For each depth image, we need to compute $d(x)$ and $w(x)$ at every voxel.

To compute the signed distance from a voxel, $x$, to the nearest surface along camera rays, we need to first identify the camera ray that passes through the voxel. We determine this camera ray by transforming the voxel from the reference orientation to the orientation of the depth image, using its inverse rigid body transformation, $T^{-1}$, and by projecting the voxel onto the image plane using the camera’s intrinsic matrix, $K$. The voxel will be projected onto some pixel, $(i, j)$, and the ray that passes through this pixel will also pass through the voxel. Utilizing the depth measurement, $Z_{i,j}$, stored at the $(i, j)^{th}$ pixel, we can compute the distance from the camera to the surface. The signed distance is calculated by subtracting the distance between the voxel and the camera from the distance between the camera and the surface,

$$d(x) = Z_{i,j} \| K^{-1} [i, j, 1]^T \|_2 - \| \hat{x} \|_2$$  \hspace{1cm} (4.24)

where

$$[\hat{x}, 1]^T = T^{-1} [x, 1]^T$$  \hspace{1cm} (4.25)

and

$$[i, j, 1]^T = K \hat{x}.$$  \hspace{1cm} (4.26)
The signed distance will be positive if the voxel is in front of the surface, negative if it is behind the surface, and zero if it is on the surface. The signed distance function along camera rays is illustrated in Figure 4.7.

The weight function, $w(x)$, corresponding to the signed distance function, $d(x)$, depends on the type of scanner used to measure the surface. For each sensor, there is some amount of uncertainty in the surface measurements. The uncertainty range for a commodity depth camera is on the order of millimeters. Therefore, the weight function should be nonzero within a few millimeters of the surface to allow for multiple measurements to be integrated. In addition, we can use the weight function to favor surface measurements that we believe to be more accurate. For the consumer depth camera, there is a greater uncertainty in a measurement when the surface is viewed at a glancing angle, or when it is viewed from a considerable distance. Accordingly, we base the weights on the angle between the surface normal and the viewing direction, as well as the distance from the camera to the surface [24].

Since the signed distance function is only accurate near the surface, we truncate the signed distances so that the values are between $+\mu$ and $-\mu$, where $\mu$ is the range of uncertainty for the depth camera [4]. To prevent surfaces on opposite sides of the face from interfering with each other, we set the weight to zero when the non-truncated signed distance is less than $-\mu$.

Volumetric integration is demonstrated in Figure 4.8, and the pseudocode for it is provided in Algorithm 3.
Figure 4.8: An example of two depth images being fused together using volumetric integration. (c) and (d) are slices of the signed distance function corresponding to the depth images (a) and (b), respectively. (e) is the combination of the two slices.
**Algorithm 3: Volumetric integration**

```plaintext
foreach \( x \) in the volume do
  \([\hat{x}, 1]^T = T^{-1} [x, 1]^T\)
  \([i, j, 1]^T = K \hat{x}\)
  if \( Z_{i,j} > 0 \) then
    \( d = Z_{i,j} \left\| K^{-1} [i, j, 1]^T \right\|_2 - \left\| \hat{x} \right\|_2 \)
    if \( d > -\mu \) then
      \( d(x) = \text{sgn} (d) \min \left( 1, \frac{|d|}{\mu} \right) \)
      \( w(x) = \exp \left( n_{i,j} \cdot [0, 0, 1]^T - 1 \right) \exp \left( -\frac{Z_{i,j}}{2\sigma} \right) \)
      \( D(x) \leftarrow \frac{W(x)D(x) + w(x)d(x)}{W(x) + w(x)} \)
      \( W(x) \leftarrow W(x) + w(x) \)
  end
end
```

### 4.2.2 Surface Extraction

We use an implicit function, \( D(x) \), to represent the geometry of the face. Therefore, to extract the surface from the volumetric model, we need to determine where the function is zero. We utilize two methods to extract the surface: ray casting [25] and marching cubes [26].

**Ray casting**

To monitor the condition of the model while reconstructing the face, we use ray casting. Rendering the state of the model, after integrating each depth image, enables the user to reposition his or her head to improve parts of the model. In addition, we can use ray casting to update the surface measurements in a depth image, which improves our ability to register subsequent images [4].

Ray casting extracts the surface by casting several rays through the volume and determining where the implicit function, \( D(x) \), equals zero. For each pixel, a corresponding ray is stepped through the volume. At each step, \( s \),
we determine the ray’s position in the volume,

\[ x_s = t + d_s \left( RK^{-1} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} \right) \]  \hspace{1cm} (4.27)

where \((i, j)\) is the pixel coordinate the ray was cast through, \(K\) is the camera’s intrinsic matrix, \(R\) and \(t\) are the rotation matrix and translation vector from the rigid body transformation, \(T\), and \(d_s\) is the distance traveled by the ray during steps 1, \ldots, \(s\).

The signed distance, \(D(x_s)\), is computed by trilinear interpolation of neighboring voxels [25]. The surface is found when the ray steps from a positive signed distance, \(D_s^+ = D(x_s) > 0\), in front of the surface, to a negative signed distance, \(D_{s+1}^- = D(x_{s+1}) < 0\), behind the surface. We can determine the surface intersection by interpolating the signed distances [4],

\[ d^* = d_s + \frac{(d_{s+1} - d_s) \, D_s^+}{D_s^+ - D_{s+1}^-} \]  \hspace{1cm} (4.28)

Afterwards, we can update our surface measurement,

\[ v_{i,j} = t + d^* \left( RK^{-1} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} \right) \]  \hspace{1cm} (4.29)

Also, we can update our estimate of the surface normal,

\[ n_{i,j} = \nabla D (v_{i,j}) \]  \hspace{1cm} (4.30)

since the surface normal is equal to the gradient of the implicit function [25].

