Rapid Modeling of Animated Faces From Video

Zicheng Liu, Zhengyou Zhang, Chuck Jacobs, Michael Cohen

February 28, 2000

Technical Report
MSR-TR-2000-11

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

{zliu,zhang,cjacobs,mcohen}@microsoft.com
http://research.microsoft.com/~zhang

Rapid Modeling of Animated Faces From Video

Zicheng Liu, Zhengyou Zhang, Chuck Jacobs, Michael Cohen

February 28, 2000

Abstract

Generating realistic 3D human face models and facial animations has been a persistent challenge in computer graphics. We have developed a system that constructs textured 3D face models from videos with minimal user interaction. Our system takes images and video sequences of a face with an ordinary video camera. After five manual clicks on two images to tell the system where the eye corners, nose top and mouth corners are, the system automatically generates a realistic looking 3D human head model and the constructed model can be animated immediately. A user, with a PC and an ordinary camera, can use our system to generate his/her face model in a few minutes.
Keywords: Face modeling, facial animation, geometric modeling, computer vision

1 Introduction

One of the most interesting and difficult problems in computer graphics is the effortless generation of realistic looking, animated human face models. Animated face models are essential to computer games, film making, online chat, virtual presence, video conferencing, etc. So far, the most popular commercially available tools have utilized laser scanners. Not only are these scanners expensive, the data are usually quite noisy, requiring hand touchup and manual registration prior to animating the model. Because inexpensive computers and cameras are widely available, there is great interest in producing face models directly from images. In spite of progress toward this goal, the available techniques are either manually intensive or computationally expensive.

The goal of our system is to allow an untrained user with a PC and an ordinary camera to create and instantly animate his/her face model in no more than a few minutes. The user interface for the process comprises three simple steps. First, the user is instructed to pose for two still images. The user is then instructed to turn his/her head horizontally, first in one direction and then in the other. Third, the user is instructed to identify a few key points in the images. Then the system computes the 3D face geometry from the two images, and tracks the video sequences to create a complete facial texture map by blending frames of the sequence. The key observation is that even though it is difficult to extract dense 3D facial geometry from two images, it is possible to match a sparse set of corners and use them to compute head motion and the 3D locations of these corner points. We can then fit a linear class of human face geometries to this sparse set of reconstructed corners to generate the complete face geometry. Linear classes of face geometry and image prototypes have been demonstrated for constructing 3D face models from images in a morphable model framework [3]. In this paper, we show that linear classes of face geometries can be used to effectively fit/interpolate a sparse set of 3D reconstructed points. This novel technique is the key to quickly generating photorealistic 3D face models with minimal user intervention.

The paper is organized as follows. Related work on face modeling is reviewed in Section 2. Section 3 provides an overview of our system. Section 4 introduces the notation used in this paper, while Section 5 describes the animated face model we use in our system. We provide details of face geometry recovery from two images in Section 6 and details of complete texture map creation from videos in Section 7. A detailed example illustrates the user interface. More results are described in Section 10. We give the conclusions and perspectives of our system in Sections 11 and 12, respectively.

2 State of the Art

Facial modeling and animation has been a computer graphics research topic for over 25 years [6, 19, 20, 21, 22, 23, 24, 25, 26, 30, 33, 34, 36]. The reader is referred to Parke and Waters' book [26] for a complete overview.

Lee et al. [20, 21] developed techniques to clean up and register data generated from laser scanners. The obtained model is then animated by using a physically based approach.

DeCarlo at al. [5] proposed a method to generate face models based on face measurements randomly generated according to anthropometric statistics. They showed that they were able to generate a variety of face geometries using these face measurements as constraints.

A number of researchers have proposed to create face models from two views [1, 16, 4]. They all require two cameras which must be carefully set up so that their directions are orthogonal. Zheng [40] developed a system to construct geometrical object models from image contours, but it requires a turn-table setup.

Pighin et al. [29] developed a system to allow a user to manually specify correspondences across multiple images, and use vision techniques to compute 3D reconstructions. A 3D mesh model is then fit to the reconstructed 3D points. They were able to generate highly realistic face models, but with a manually intensive procedure.

Our work is inspired by Blanz and Vetter's work [3]. They demonstrated that linear classes of face geometries and images are very powerful in generating convincing 3D human face models from images. We use a linear class of geometries to constrain our geometrical search space. One main difference is that we do not use an image database. Consequently, the types of skin colors we can handle are not limited by the image database. This eliminates the need for a fairly large image database to cover every skin type. Another advantage is that there are fewer unknowns since we need not solve for image database coefficients and illumination parameters. The price we pay is that we cannot generate models from a single image.

Kang et al. [17] also use linear spaces of geometrical models to construct 3D face models from multiple images. But their approach requires manually aligning the generic mesh to one of the images, which is in general a tedious task for an average user.

Guenter et al. [12] developed a facial animation capturing system to capture both the 3D geometry and texture image of each frame and reproduce high quality facial animations. The problem they solved is different from what is addressed here in that they assumed the person's 3D model was available and the goal was to track the subsequent facial deformations.

Fua et al. [9, 10, 11, 8] have done impressive work on face modeling from images, and their approach is the most similar to ours in terms of the vision technologies exploited. There are three major differences between their work and ours: face model representation, model fitting, and camera motion estimation. They deform a generic face model to fit dense stereo data. Their face model contains

