Synthesizing Realistic Facial Expressions from Photographs

Frédéric Pighin *Jamie Hecker* *Dani Lischinski*[†] *Richard Szeliski*[‡] *David H. Salesin*

University of Washington [†]The Hebrew University [‡]Microsoft Research

Abstract

We present new techniques for creating photorealistic textured 3D facial models from photographs of a human subject, and for creating smooth transitions between different facial expressions by morphing between these different models. Starting from several uncalibrated views of a human subject, we employ a user-assisted technique to recover the camera poses corresponding to the views as well as the 3D coordinates of a sparse set of chosen locations on the subject's face. A scattered data interpolation technique is then used to deform a generic face mesh to fit the particular geometry of the subject's face. Having recovered the camera poses and the facial geometry, we extract from the input images one or more texture maps for the model. This process is repeated for several facial expressions of a particular subject. To generate transitions between these facial expressions we use 3D shape morphing between the corresponding face models, while at the same time blending the corresponding textures. Using our technique, we have been able to generate highly realistic face models and natural looking animations.

CR Categories: I.2.10 [Artificial Intelligence]: Vision and Scene Understanding — Modeling and recovery of physical attributes; I.3.7 [Computer Graphics]: Three-Dimensional Graphics — Animation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics — Color, shading, shadowing and texture.

Additional Keywords: facial modeling, facial expression generation, facial animation, photogrammetry, morphing, view-dependent texture-mapping

1 Introduction

> *There is no landscape that we know as well as the human face. The twenty-five-odd square inches containing the features is the most intimately scrutinized piece of territory in existence, examined constantly, and carefully, with far more than an intellectual interest. Every detail of the nose, eyes, and mouth, every regularity in proportion, every variation from one individual to the next, are matters about which we are all authorities.*
>
> — Gary Faigin [14],
> from *The Artist's Complete Guide to Facial Expression*

Realistic facial synthesis is one of the most fundamental problems in computer graphics — and one of the most difficult. Indeed, attempts to model and animate realistic human faces date back to the early 70's [34], with many dozens of research papers published since. The applications of facial animation include such diverse fields as character animation for films and advertising, computer games [19], video teleconferencing [7], user-interface agents and avatars [44], and facial surgery planning [23, 45]. Yet no perfectly realistic facial animation has ever been generated by computer: no "facial animation Turing test" has ever been passed.

There are several factors that make realistic facial animation so elusive. First, the human face is an extremely complex geometric form. For example, the human face models used in Pixar's *Toy Story* had several thousand control points each [10]. Moreover, the face exhibits countless tiny creases and wrinkles, as well as subtle variations in color and texture — all of which are crucial for our comprehension and appreciation of facial expressions. As difficult as the face is to model, it is even more problematic to animate, since facial movement is a product of the underlying skeletal and muscular forms, as well as the mechanical properties of the skin and subcutaneous layers (which vary in thickness and composition in different parts of the face). All of these problems are enormously magnified by the fact that we as humans have an uncanny ability to read expressions — an ability that is not merely a learned skill, but part of our deep-rooted instincts. For facial expressions, the slightest deviation from truth is something any person will immediately detect.

A number of approaches have been developed to model and animate realistic facial expressions in three dimensions. (The reader is referred to the recent book by Parke and Waters [36] for an excellent survey of this entire field.) Parke's pioneering work introduced simple geometric interpolation between face models that were digitized by hand [34]. A radically different approach is performance-based animation, in which measurements from real actors are used to drive synthetic characters [4, 13, 47]. Today, face models can also be obtained using laser-based cylindrical scanners, such as those produced by Cyberware [8]. The resulting range and color data can be fitted with a structured face mesh, augmented with a physically-based model of skin and muscles [29, 30, 43, 46]. The animations produced using these face models represent the state-of-the-art in automatic physically-based facial animation.

For sheer photorealism, one of the most effective approaches to date has been the use of 2D morphing between photographic images [3]. Indeed, some remarkable results have been achieved in this way — most notably, perhaps, the Michael Jackson video produced by PDI, in which very different-looking actors are seemingly transformed into one another as they dance. The production of this video, however, required animators to painstakingly specify a few dozen carefully chosen correspondences between physical features of the actors in almost every frame. Another problem with 2D image morphing is that it does not correctly account for changes in viewpoint or object pose. Although this shortcoming has been recently addressed by a technique called "view morphing" [39], 2D morphing still lacks some of the advantages of a 3D model, such as the complete freedom of viewpoint and the ability to composite the image with other 3D graphics. Morphing has also been applied in 3D: Chen *et al.* [6] applied Beier and Neely's 2D morphing technique [3] to morph between cylindrical laser scans of human heads. Still, even in this case the animator must specify correspondences for every pair of expressions in order to produce a transition between them. More recently,

Bregler *et al.* [5] used morphing of mouth regions to lip-synch existing video to a novel sound-track.