The amount the ray travels between steps is less than the uncertainty range, \(\mu\), since a larger interval may cause the ray to miss the zero-crossing [4]. Figure 4.9 shows several examples of surfaces extracted using ray casting, and Figure 4.10 illustrates the ray-casting process.
Figure 4.9: Surfaces extracted by ray casting the volumetric model after a number of depth images have been integrated.
Marching cubes

After integrating the entire sequence of depth images into the volumetric model, we can use marching cubes to extract the surface and construct a triangular mesh [26].

Marching cubes extracts the surface by marching through the volume. At each step, it examines a cube created by eight voxels, and determines whether or not the surface intersects the cube and where. If the surface intersects the cube, some of the eight voxels will have a positive signed distance and the others will have a negative signed distance. There are $2^8 = 256$ ways for the surface to pass through the cube, and the marching cubes algorithm uses a lookup table to enumerate all the different cases [26]. Once we determine the topology of the surface within the cube, we use the signed distances stored in the voxels to refine the position of triangles.

After marching cubes has marched through the entire volume, we have a triangular mesh representing the surface of the face. Figure 4.11 depicts a triangular mesh generated using marching cubes.
Figure 4.11: Triangular mesh generated by marching cubes.
CHAPTER 5

EXPERIMENTAL RESULTS

The objective of our proposed method is to construct a high quality 3D face model in real time, using a low-cost depth camera. To evaluate our method, we analyze the efficiency of our algorithm, as well as the quality of our 3D face model.

5.1 Runtime

We would like the user to be able to inspect the 3D model as it is being constructed; therefore, the efficiency of our algorithm is important. In addition, a poor performing algorithm would affect our ability to align the surfaces, since we assume the relative pose of the face does not change significantly between images.

To execute our algorithm in real time on commodity hardware, we parallelized several components of our method. We run our algorithm on a multicore central processing unit (CPU) and a graphics processing unit (GPU). We profiled the runtime of our system using an Intel Core i7 CPU and a NVIDIA GeForce GTX 570 GPU, and the results are listed in Table 5.1. Our system is capable of processing more than 15 images per second.

Table 5.1: Runtime of our method.

<table>
<thead>
<tr>
<th>Component</th>
<th>Runtime (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face Detection &amp; Segmentation</td>
<td>19.99</td>
</tr>
<tr>
<td>Noise Reduction</td>
<td>0.48</td>
</tr>
<tr>
<td>Point Cloud Generation</td>
<td>0.09</td>
</tr>
<tr>
<td>Iterative Closest Point</td>
<td>24.18</td>
</tr>
<tr>
<td>Volumetric Integration</td>
<td>2.53</td>
</tr>
<tr>
<td>Surface Extraction</td>
<td>18.59</td>
</tr>
<tr>
<td>Total</td>
<td>65.90</td>
</tr>
</tbody>
</table>
5.2 Quality

To assess the quality of our face model, we compared our 3D model that was built using a low-resolution depth camera, to the 3D model constructed using a high-resolution 3D scanner (Figure 5.1). To generate the reference model, we used a Steinbichler Comet L3D scanner, which captures the geometry of the face using structured light projection [27].

We analyzed the error of our method by computing the absolute distance between each point in our model, to the closest point in the reference model. Figure 5.2 visualizes the error in our model. The mean absolute distance between the two models is 0.4988 millimeters with a standard deviation of 0.5300 millimeters.

Although our method is unable to capture the same level of detail as the high-resolution scanner, we are able to construct an accurate model of the face in less time with a device that is a fraction of the price. A consumer depth camera, such as Microsoft’s Kinect, costs roughly $100, whereas the Steinbichler Comet L3D scanner cost upwards of $60,000. Additionally, our method builds a 3D model of the face in seconds, whereas the high-resolution Scanner.

Figure 5.1: Comparison between a face model built using our method (a) and a face model constructed using a high-resolution 3D scanner (b).
Figure 5.2: Visualization of the model’s surface error. The error is determined by computing the absolute distance in millimeters between each point in our model to the closest point in the reference model.

scanner requires tens of minutes. Our system allows a person to model his or her face simply by moving his or her head in front of a fixed depth camera; however, the Comet L3D scanner requires the person to remain perfectly still while another person operates the device. Our proposed method provides a trade-off between high levels of accuracy and convenience.
CHAPTER 6

CONCLUSION

In this thesis, we proposed a method for 3D face modeling with a consumer depth camera. Prior to the development of low-cost depth cameras, constructing a high quality face model required an expensive 3D scanner. Depth cameras, like Microsoft’s Kinect, provide low-resolution depth images, and by combining multiple depth images, we are able to produce a high quality model of the face. To efficiently segment a face from a depth image, we developed a method for accelerating the face detection process. In addition, we are able to fuse a sequence of depth images in real time by incrementally registering and integrating the surface measurements. We demonstrated that our method can produce accurate 3D face models that are comparable to models constructed using a high-resolution 3D scanner.
REFERENCES