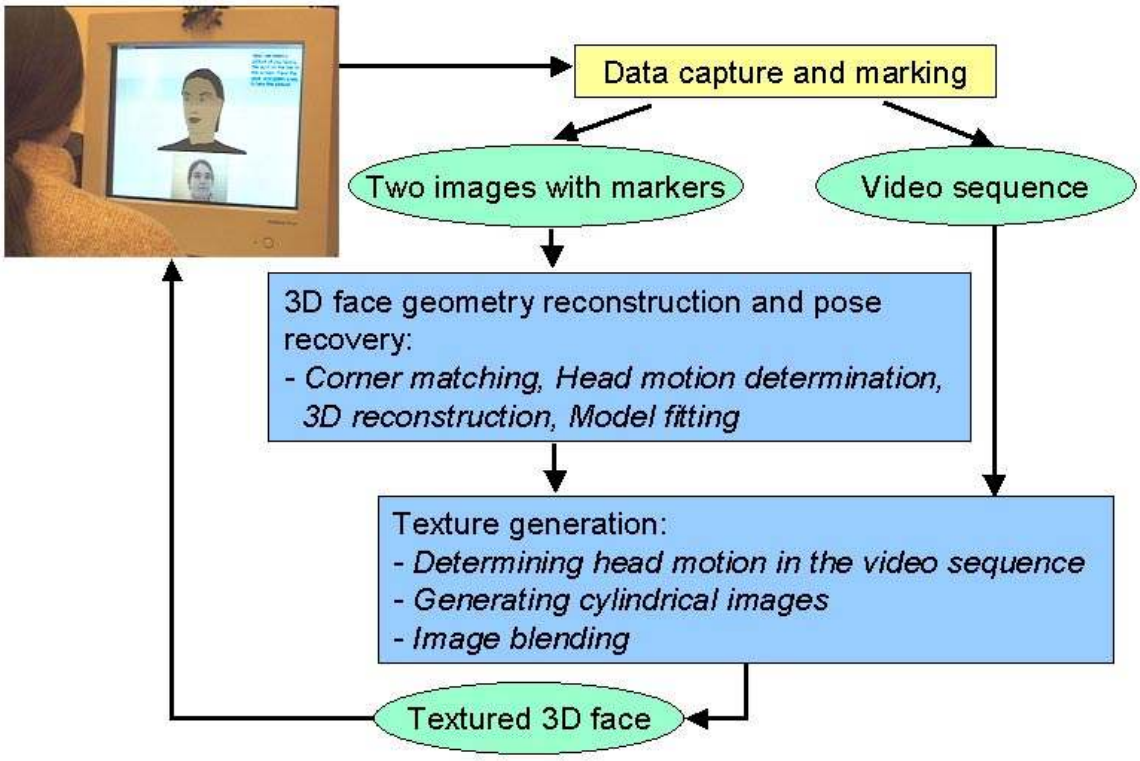


Figure 1: System overview

a lot of free parameters (each vertex has three unknowns). Although some smoothness constraint is imposed, the fitting still requires many 3D reconstructed points. With our model, we only have about 60 parameters to estimate (see Section 5), and thus only a small set of feature points is necessary. Dense stereo matching is usually computationally more expensive than feature matching. Our system can produce a head model with 40 images in about two minutes on a PC with 366 MHz processor. Regarding camera motion estimation, they use a regularized bundle-adjustment on every image triplet, while we use a bundle-adjustment on the first two views and determine camera motion for other views using 3D head model.

3 System Overview

Figure 1 outlines the components of our system. The equipment include a computer and a video camera. We assume the intrinsic camera parameters have been calibrated, a reasonable assumption given the simplicity of calibration procedures [39].

The first stage is data capture. The user takes two images with a small relative head motion, and two video sequences: one with head turning to each side. Or alternatively, the user can simply turn his/her head from left all the way to the right, or vice versa. In that case, the user needs to select two approximately frontal views with head motion of 5 to 10 degrees, and edit the video into two sequences.

In the sequel, we call the two images the *base images*.

The user then locates 5 markers in each of the two base images. The 5 markers correspond to the two inner eye corners, nose top, and two mouth corners (see Figure 3 for an example).

The next processing stage computes the face mesh geometry and the head pose with respect to the camera frame using the two base images and markers as input.

The final stage determines the head motions in the video sequences, and blends the images to generate a facial texture map.

4 Notation

We denote the homogeneous coordinates of a vector $\mathbf{x}$ by $\tilde{\mathbf{x}}$, i.e., the homogeneous coordinates of an image point $\mathbf{m} = (u, v)^T$ are $\tilde{\mathbf{m}} = (u, v, 1)^T$, and those of a 3D point $\mathbf{p} = (x, y, z)^T$ are $\tilde{\mathbf{p}} = (x, y, z, 1)^T$. A camera is described by a pinhole model, and a 3D point $\mathbf{p}$ and its image point $\mathbf{m}$ are related by

$$\lambda \tilde{\mathbf{m}} = \mathbf{A}\mathbf{P}\Omega\tilde{\mathbf{p}} \tag{1}$$

where λ is a scale, and $\mathbf{A}$, $\mathbf{P}$ and Ω are given by

$$\mathbf{A} = \begin{pmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad \Omega = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix}$$

The elements of matrix $\mathbf{A}$ are the intrinsic parameters of the camera and matrix $\mathbf{A}$ maps the normalized image coordinates to the pixel image coordinates (see e.g. [7]). Matrix $\mathbf{P}$ is the perspective projection matrix. Matrix Ω is the 3D rigid transformation (rotation $\mathbf{R}$ and translation $\mathbf{t}$) from the object/world coordinate system to the camera coordinate system. When two images are concerned, a prime $'$ is added to denote the quantities related to the second image.

The fundamental geometric constraint between two images is known as the *epipolar constraint* [7, 38]. It states that in order for a point $\mathbf{m}$ in one image and a point $\mathbf{m}'$ in the other image to be the projections of a single physical point in space, or, in other words, in order for them to be matched, they must satisfy

$$\tilde{\mathbf{m}}'^T \mathbf{A}'^{-T} \mathbf{E} \mathbf{A}^{-1} \tilde{\mathbf{m}} = 0 \tag{2}$$

where $\mathbf{E} = [\mathbf{t}_r]_\times \mathbf{R}_r$ is known as the essential matrix, $(\mathbf{R}_r, \mathbf{t}_r)$ is the relative motion between the two images, and $[\mathbf{t}_r]_\times$ is a skew symmetric matrix such that $\mathbf{t}_r \times \mathbf{v} = [\mathbf{t}_r]_\times \mathbf{v}$ for any 3D vector $\mathbf{v}$.

5 Linear class of face geometries

Vetter and Poggio [35] represented an arbitrary face image as a linear combination of some number of prototypes and used this representation (called linear object class) for image recognition, coding, and image synthesis. Blanz and Vetter [3] used linear class of both images and 3D geometries for image matching and face modeling. The advantage of using linear class of objects is that it eliminates most of the non-natural faces and significantly reduces the search space.

Instead of representing a face as a linear combination of real faces, we represent it as a linear combination of a neutral face and some number of face *metrics* where a metric is vector that linearly deforms a face in certain way, such as to make the head wider, make the nose bigger, etc. To be more precise, let's denote the face geometry by a vector $\mathcal{S} = (\mathbf{v}_1^T, \ldots, \mathbf{v}_n^T)^T$ where $\mathbf{v}_i = (X_i, Y_i, Z_i)^T$ $(i = 1, \ldots, n)$ are the vertices, and a metric by a vector $\mathcal{M} = (\delta\mathbf{v}_1, \ldots, \delta\mathbf{v}_n)^T$, where $\delta\mathbf{v}_i = (\delta X_i, \delta Y_i, \delta Z_i)^T$. Given a neutral face $\mathcal{S}^0 = (\mathbf{v}_1^{0T}, \ldots, \mathbf{v}_n^{0T})^T$, and a set of m metrics $\mathcal{M}^j = (\delta\mathbf{v}_1^{jT}, \ldots, \delta\mathbf{v}_n^{jT})^T$, the linear space of face geometries spanned by these metrics is

$$\mathcal{S} = \mathcal{S}^0 + \sum_{j=1}^{m} c_j \mathcal{M}^j \quad \text{subject to } c_j \in [l_j, u_j] \tag{3}$$

where c_j's are the metric coefficients and l_j and u_j are the valid range of c_j. In our implementation, the neutral face and all the metrics are designed by an artist, and it is done once. The neutral face (see Figure 2) contains 194 vertices and 360 triangles. There are 65 metrics.