In this paper, we show how 2D morphing techniques can be combined with 3D transformations of a geometric model to automatically produce 3D facial expressions with a high degree of realism. Our process consists of several basic steps. First, we capture multiple views of a human subject (with a given facial expression) using cameras at arbitrary locations. Next, we digitize these photographs and manually mark a small set of initial corresponding points on the face in the different views (typically, corners of the eyes and mouth, tip of the nose, etc.). These points are then used to automatically recover the camera parameters (position, focal length, etc.) corresponding to each photograph, as well as the 3D positions of the marked points in space. The 3D positions are then used to deform a generic 3D face mesh to fit the face of the particular human subject. At this stage, additional corresponding points may be marked to refine the fit. Finally, we extract one or more texture maps for the 3D model from the photos. Either a single view-independent texture map can be extracted, or the original images can be used to perform view-dependent texture mapping. This whole process is repeated for the same human subject, with several different facial expressions. To produce facial animations, we interpolate between two or more different 3D models constructed in this way, while at the same time blending the textures. Since all the 3D models are constructed from the same generic mesh, there is a natural correspondence between all geometric points for performing the morph. Thus, transitions between expressions can be produced entirely automatically once the different face models have been constructed, without having to specify pairwise correspondences between any of the expressions.

Our modeling approach is based on photogrammetric techniques in which images are used to create precise geometry [31, 40]. The earliest such techniques applied to facial modeling and animation employed grids that were drawn directly on the human subject's face [34, 35]. One consequence of these grids, however, is that the images used to construct geometry can no longer be used as valid texture maps for the subject. More recently, several methods have been proposed for modeling the face photogrammetrically without the use of grids [20, 24]. These modeling methods are similar in concept to the modeling technique described in this paper. However, these previous techniques use a small predetermined set of features to deform the generic face mesh to the particular face being modeled, and offer no mechanism to further improve the fit. Such an approach may perform poorly on faces with unusual features or other significant deviations from the normal. Our system, by contrast, gives the user complete freedom in specifying the correspondences, and enables the user to refine the initial fit as needed. Another advantage of our technique is its ability to handle fairly arbitrary camera positions and lenses, rather than using a fixed pair that are precisely oriented. Our method is similar, in concept, to the work done in architectural modeling by Debevec *et al.* [9], where a set of annotated photographs are used to model buildings starting from a rough description of their shape. Compared to facial modeling methods that utilize a laser scanner, our technique uses simpler acquisition equipment (regular cameras), and it is capable of extracting texture maps of higher resolution. (Cyberware scans typically produce a cylindrical grid of 512 by 256 samples). The price we pay for these advantages is the need for user intervention in the modeling process.

We employ our system not only for creating realistic face models, but also for performing realistic transitions between different expressions. One advantage of our technique, compared to more traditional animatable models with a single texture map, is that we can capture the subtle changes in illumination and appearance (e.g., facial creases) that occur as the face is deformed. This degree of realism is difficult to achieve even with physically-based models, because of the complexity of skin folding and the difficulty of simulating interreflections and self-shadowing [18, 21, 32].

This paper also presents several new expression synthesis techniques based on extensions to the idea of morphing. We develop a morphing technique that allows for different regions of the face to have different "percentages" or "mixing proportions" of facial expressions. We also introduce a painting interface, which allows users to locally add in a little bit of an expression to an existing composite expression. We believe that these novel methods for expression generation and animation may be more natural for the average user than more traditional animation systems, which rely on the manual adjustments of dozens or hundreds of control parameters.

The rest of this paper is organized as follows. Section 2 describes our method for fitting a generic face mesh to a collection of simultaneous photographs of an individual's head. Section 3 describes our technique for extracting both view-dependent and view-independent texture maps for photorealistic rendering of the face. Section 4 presents the face morphing algorithm that is used to animate the face model. Section 5 describes the key aspects of our system's user interface. Section 6 presents the results of our experiments with the proposed techniques, and Section 7 offers directions for future research.

2 Model fitting

The task of the model-fitting process is to adapt a generic face model to fit an individual's face and facial expression. As input to this process, we take several images of the face from different viewpoints (Figure 1a) and a generic face model (we use the generic face model created with Alias|Wavefront [2] shown in Figure 1c). A few features points are chosen (13 in this case, shown in the frames of Figure 1a) to recover the camera pose. These same points are also used to refine the generic face model (Figure 1d). The model can be further refined by drawing corresponding curves in the different views (Figure 1b). The output of the process is a face model that has been adapted to fit the face in the input images (Figure 1e), along with a precise estimate of the camera pose corresponding to each input image.