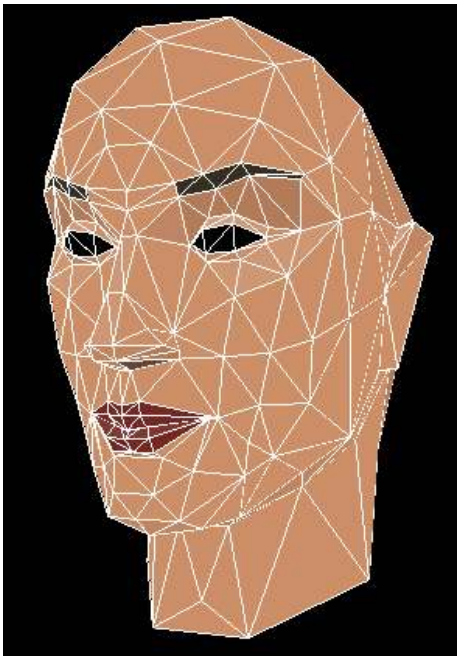


Figure 2: Neutral face.

6 Face Geometry From Two Views

In this section, we describe our techniques to determine the face geometry from just two views. The two base images are taken in a normal room by a static camera while the head is moving in front. There is no control on the head motion, and the motion is unknown, of course. We have to determine first the motion of the head and match some pixels across the two views before we can fit an animated face model to the images. However, some preprocessing of the images is necessary.

Figure 3: An example of two base images used for face modeling. Also shown are five manually picked markers indicated by yellow dots.

Throughout this section, we will use the two base images shown in Figure 3 to explain step by step our technique. The five manually picked markers are also shown.

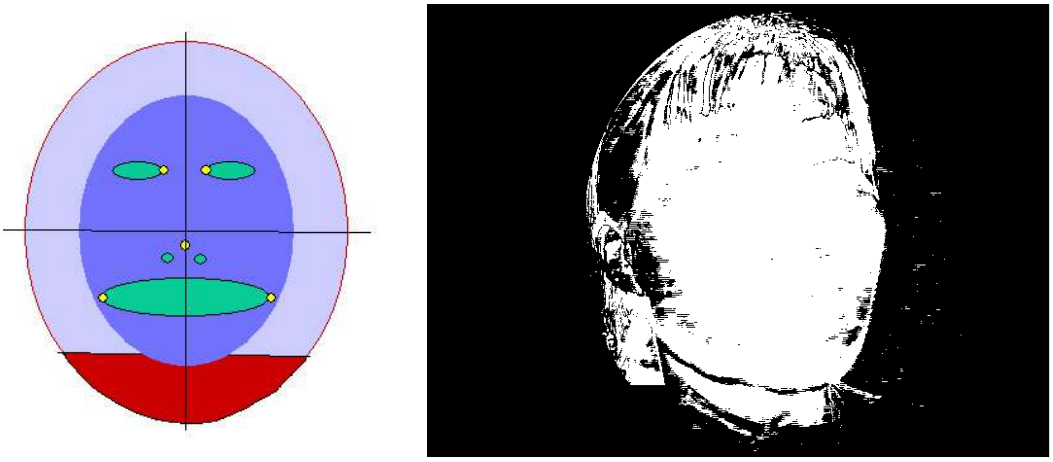
6.1 Preprocessing

(a) (b)

Figure 4: (a) Mask obtained by subtracting images (black pixels are considered as background); (b) Mask obtained by using a face skin color model (white pixels are considered as face skin).

There are at least three major groups of objects undergoing different motions between the two views: background, head, and other parts of the body such as the shoulder. If we do not separate them, there is no way to determine a meaningful head motion. Since the camera is static, we can expect to remove the background by subtracting one image from the other. However, as the face color changes smoothly, a portion of the face may be marked as background, as shown in Figure 4a. Another problem with this image subtraction technique is that the moving body and the head cannot be distinguished.

As we have marked five points on the face, we can actually build a color model of the face skin. We select pixels below the eyes and above the mouth, and compute a Gaussian distribution of their colors in the RGB space. If the color of a pixel matches this face skin color model, the pixel is marked as a part of the face. An example is shown in Figure 4b. As we can notice, some background pixels are marked as face skin.



(a) (b)

Figure 5: (a) Definition of face ellipses; (b) Final mask obtained with our preprocessing technique.

Either union or intersection of the two mask images is not enough to locate the face because it will include either too many (e.g., including undesired moving body) or too few (e.g., missing desired eyes and mouth) pixels. As we already have information about the position of eye corners and mouth corners, we define two ellipses as shown in Figure 5a. The inner ellipse covers most of the face, while the outer ellipse is usually large enough to enclose the whole head. Let d_e be the image distance between the two inner eye corners, and d_{em}, the vertical distance between the eyes and the mouth. The width and height of the inner ellipse are set to $5d_e$ and $3d_{em}$. The outer ellipse is 25% larger than the inner one. Within the inner ellipse, the "union" operation is used. Between the inner and out ellipses, only the image subtraction is used, except for the lower part where the "intersection" operation is used. The lower part aims at removing the moving body, and is defined to be $0.6d_{em}$ below the mouth, as illustrated by the red area in Figure 5a. An example of the final mask is shown in Figure 5b.

6.2 Corner matching and motion determination

One popular technique of image registration is optical flow [15, 2], which is based on the assumption that the intensity/color is conserved. This is not the case in our situation: the color of the same physical point appears to be different in images because the illumination changes when the head is moving. We therefore resort to a feature-based approach that is more robust to intensity/color variations. It consists of the following steps: (i) detecting corners in each image; (ii) matching corners between the two images; (iii) detecting false matches based on a robust estimation technique; (iv) determining the head motion; (v) reconstructing matched points in 3D space.

Corner detection. We use the Plessey corner detector, a well-known technique in computer vision [13]. It locates corners corresponding to high curvature points in the intensity surface if we view an image as a 3D surface with the third dimension being the intensity. Only corners whose pixels are white in the mask image are considered. See Figure 6 for the detected corners of the images shown in Figure 3.

Corner matching. For each corner in the first image, we choose an 11×11 window centered on it, and compare the window with windows of the same size, centered on the corners in the second image. A zero-mean normalized cross correlation between two windows is computed [7]. If we rearrange the pixels in each window as a vector, the correlation score is equivalent to the cosine angle between two intensity vectors. It ranges from -1, for two windows which are not similar at all, to 1, for two windows which are identical. If the largest correlation score exceeds a prefixed threshold (0.866 in our case), then that corner in the second image is considered to be the *match candidate* of the corner in the first image. The match candidate is retained as a *match* if and only if its match candidate in the first image happens to be *the corner being considered*. This symmetric test reduces many potential matching errors.

For the example shown in Figure 3, the set of matches established by this correlation technique is shown in Figure 6. There are 247 matches in total.

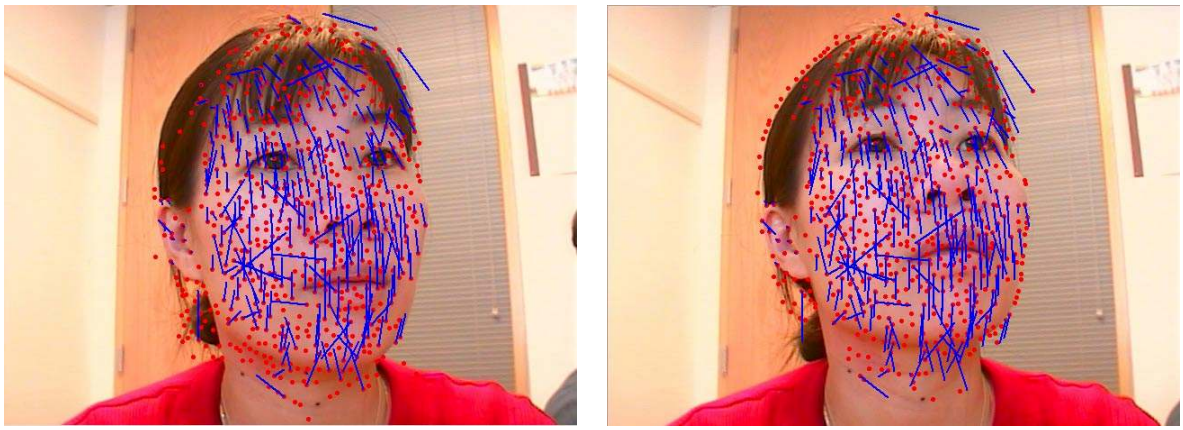


Figure 6: The set of matches established by correlation for the pair of images shown in Figure 3. Red dots are the detected corners. Blue lines are the motion vectors of the matches, with one endpoint (indicated by a red dot) being the matched corner in the current image and the other endpoint being the matched corner in the other image.

False match detection. The set of matches established so far usually contains false matches because correlation is only a heuristic. The only geometric constraint between two images is the epipolar constraint (2). If two points are correctly matched, they must satisfy this constraint, which is unknown in our case. Inaccurate location of corners because of intensity variation or lack of strong texture features is another source of error. We use the technique described in [38] to detect both false matches and poorly located corners, and simultaneously estimate the epipolar geometry (in terms of the essential matrix $\mathbf{E}$). That technique is based on a robust estimation technique known as the *least median squares* [31],

which searches in the parameter space to find the parameters yielding the smallest value for the *median* of squared residuals computed for the entire data set. Consequently, it is able to detect false matches in as many as 49.9% of the whole set of matches.

For the example shown in Figure 3, the final set of matches is shown in Figure 7. There are 154 remaining matches. Compared with those shown in Figure 6, it is noticed that 93 matches have been discarded.

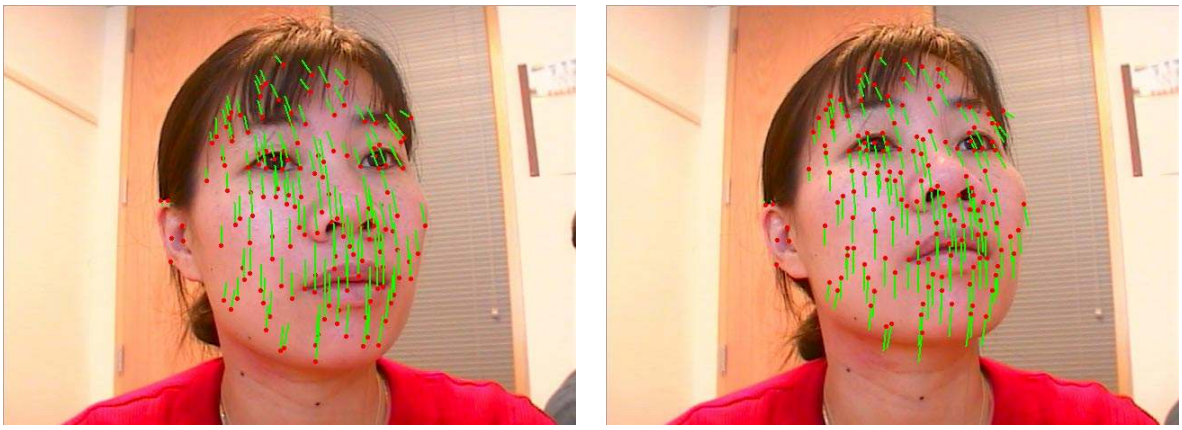


Figure 7: The final set of matches after discarding automatically false matches for the pair of images shown in Figure 3. Green lines are the motion vectors of the matches, with one endpoint (indicated by a red dot) being the matched corner in the current image and the other endpoint being the matched corner in the other image.

Motion estimation. An initial estimate of the relative head motion between two images, denoted by rotation $\mathbf{R}_r$ and translation $\mathbf{t}_r$), is computed from Matrix $\mathbf{E}$ [7, 37]. Motion $(\mathbf{R}_r, \mathbf{t}_r)$ is then re-estimated with a nonlinear least-squares technique using all remaining matches after having discarded the false matches [37].

3D reconstruction. Once the motion is estimated, matched points can be reconstructed in 3D space with respect to the camera frame at the time when the first base image was taken. Let $(\mathbf{m}, \mathbf{m}')$ be a couple of matched points, and $\mathbf{p}$ be their corresponding point in space. 3D point $\mathbf{p}$ is estimated such that $\|\mathbf{m} - \hat{\mathbf{m}}\|^2 + \|\mathbf{m}' - \hat{\mathbf{m}}'\|^2$ is minimized, where $\hat{\mathbf{m}}$ and $\hat{\mathbf{m}}'$ are projections of $\mathbf{p}$ in both images according to (1).

3D positions of the markers are determined in the same way.

Two views of the 3D reconstructed points for the example shown in Figure 3 are shown in Figure 8.

6.3 Fitting a face model

The face model fitting process consists of two steps: fitting to 3D reconstructed points and fine adjustment using image information.