The model-fitting process consists of three stages. In the *pose recovery* stage, we apply computer vision techniques to estimate the viewing parameters (position, orientation, and focal length) for each of the input cameras. We simultaneously recover the 3D coordinates of a set of *feature points* on the face. These feature points are selected interactively from among the face mesh vertices, and their positions in each image (where visible) are specified by hand. The *scattered data interpolation* stage uses the estimated 3D coordinates of the feature points to compute the positions of the remaining face mesh vertices. In the *shape refinement* stage, we specify additional correspondences between facial vertices and image coordinates to improve the estimated shape of the face (while keeping the camera pose fixed).

2.1 Pose recovery

Starting with a rough knowledge of the camera positions (e.g., frontal view, side view, etc.) and of the 3D shape (given by the generic head model), we iteratively improve the pose and the 3D shape estimates in order to minimize the difference between the predicted and observed feature point positions. Our formulation is based on the non-linear least squares structure-from-motion algorithm introduced by Szeliski and Kang [41]. However, unlike the method they describe, which uses the Levenberg-Marquardt algorithm to perform a complete iterative minimization over all of the unknowns simultaneously, we break the problem down into a series of linear least squares problems that can be solved using very simple

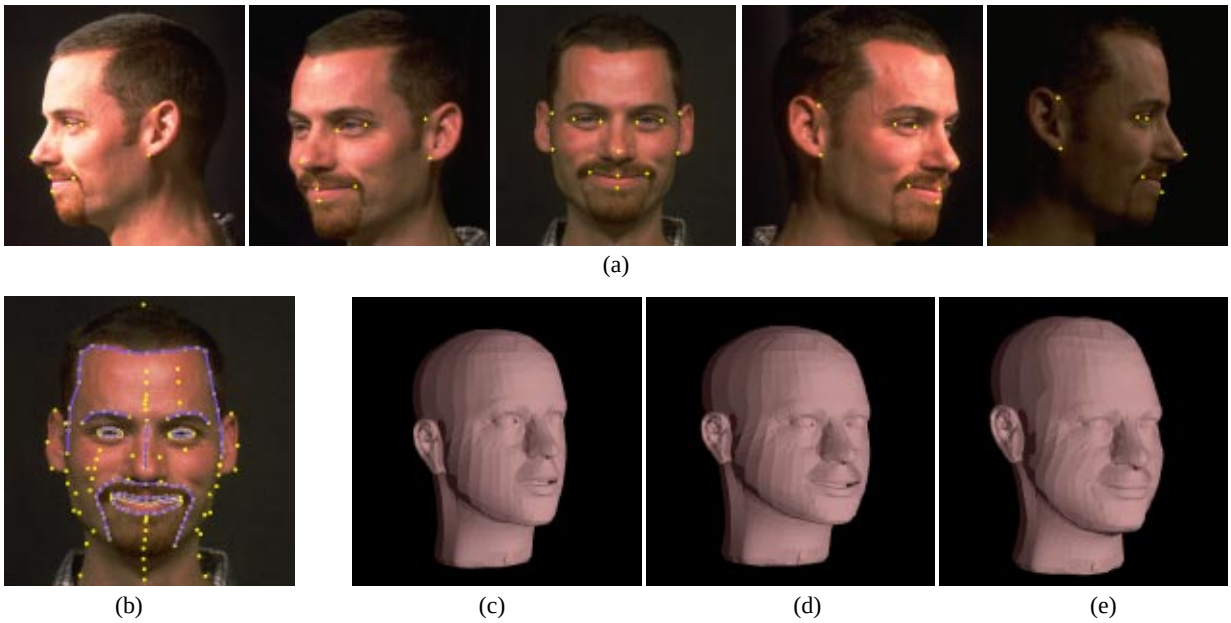


Figure 1 Model-fitting process: (a) a set of input images with marked feature points, (b) facial features annotated using a set of curves, (c) generic face geometry (shaded surface rendering), (d) face adapted to initial 13 feature points (after pose estimation) (e) face after 99 additional correspondences have been given.

and numerically stable techniques [16, 37].