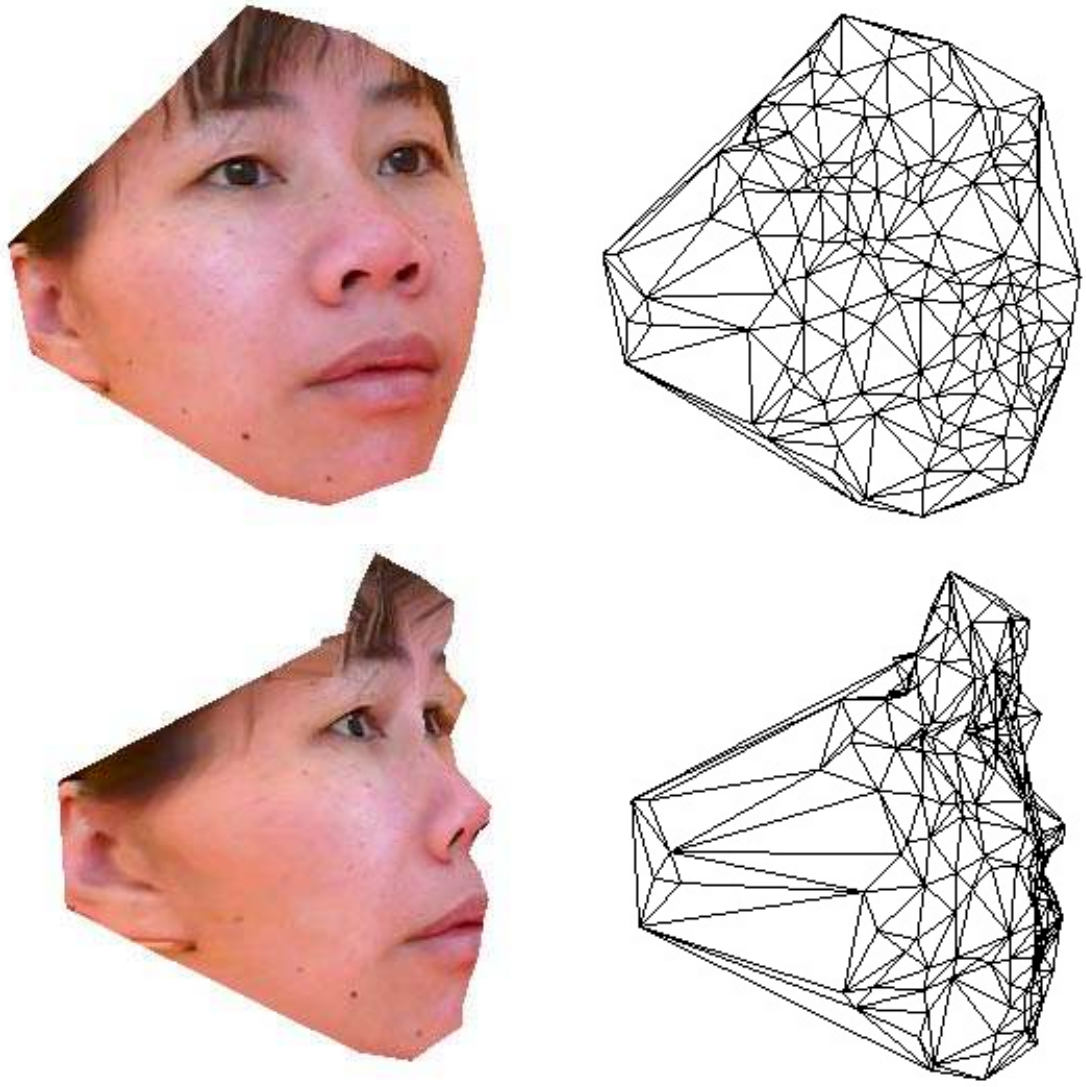


Figure 8: Reconstructed corner points. This coarse mesh is used later to fit a face model.

6.3.1 3D fitting

Given a set of reconstructed 3D points from matched corners and markers, the fitting process searches for both the *pose* of the face and the metric coefficients to minimize the distances from the reconstructed 3D points to the face mesh. The pose of the face is the transformation $\mathbf{T} = \begin{pmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix}$ from the coordinate frame of the neutral face mesh to the camera frame, where $\mathbf{R}$ is a 3×3 rotation matrix, $\mathbf{t}$ is a translation, and s is a global scale. For any 3D vector $\mathbf{p}$, we use notation $\mathbf{T}(\mathbf{p}) = s\mathbf{R}\mathbf{p} + \mathbf{t}$.

The vertex coordinates of the face mesh in the camera frame is a function of both the metric coefficients and the pose of the face. Given metric coefficients $(c_1, \ldots, c_m)$ and pose $\mathbf{T}$, the face

geometry in the camera frame is given by

$$\mathcal{S} = \mathbf{T}(\mathcal{S}^0 + \sum_{i=1}^{n} c_i \mathcal{M}^i). \tag{4}$$

Since the face mesh is a triangular mesh, any point on a triangle is a linear combination of the three triangle vertexes in terms of barycentric coordinates. So any point on a triangle is also a function of $\mathbf{T}$ and metric coefficients. Furthermore, when $\mathbf{T}$ is fixed, it is simply a linear function of the metric coefficients.

Let $(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k)$ be the reconstructed corner points, and $(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_5)$ be the reconstructed markers. Denote the distance from $\mathbf{p}_i$ to the face mesh $\mathcal{S}$ by $d(\mathbf{p}_i, \mathcal{S})$. Assume marker $\mathbf{q}_j$ corresponds to vertex $\mathbf{v}_{m_j}$ of the face mesh, and denote the distance between $\mathbf{q}_j$ and $\mathbf{v}_{m_j}$ by $d(\mathbf{q}_j, \mathbf{v}_{m_j})$. The fitting process consists in finding pose $\mathbf{T}$ and metric coefficients $\{c_1, \ldots, c_n\}$ by minimizing

$$\sum_{i=1}^{n} w_i d^2(\mathbf{p}_i, \mathcal{S}) + \sum_{j=1}^{5} d^2(\mathbf{q}_j, \mathbf{v}_{m_j}) \tag{5}$$

where w_i is a weighting factor.

To solve this problem, we use an iterative closest point approach. At each iteration, we first fix $\mathbf{T}$. For each $\mathbf{p}_i$, we find the closest point $\mathbf{g}_i$ on the current face mesh $\mathcal{S}$. We then minimize $\sum w_i d^2(\mathbf{p}_i, \mathbf{g}_i) + \sum d^2(\mathbf{q}_j, \mathbf{v}_{m_j})$. We set w_i to be 1 at the first iteration and $1.0/(1 + d^2(\mathbf{p}_i, \mathbf{g}_i))$ in the subsequent iterations. The reason for using weights is that the reconstruction from images is noisy and such a weight scheme is an effective way to avoid overfitting to the noisy data [8]. Since both $\mathbf{g}_i$ and $\mathbf{v}_{m_j}$ are linear functions of the metric coefficients for fixed $\mathbf{T}$, the above problem is a linear least square problem. We then fix the metric coefficients, and solve for the pose. To do that, we recompute $\mathbf{g}_i$ using the new metric coefficients. Given a set of 3D corresponding points $(\mathbf{p}_i, \mathbf{g}_i)$ and $(\mathbf{q}_j, \mathbf{v}_{m_j})$, there are well known algorithms to solve for the pose. We use the quaternion-based technique described in [14].

To initialize this iterative process, we first use the 5 markers to compute an initial estimate of the pose. In addition, to get a reasonable estimate of the head size, we solve for the head-size related metric coefficients such that the resulting face mesh matches the bounding box of the reconstructed 3D points. Occasionally the corner matching algorithm may produce points not on the face. In that case, the metric coefficients will be out of the valid ranges, and we throw away the point that is the most distant from the center of the face. We repeat this process until metric coefficients become valid.

6.3.2 Fine adjustment using image information

After the geometric fitting process, we have now a face mesh that is a close approximation to the real face. To further improve the result, we search for silhouettes and other face features in the images and use them to refine the face geometry. The general problem of locating silhouettes and face features in images is difficult, and is still a very active research area in computer vision. However, the face mesh that we have obtained provides a good estimate of the locations of the face features, so we only need to perform search in a small region.

We use the snake approach [18] to compute the silhouettes of the face. The silhouette of the current face mesh is used as the initial estimate. For each point on this piecewise linear curve, we find the

maximum gradient location along the normal direction within a small range (10 pixels each side in our implementation). Then we solve for the vertexes (acting as control points) to minimize the total distance between all the points and their corresponding maximum gradient locations.

We use a similar approach to find the upper lips.

To find the outer eye corner (not marked), we rotate the current estimate of that eye corner (given by the face mesh) around the marked eye corner by a small angle, and look for the eye boundary using image gradient information. This is repeated for several angles, and the boundary point that is the most distant to the marked corner is chosen as the outer eye corner.