To formulate the pose recovery problem, we associate a rotation matrix $\boldsymbol{R}^k$ and a translation vector $\boldsymbol{t}^k$ with each camera pose k. (The three rows of $\boldsymbol{R}^k$ are $\boldsymbol{r}_x^k$, $\boldsymbol{r}_y^k$, and $\boldsymbol{r}_z^k$, and the three entries in $\boldsymbol{t}^k$ are t_x^k, t_y^k, t_z^k.) We write each 3D feature point as $\boldsymbol{p}_i$, and its 2D screen coordinates in the k-th image as (x_i^k, y_i^k).

Assuming that the origin of the (x, y) image coordinate system lies at the optical center of each image (i.e., where the optical axis intersects the image plane), the traditional 3D projection equation for a camera with a focal length f^k (expressed in pixels) can be written as

$$x_i^k = f^k \frac{\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k}{\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i + t_z^k} \qquad y_i^k = f^k \frac{\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k}{\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i + t_z^k} \tag{1}$$

(This is just an explicit rewriting of the traditional projection equation $\boldsymbol{x}_i^k \propto \boldsymbol{R}^k \boldsymbol{p}_i + \boldsymbol{t}^k$ where $\boldsymbol{x}_i^k = (x_i^k, y_i^k, f^k)$.)

Instead of using (1) directly, we reformulate the problem to estimate inverse distances to the object [41]. Let $\eta^k = 1/t_z^k$ be this inverse distance and $s^k = f^k \eta^k$ be a world-to-image scale factor. The advantage of this formulation is that the scale factor s^k can be reliably estimated even when the focal length is long, whereas the original formulation has a strong coupling between the f^k and t_z^k parameters.

Performing these substitution, we obtain

$$x_i^k = s^k \frac{\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k}{1 + \eta^k \boldsymbol{r}_z^k \cdot \boldsymbol{p}_i}$$
$$y_i^k = s^k \frac{\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k}{1 + \eta^k \boldsymbol{r}_z^k \cdot \boldsymbol{p}_i}.$$

If we let $w_i^k = (1 + \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i))^{-1}$ be the inverse denominator, and collect terms on the left-hand side, we get

$$w_i^k \left(x_i^k + x_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) - s^k (\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k) \right) = 0 \tag{2}$$
$$w_i^k \left(y_i^k + y_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) - s^k (\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k) \right) = 0$$

Note that these equations are linear in each of the unknowns that we wish to recover, i.e., $\boldsymbol{p}_i$, t_x^k, t_y^k, η^k, s^k, and $\boldsymbol{R}^k$, if we ignore the variation of w_i^k with respect to these parameters. (The reason we keep the w_i^k term, rather than just dropping it from these equations, is so that the linear equations being solved in the least squares step have the same magnitude as the original measurements (x_i^k, y_i^k). Hence, least-squares will produce a *maximum likelihood* estimate for the unknown parameters [26].)

Given estimates for initial values, we can solve for different subsets of the unknowns. In our current algorithm, we solve for the unknowns in five steps: first s^k, then $\boldsymbol{p}_i$, $\boldsymbol{R}^k$, t_x^k and t_y^k, and finally η^k. This order is chosen to provide maximum numerical stability given the crude initial pose and shape estimates. For each parameter or set of parameters chosen, we solve for the unknowns using linear least squares (Appendix A). The simplicity of this approach is a result of solving for the unknowns in five separate stages, so that the parameters for a given camera or 3D point can be recovered independently of the other parameters.

2.2 Scattered data interpolation

Once we have computed an initial set of coordinates for the feature points $\boldsymbol{p}_i$, we use these values to deform the remaining vertices on the face mesh. We construct a smooth interpolation function that gives the 3D displacements between the original point positions and the new adapted positions for every vertex in the original generic face mesh. Constructing such an interpolation function is a standard problem in scattered data interpolation. Given a set of known displacements $\boldsymbol{u}_i = \boldsymbol{p}_i - \boldsymbol{p}_i^{(0)}$ away from the original positions $\boldsymbol{p}_i^{(0)}$ at every constrained vertex i, construct a function that gives the displacement $\boldsymbol{u}_j$ for every unconstrained vertex j.

There are several considerations in choosing the particular data interpolant [33]. The first consideration is the embedding space, that is, the domain of the function being computed. In our case, we use the original 3D coordinates of the points as the domain. (An alternative would be to use some 2D parameterization of the surface mesh, for instance, the cylindrical coordinates described in Section 3.) We therefore attempt to find a smooth vector-valued function $\boldsymbol{f}(\boldsymbol{p})$ fitted

to the known data $\boldsymbol{u}_i = \boldsymbol{f}(\boldsymbol{p}_i)$, from which we can compute $\boldsymbol{u}_j = \boldsymbol{f}(\boldsymbol{p}_j)$.

There are also several choices for how to construct the interpolating function [33]. We use a method based on *radial basis functions*, that is, functions of the form

$$\boldsymbol{f}(\boldsymbol{p}) = \sum_i \boldsymbol{c}_i \phi(\|\boldsymbol{p} - \boldsymbol{p}_i\|),$$

where $\phi(r)$ are radially symmetric basis functions. A more general form of this interpolant also adds some low-order polynomial terms to model global, e.g., affine, deformations [27, 28, 33]. In our system, we use an affine basis as part of our interpolation algorithm, so that our interpolant has the form:

$$\boldsymbol{f}(\boldsymbol{p}) = \sum_i \boldsymbol{c}_i \phi(\|\boldsymbol{p} - \boldsymbol{p}_i\|) + \boldsymbol{M}\boldsymbol{p} + \boldsymbol{t}, \quad (3)$$

To determine the coefficients $\boldsymbol{c}_i$ and the affine components $\boldsymbol{M}$ and $\boldsymbol{t}$, we solve a set of linear equations that includes the interpolation constraints $\boldsymbol{u}_i = \boldsymbol{f}(\boldsymbol{p}_i)$, as well as the constraints $\sum_i \boldsymbol{c}_i = 0$ and $\sum_i \boldsymbol{c}_i \boldsymbol{p}_i^{\mathrm{T}} = 0$, which remove affine contributions from the radial basis functions.

Many different functions for $\phi(r)$ have been proposed [33]. After experimenting with a number of functions, we have chosen to use $\phi(r) = e^{-r/64}$, with units measured in inches.

Figure 1d shows the shape of the face model after having interpolated the set of computed 3D displacements at 13 feature points shown in Figure 1 and applied them to the entire face.

2.3 Correspondence-based shape refinement

After warping the generic face model into its new shape, we can further improve the shape by specifying additional correspondences. Since these correspondences may not be as easy to locate correctly, we do not use them to update the camera pose estimates. Instead, we simply solve for the values of the new feature points $\boldsymbol{p}_i$ using a simple least-squares fit, which corresponds to finding the point nearest the intersection of the viewing rays in 3D. We can then re-run the scattered data interpolation algorithm to update the vertices for which no correspondences are given. This process can be repeated until we are satisfied with the shape.

Figure 1e shows the shape of the face model after 99 additional correspondences have been specified. To facilitate the annotation process, we grouped vertices into polylines. Each polyline corresponds to an easily identifiable facial feature such as the eyebrow, eyelid, lips, chin, or hairline. The features can be annotated by outlining them with hand-drawn curves on each photograph where they are visible. The curves are automatically converted into a set of feature points by stepping along them using an arc-length parametrization. Figure 1b shows annotated facial features using a set of curves on the front view.

3 Texture extraction

In this section we describe the process of extracting the texture maps necessary for rendering photorealistic images of a reconstructed face model from various viewpoints.

The texture extraction problem can be defined as follows. Given a collection of photographs, the recovered viewing parameters, and the fitted face model, compute for each point $\boldsymbol{p}$ on the face model its texture color $T(\boldsymbol{p})$.

Each point $\boldsymbol{p}$ may be visible in one or more photographs; therefore, we must identify the corresponding point in each photograph and decide how these potentially different values should be combined

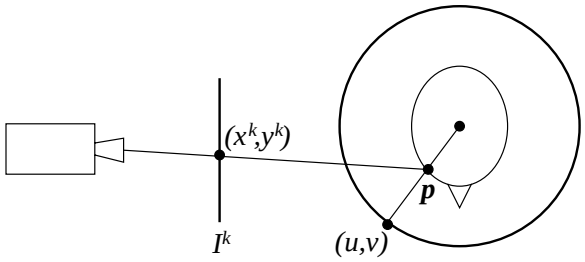


Figure 2 Geometry for texture extraction

(blended) together. There are two principal ways to blend values from different photographs: *view-independent blending*, resulting in a texture map that can be used to render the face from any viewpoint; and *view-dependent blending*, which adjusts the blending weights at each point based on the direction of the current viewpoint [9, 38]. Rendering takes longer with view-dependent blending, but the resulting image is of slightly higher quality (see Figure 3).

3.1 Weight maps

As outlined above, the texture value $T(\boldsymbol{p})$ at each point on the face model can be expressed as a convex combination of the corresponding colors in the photographs:

$$T(\boldsymbol{p}) = \frac{\sum_k m^k(\boldsymbol{p})\, I^k(x^k, y^k)}{\sum_k m^k(\boldsymbol{p})}.$$

Here, I^k is the image function (color at each pixel of the k-th photograph,) and (x^k, y^k) are the image coordinates of the projection of $\boldsymbol{p}$ onto the k-th image plane. The *weight map* $m^k(\boldsymbol{p})$ is a function that specifies the contribution of the k-th photograph to the texture at each facial surface point.