We could also use the snake approach to search for eyebrows. However, our current implementation uses a slightly different approach. Instead of maximizing image gradients across contours, we minimize the average intensity of the image area that is covered by the eyebrow triangles. Again, the vertices of the eyebrows are only allowed to move in a small region bounded by their neighboring vertices. This has worked very robustly in our experiments.

We then use the face features and the image silhouettes as constraints in our system to further improve the mesh. Notice that each vertex on the mesh silhouette corresponds to a vertex on the image silhouette. We cast a ray from the camera center through the vertex on the image silhouette. The projection of the corresponding mesh vertex on this ray acts as the target position of the mesh vertex. Let $\mathbf{v}$ be the mesh vertex and $\mathbf{h}$ the projection. We have equation $\mathbf{v} = \mathbf{h}$. For each face feature, we obtain an equation in a similar way. These equations are added to equation (5). The total set of equations is solved as before, i.e., we first fix the pose $\mathbf{T}$ and use a linear least square approach to solve the metric coefficients, and then fix the metric coefficients while solving for the pose.

Figure 9 shows the reconstructed 3D face mesh from the two example images (see Figure 3). The mesh is projected back to the two images. Figure 10 shows two novel views using the first image as the texture map. The texture corresponding to the right side of the face is still missing.

Figure 9: The constructed 3D face mesh is projected back to the two base images.

7 Face Texture From Video Sequence

Now we have the geometry of the face from only two views that are close to the frontal position. For the sides of the face, the texture from the two images is therefore quite poor or even not available at

Figure 10: Two novel views of the reconstructed 3D face mesh with the the first base image as texture.

all. Since each image only covers a portion of the face, we need to combine all the images in the video sequence to obtain a complete texture map. This is done by first determining the head pose for the images in the video sequence and then blending them to create a complete texture map.

7.1 Determining head motions in video sequences

Successive images are first matched using the same technique as described in Section 6.2. We could combine the resulting motions incrementally to determine the head pose. However, this estimation is quite noisy because it is computed only from 2D points. As we already have the 3D face geometry, a more reliable pose estimation can be obtained by combining both 3D and 2D information, as follows.

Let us denote the first base image by I_0, the images on the video sequences by $I_1, \ldots, I_v$, the relative head motion from I_{i-1} to I_i by $\mathcal{R}_i = \begin{pmatrix} \mathbf{R}_{ri} & \mathbf{t}_{ri} \\ \mathbf{0}^T & 1 \end{pmatrix}$, and the head pose corresponding to image I_i with respect to the camera frame by Ω_i. The algorithm works incrementally, starting with I_0 and I_1. For each pair of images (I_{i-1}, I_i), we first use the corner matching algorithm described in Section 6.2 to find a set of matched corner pairs $\{(\mathbf{m}_j, \mathbf{m}'_j) | j = 1, \ldots, l\}$. For each $\mathbf{m}_j$ in I_{i-1}, we cast a ray from the camera center through $\mathbf{m}_j$, and compute the intersection $\mathbf{x}_j$ of that ray with the face mesh corresponding to image I_{i-1}. According to (1), $\mathcal{R}_i$ is subject to the following equations

$$\mathbf{A}\mathbf{P}\mathcal{R}_i\tilde{\mathbf{x}}_j = \lambda_j\tilde{\mathbf{m}}'_j \quad \text{for } j = 1, \ldots, l \tag{6}$$

where $\mathbf{A}$, $\mathbf{P}$, $\mathbf{x}_j$ and $\mathbf{m}'_j$ are known. Each of the above equations gives two constraints on $\mathcal{R}_i$. We compute $\mathcal{R}_i$ with a technique described in [7]. After $\mathcal{R}_i$ is computed, the head pose for image I_i in the camera frame is given by $\Omega_i = \mathcal{R}_i\Omega_{i-1}$. The head pose Ω_0 is known from Section 6.3.

In general, it is inefficient to use all the images in the video sequence for texture blending, because head motion between two consecutive frames is usually very small. To avoid unnecessary computation, the following process is used to automatically select images from the video sequence. Let us call the amount of rotation of the head between two consecutive frames the *rotation speed*. If s is the current rotation speed and α is the desired angle between each pair of selected images, the next image is selected $\lfloor(\alpha/s)\rfloor$ frames away. In our implementation, the initial guess of the rotation speed is set to 1 degree/frame and the desired separation angle is equal to 5 degrees.

Figure 11 and Figure 12 show the tracking results of the two example video sequences (The two base images are shown in Figure 3). The images from each video sequence are automatically selected using the above algorithm.

Figure 11: The face mesh is projected back to the automatically selected images from the video sequence where the head turns to the left .

7.1.1 Texture blending

After the head pose of an image is computed, we use an approach similar to Pighin et al.'s method [29] to generate a view independent texture map. We also construct the texture map on a virtual cylinder enclosing the face model. But instead of casting a ray from each pixel to the face mesh and computing the texture blending weights on a pixel by pixel basis, we use a more efficient approach. For each vertex on the face mesh, we compute the blending weight for each image based on the angle between surface normal and the camera direction [29]. If the vertex is invisible, its weight is set to 0.0. The weights are then normalized so that the sum of the weights over all the images is equal to 1.0. We then set the colors of the vertexes to be their weights, and use the rendered image of the cylindrical mapped

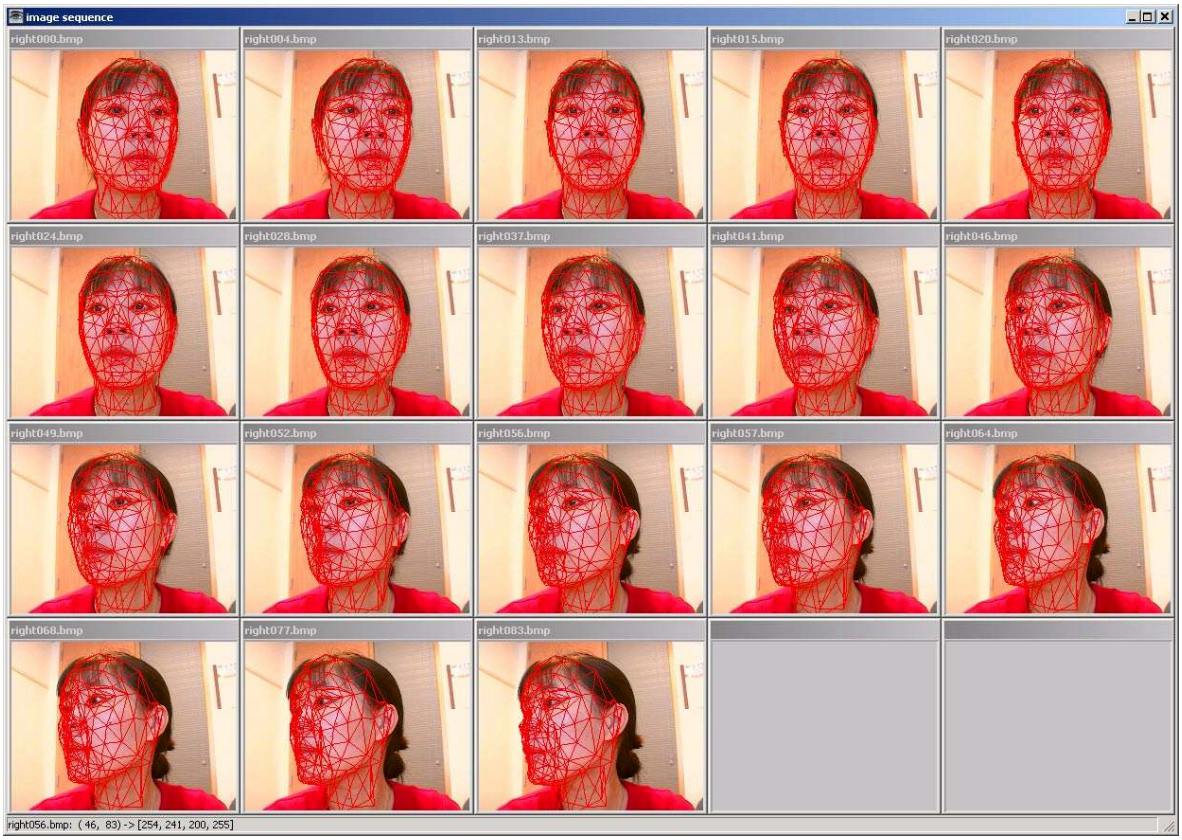


Figure 12: The face mesh is projected back to the automatically selected images from the video sequence where the head turns to the right .

mesh as the weight map. For each image, we also generate a cylindrical texture map by rendering the cylindrical mapped mesh with the current image as texture map. Let C_i and W_i ($i = 1, \ldots, k$) be the cylindrical texture maps and the weight maps. Let C be the final blended texture map. For each pixel (u, v), its color on the final blended texture map is

$$C(u, v) = \sum_{i=1}^{k} W_i(u, v) C_i(u, v). \tag{7}$$

Because the rendering operations can be done using graphics hardware, this approach is very fast.

Figure 13shows the blended texture map from the example video sequences 11 and 12. Figure 15 shows the a side-by-side comparison of the original images with the reconstructed model.

8 User Interface

We have built a user interface to guide the user through collecting the required images and video sequences, and marking two images. The generic head model without texture is used as a guide.

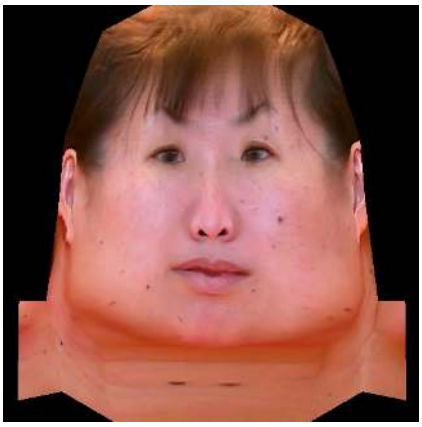
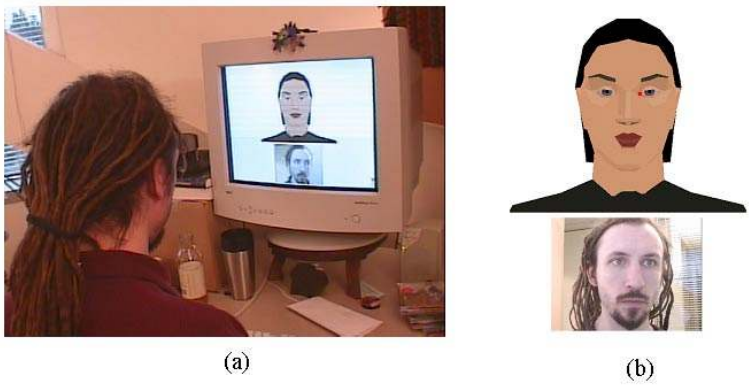


Figure 13: The blended texture image.



(a) (b)

Figure 14: (a) The user interface. (b)The user is guided to put marks on the images.

Recorded instructions are lip-synced with the head directing the user to first look at a dot on the screen and push a key to take a picture (see Figure 14(a) and the accompanying video). A second dot appears and the user is asked to take the second still image. The synthetic face mimics the actions the user is to follow. After the two still images are taken, the guide directs the user to slowly turn his/her head to record the video sequences. Finally, as shown in Figure 14(b), the guide places red dots on her own face and directs the user to do the same on the two still images. The collected images and markings are then processed and a minute or two later they have a synthetic head that resembles them.

9 Animation

Having obtained the 3D textured face model, the user can immediately animate the model with the application of facial expressions including frowns, smiles, mouth open, etc.

To accomplish this we have defined a set of vectors, which we call *posemes*. Like the metric vectors described previously, posemes are a collection of artist-designed displacements. We can apply these displacements to any face as long as it has the same topology as the neutral face. Posemes are collected

in a library of actions and expressions.

The idle motions of the head and eye balls are generated using Perlin's noise functions[27, 28].

10 Results

We have used our system to construct face models for various people. Figure 16 shows side-by-side comparisons of seven reconstructed models with the real images. The accompanying video shows the animations of these models. In all these examples, the video sequences were taken using ordinary video camera in people's offices. No special lighting equipment or background was used. After data-capture and marking, the computations take between 1 and 2 minutes to generate the synthetic textured head. Most of this time is spent tracking the video sequences.

For people with hair on the sides or the front of the face, our system will sometimes pick up corner points on the hair and treat them as points on the face. The reconstructed model may be affected by them. For example, the female in Figure 3 has hair on her forehead above her eyebrows. Our system treats the points on the hair as normal points on the face, thus the forehead of the reconstructed model is higher than the real forehead.

In the animations shown in the accompanying video, we have automatically cut out the eye regions and inserted separate geometries for the eye balls. We scale and translate a generic eyeball model. In some cases, the eye textures are modified manually by scaling the color channels of a real eye image to match the face skin colors. We plan to automate this last step shortly.

In spite of its robustness, the system fails sometimes. We have tried our system on thirteen people, and our system failed on two of them. Both people are young females with very smooth skin. There are a few possible causes for failures. If the marks are not placed in the correct order or are poorly placed the system may fail or produce poor results. If the video sequences do not have a similar pose (within about 15 degrees) to the initial still images, the video tracking may fail. Finally, in a few cases, if the skin is so smooth and free of blemishes, the corner point matching may fail. The first problem is simply a UI issue and can be solved. We hope to find a solution soon for the third cause of failure, but for now, we just compliment the beauty of the user.

11 Conclusions

We have developed a system to construct textured 3D face models from video sequences with minimal user intervention. With a few simple clicks by the user, our system quickly generates a person's face model which is animated right away. Our experiments show that our system is able to generate face models for people of different races, of different ages, and with different skin colors. Such a system can be potentially used by an ordinary user at home to make their own face models. These face models can be used, for example, as avatars in computer games, online chatting, virtual conferencing, etc.

12 Perspectives

Very good results obtained with the current system encourage us to improve the system along three directions.

Figure 15: Side by side comparison of the original images with the reconstructed model.

Figure 16: Side by side comparison of the original images with the reconstructed models of various people.

First, we are working at extracting more face features from two images, including the lower lip and nose.