The construction of these weight maps is probably the trickiest and the most interesting component of our texture extraction technique. There are several important considerations that must be taken into account when defining a weight map:

1. *Self-occlusion:* $m^k(\boldsymbol{p})$ should be zero unless $\boldsymbol{p}$ is front-facing with respect to the k-th image and visible in it.
2. *Smoothness:* the weight map should vary smoothly, in order to ensure a seamless blend between different input images.
3. *Positional certainty:* $m^k(\boldsymbol{p})$ should depend on the "positional certainty" [24] of $\boldsymbol{p}$ with respect to the k-th image. The positional certainty is defined as the dot product between the surface normal at $\boldsymbol{p}$ and the k-th direction of projection.
4. *View similarity:* for view-dependent texture mapping, the weight $m^k(\boldsymbol{p})$ should also depend on the angle between the direction of projection of $\boldsymbol{p}$ onto the j-th image and its direction of projection in the new view.

Previous authors have taken only a subset of these considerations into account when designing their weighting functions. For example, Kurihara and Arai [24] use positional certainty as their weighting function, but they do not account for self-occlusion. Akimoto *et al.* [1] and Ip and Yin [20] blend the images smoothly, but address neither self-occlusion nor positional certainty. Debevec *et al.* [9], who describe a view-dependent texture mapping technique for modeling and rendering buildings from photographs, do address occlusion but do not account for positional certainty. (It should be noted, however, that positional certainty is less critical in photographs of buildings, since most buildings do not tend to curve away from the camera.)

To facilitate fast visibility testing of points on the surface of the face from a particular camera pose, we first render the face model using the recovered viewing parameters and save the resulting depth map from the Z-buffer. Then, with the aid of this depth map, we can quickly classify the visibility of each facial point by applying the viewing transformation and comparing the resulting depth to the corresponding value in the depth map.

3.2 View-independent texture mapping

In order to support rapid display of the textured face model from any viewpoint, it is desirable to blend the individual photographs together into a single texture map. This texture map is constructed on a virtual cylinder enclosing the face model. The mapping between the 3D coordinates on the face mesh and the 2D texture space is defined using a cylindrical projection, as in several previous papers [6, 24, 29].

For view-independent texture mapping, we will index the weight map m^k by the (u, v) coordinates of the texture being created. Each weight $m^k(u, v)$ is determined by the following steps:

1. Construct a feathered visibility map F^k for each image k. These maps are defined in the same cylindrical coordinates as the texture map. We initially set $F^k(u, v)$ to 1 if the corresponding facial point $\boldsymbol{p}$ is visible in the k-th image, and to 0 otherwise. The result is a binary visibility map, which is then smoothly ramped (feathered) from 1 to 0 in the vicinity of the boundaries [42]. A cubic polynomial is used as the ramping function.
2. Compute the 3D point $\boldsymbol{p}$ on the surface of the face mesh whose cylindrical projection is (u, v) (see Figure 2). This computation is performed by casting a ray from (u, v) on the cylinder towards the cylinder's axis. The first intersection between this ray and the face mesh is the point $\boldsymbol{p}$. (Note that there can be more than one intersection for certain regions of the face, most notably the ears. These special cases are discussed in Section 3.4.) Let $P^k(\boldsymbol{p})$ be the positional certainty of $\boldsymbol{p}$ with respect to the k-th image.
3. Set weight $m^k(u, v)$ to the product $F^k(u, v)\, P^k(\boldsymbol{p})$.

For view-independent texture mapping, we will compute each pixel of the resulting texture $T(u, v)$ as a weighted sum of the original image functions, indexed by (u, v).

3.3 View-dependent texture mapping

The main disadvantage of the view-independent cylindrical texture map described above is that its construction involves blending together resampled versions of the original images of the face. Because of this resampling, and also because of slight registration errors, the resulting texture is slightly blurry. This problem can be alleviated to a large degree by using a view-dependent texture map [9] in which the blending weights are adjusted dynamically, according to the current view.

For view-dependent texture mapping, we render the model several times, each time using a different input photograph as a texture map, and blend the results. More specifically, for each input photograph, we associate texture coordinates and a blending weight with each vertex in the face mesh. (The rendering hardware performs perspective-correct texture mapping along with linear interpolation of the blending weights.)

Given a viewing direction $\boldsymbol{d}$, we first select the subset of photographs used for the rendering and then assign blending weights to each of these photographs. Pulli *et al.* [38] select three photographs based on a Delaunay triangulation of a sphere surrounding the object. Since our cameras were positioned roughly in the same plane, we select just the two photographs whose view directions $\boldsymbol{d}^\ell$ and $\boldsymbol{d}^{\ell+1}$ are the closest to $\boldsymbol{d}$ and blend between the two.

Figure 3 Comparison between view-independent (left) and view-dependent (right) texture mapping. Higher frequency details are visible in the view-dependent rendering.

In choosing the view-dependent term $V^k(\boldsymbol{d})$ of the blending weights, we wish to use just a single photo if that photo's view direction matches the current view direction precisely, and to blend smoothly between the nearest two photos otherwise. We used the simplest possible blending function having this effect:

$$V^k(\boldsymbol{d}) = \begin{cases} \boldsymbol{d} \cdot \boldsymbol{d}^k - \boldsymbol{d}^\ell \cdot \boldsymbol{d}^{\ell+1} & \text{if } \ell \le k \le \ell + 1 \\ 0 & \text{otherwise} \end{cases}$$

For the final blending weights $m^k(\boldsymbol{p}, \boldsymbol{d})$, we then use the product of all three terms $F^k(x^k, y^k)\, P^k(\boldsymbol{p})\, V^k(\boldsymbol{d})$.

View-dependent texture maps have several advantages over cylindrical texture maps. First, they can make up for some lack of detail in the model. Second, whenever the model projects onto a cylinder with overlap, a cylindrical texture map will not contain data for some parts of the model. This problem does not arise with view-dependent texture maps if the geometry of the mesh matches the photograph properly. One disadvantage of the view-dependent approach is its higher memory requirements and slower speed due to the multi-pass rendering. Another drawback is that the resulting images are much more sensitive to any variations in exposure or lighting conditions in the original photographs.

3.4 Eyes, teeth, ears, and hair

The parts of the mesh that correspond to the eyes, teeth, ears, and hair are textured in a separate process. The eyes and teeth are usually partially occluded by the face; hence it is difficult to extract a texture map for these parts in every facial expression. The ears have an intricate geometry with many folds and usually fail to project without overlap on a cylinder. The hair has fine-detailed texture that is difficult to register properly across facial expressions. For these reasons, each of these facial elements is assigned an individual texture map. The texture maps for the eyes, teeth, and ears are computed by projecting the corresponding mesh part onto a selected input image where that part is clearly visible (the front view for eyes and teeth, side views for ears).

The eyes and the teeth are usually partially shadowed by the eyelids and the mouth respectively. We approximate this shadowing by modulating the brightness of the eye and teeth texture maps according to the size of the eyelid and mouth openings.

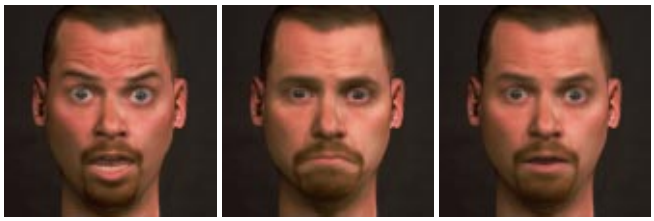


Figure 4 A global blend between "surprised" (left) and "sad" (center) produces a "worried" expression (right).

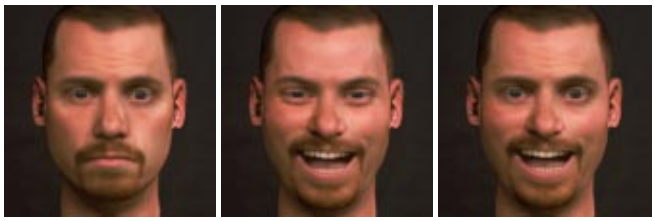


Figure 5 Combining the upper part of a "neutral" expression (left) with the lower part of a "happy" expression (center) produces a "fake smile" (right).

4 Expression morphing

A major goal of this work is the generation of continuous and realistic transitions between different facial expressions. We achieve these effects by morphing between corresponding face models.

In general the problem of morphing between arbitrary polygonal meshes is a difficult one [22], since it requires a set of correspondences between meshes with potentially different topology that can produce a reasonable set of intermediate shapes. In our case, however, the topology of all the face meshes is identical. Thus, there is already a natural correspondence between vertices. Furthermore, in creating the models we attempt to mark facial features consistently across different facial expressions, so that the major facial features correspond to the same vertices in all expressions. In this case, a satisfactory 3D morphing sequence can be obtained using simple linear interpolation between the geometric coordinates of corresponding vertices in each of the two face meshes.