Second, face geometry is currently determined from only two views, and video sequences are used merely for creating a complete face texture. We are confident that a more accurate face geometry can be recovered from the complete video sequences.

Third, the current face mesh is very sparse. We are investigating techniques to increase the mesh resolution by using higher resolution face metrics or prototypes. Another possibility is to compute a displacement map for each triangle using color information.

Several researchers in computer vision are working at automatically locating facial features in images [32]. With the advancement of those techniques, a completely automatic face modeling system can be expected, even though it is not a burden to click just five points with our current system.

Additional challenges include automatic generation of eyeballs and eye texture maps, as well as accurate incorporation of hair, teeth, and tongues.

References

[1] T. Akimoto, Y. Suenaga, and R. S. Wallace. Automatic 3d facial models. *IEEE Computer Graphics and Applications*, 13(5):16–22, September 1993.

[2] J. Barron, D. Fleet, and S. Beauchemin. Performance of optical flow techniques. *The International Journal of Computer Vision*, 12(1):43–77, 1994.

[3] V. Blanz and T. Vetter. A morphable model for the synthesis of 3d faces. In *Computer Graphics, Annual Conference Series*, pages 187–194. Siggraph, August 1999.

[4] B. Dariush, S. B. Kang, and K. Waters. Spatiotemporal analysis of face profiles: Detection, segmentation, and registration. In *Proc. of the 3rd International Conference on Automatic Face and Gesture Recognition*, pages 248–253. IEEE, April 1998.

[5] D. DeCarlo, D. Metaxas, and M. Stone. An anthropometric face model using variational techniques. In *Computer Graphics, Annual Conference Series*, pages 67–74. Siggraph, July 1998.

[6] S. DiPaola. Extending the range of facial types. *Journal of Visualization and Computer Animation*, 2(4):129–131, 1991.

[7] O. Faugeras. *Three-Dimensional Computer Vision: a Geometric Viewpoint*. MIT Press, 1993.

[8] P. Fua. Using model-driven bundle-adjustment to model heads from raw video sequences. In *International Conference on Computer Vision*, pages 46–53, Sept. 1999.

[9] P. Fua and C. Miccio. From regular images to animated heads: A least squares approach. In *European Conference on Computer Vision*, pages 188–202, 1998.

[10] P. Fua and C. Miccio. Animated heads from ordinary images: A least-squares approach. *Computer Vision and Image Understanding*, 75(3):247–259, 1999.

[11] P. Fua, R. Plaenkers, and D. Thalmann. From synthesis to analysis: Fitting human animation models to image data. In *Computer Graphics International*, Alberta, Canada, June 1999.

[12] B. Guenter, C. Grimm, D. Wood, H. Malvar, and F. Pighin. Making faces. In *Computer Graphics, Annual Conference Series*, pages 55–66. Siggraph, July 1998.

[13] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. 4th Alvey Vision Conf.*, pages 189–192, 1988.

[14] B. K. Horn. Closed-form Solution of Absolute Orientation using Unit Quaternions. *Journal of the Optical Society A*, 4(4):629–642, Apr. 1987.

[15] B. K. P. Horn and B. G. Schunk. Determining Optical Flow. *Artificial Intelligence*, 17:185–203, 1981.

[16] H. H.S.Ip and L. Yin. Constructing a 3d individualized head model from two orthogonal views. *The Visual Computer*, (12):254–266, 1996.

[17] S. B. Kang and M. Jones. Appearance-based structure from motion using linear classes of 3-d models. *Manuscript*, 1999.

[18] M. Kass, A. Witkin, and D. Terzopoulos. SNAKES: Active contour models. *The International Journal of Computer Vision*, 1:321–332, Jan. 1988.

[19] A. Lanitis, C. J. Taylor, and T. F. Cootes. Automatic interpretation and coding of face images using flexible models. *IEEE Transations on Pattern Analysis and Machine Intelligence*, 19(7):743–756, 1997.

[20] Y. C. Lee, D. Terzopoulos, and K. Waters. Constructing physics-based facial models of individuals. In *Proceedings of Graphics Interface*, pages 1–8, 1993.

[21] Y. C. Lee, D. Terzopoulos, and K. Waters. Realistic modeling for facial animation. In *Computer Graphics, Annual Conference Series*, pages 55–62. SIGGRAPH, 1995.

[22] J. P. Lewis. Algorithms for solid noise synthesis. In *Computer Graphics, Annual Conference Series*, pages 263–270. Siggraph, 1989.

[23] N. Magneneat-Thalmann, H. Minh, M. Angelis, and D. Thalmann. Design, transformation and animation of human faces. *Visual Computer*, (5):32–39, 1989.

[24] F. I. Parke. Computer generated animation of faces. In *ACM National Conference*, November 1972.

[25] F. I. Parke. *A Parametric Model of human Faces*. PhD thesis, University of Utah, 1974.

[26] F. I. Parke and K. Waters. *Computer Facial Animation*. AKPeters, Wellesley, Massachusetts, 1996.

[27] K. Perlin. Real time responsive animation with personality. *IEEE Transations on Visualization and Computer Graphics*, 1(1), 1995.

[28] K. Perlin and A. Goldberg. Improv: A system for scripting interactive actors in virtual worlds. In *Computer Graphics, Annual Conference Series*, pages 205–216. Siggraph, August 1995.

[29] F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D. H. Salesin. Synthesizing realistic facial expressions from photographs. In *Computer Graphics, Annual Conference Series*, pages 75–84. Siggraph, July 1998.

[30] S. Platt and N. Badler. Animating facial expression. *Computer Graphics*, 15(3):245–252, 1981.

[31] P. Rousseeuw and A. Leroy. *Robust Regression and Outlier Detection*. John Wiley & Sons, New York, 1987.

[32] T. Shakunaga, K. Ogawa, and S. Oki. Integration of eigentemplate and structure matching for automatic facial feature detection. In *Proc. of the 3rd International Conference on Automatic Face and Gesture Recognition*, pages 94–99, April 1998.

[33] D. Terzopoulos and K. Waters. Physically based facial modeling, analysis, and animation. In *Visualization and Computer Animation*, pages 73–80, 1990.

[34] J. T. Todd, S. M. Leonard, R. E. Shaw, and J. B. Pittenger. The perception of human growth. *Scientific American*, (1242):106–114, 1980.

[35] T. Vetter and T. Poggio. Linear object classes and image synthesis from a single example image. *IEEE Transations on Pattern Analysis and Machine Intelligence*, 19(7):733–742, 1997.

[36] K. Waters. A muscle model for animating three-dimensional facial expression. *Computer Graphics*, 22(4):17–24, 1987.

[37] Z. Zhang. Motion and structure from two perspective views: From essential parameters to euclidean motion via fundamental matrix. *Journal of the Optical Society of America A*, 14(11):2938–2950, 1997.

[38] Z. Zhang. Determining the epipolar geometry and its uncertainty: A review. *The International Journal of Computer Vision*, 27(2):161–195, 1998.

[39] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *International Conference on Computer Vision (ICCV'99)*, pages 666–673, 1999.

[40] J. Y. Zheng. Acquiring 3-d models from sequences of contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):163–178, February 1994.