Together with the geometric interpolation, we need to blend the associated textures. Again, in general, morphing between two images requires pairwise correspondences between images features [3]. In our case, however, correspondences between the two textures are implicit in the texture coordinates of the two associated face meshes. Rather than warping the two textures to form an intermediate one, the intermediate face model (obtained by geometric interpolation) is rendered once with the first texture, and again with the second. The two resulting images are then blended together. This approach is faster than warping the textures (which typically have high resolution), and it avoids the resampling that is typically performed during warping.

4.1 Multiway blend and localized blend

Given a set of facial expression meshes, we have explored ways to enlarge this set by combining expressions. The simplest approach is to use the morphing technique described above to create new facial expressions, which can be added to the set. This idea can be generalized to an arbitrary number of starting expressions by taking convex combinations of them all, using weights that apply both to the coordinates of the mesh vertices and to the values in the texture map. (Extrapolation of expressions should also be possible by allowing weights to have values outside of the interval $[0, 1]$; note, however, that such weights might result in colors outside of the allowable gamut.)

We can generate an even wider range of expressions using a localized blend of the facial expressions. Such a blend is specified by a set of blend functions, one for each expression, defined over the vertices of the mesh. These blend functions describe the contribution of a given expression at a particular vertex.

Although it would be possible to compute a texture map for each new expression, doing so would result in a loss of texture quality. Instead, the weights for each new blended expression are always factored into weights over the vertices of the original set of expressions. Thus, each blended expression is rendered using the texture map of an original expression, along with weights at each vertex, which control the opacity of that texture. The opacities are linearly interpolated over the face mesh using Gouraud shading.

4.2 Blend specification

In order to design new facial expressions easily, the user must be provided with useful tools for specifying the blending functions. These tools should satisfy several requirements. First, it should be possible to edit the blend at different resolutions. Moreover, we would like the specification process to be continuous so that small changes in the blend parameters do not trigger radical changes in the resulting expression. Finally, the tools should be intuitive to the user; it should be easy to produce a particular target facial expression from an existing set.

We explored several different ways of specifying the blending weights:

- *Global blend.* The blending weights are constant over all vertices. A set of sliders controls the mixing proportions of the contributing expressions. Figure 4 shows two facial expressions blended in equal proportions to produce a halfway blend.
- *Regional blend.* According to studies in psychology, the face can be split into several regions that behave as coherent units [11]. Usually, three regions are considered: one for the forehead (including the eyebrows), another for the eyes, and another for the lower part of the face. Further splitting the face vertically down the center results in six regions and allows for asymmetric expressions. We similarly partition the face mesh into several (softly feathered) regions and assign weights so that vertices belonging to the same region have the same weights. The mixing proportions describing a selected region can be adjusted by manipulating a set of sliders. Figure 5 illustrates the blend of two facial expressions with two regions: the upper part of the face (including eyes and forehead) and the lower part (including nose, mouth, and chin.)
- *Painterly interface.* The blending weights can be assigned to the vertices using a 3D painting tool. This tool uses a palette in which the "colors" are facial expressions (both geometry and color), and the "opacity" of the brush controls how much the expression contributes to the result. Once an expression is selected, a 3D brush can be used to modify the blending weights in selected areas of the mesh. The fraction painted has a gradual drop-off and is controlled by the opacity of the brush. The strokes are applied directly on the rendering of the current facial blend, which is updated in real-time. To improve the rendering speed, only the portion of the mesh that is being painted is re-rendered. Figure 7 illustrates the design of a debauched smile: starting with a neutral expression, the face is locally modified using three other expressions. Note that in the last step, the use of a partially transparent brush with the "sleepy" expression results in the actual geometry of the eyelids becoming partially lowered.

Figure 6 Animation interface. On the left is the "expression gallery"; on the right an expression is being designed. At the bottom expressions and poses are scheduled on the timeline.

Combining different original expressions enlarges the repertoire of expressions obtained from a set of photographs. The expressions in this repertoire can themselves be blended to create even more expressions, with the resulting expression still being representable as a (locally varying) linear combination of the original expressions.

5 User interface

We designed an interactive tool to fit a 3D face mesh to a set of images. This tool allows a user to select vertices on the mesh and mark where these curves or vertices should project on the images. After a first expression has been modeled, the set of annotations can be used as an initial guess for subsequent expressions. These guesses are automatically refined using standard correlation-based search. Any resulting errors can be fixed up by hand. The extraction of the texture map does not require user intervention, but is included in the interface to provide feedback during the modeling phase.

We also designed a keyframe animation system to generate facial animations. Our animation system permits a user to blend facial expressions and to control the transitions between these different expressions (Figure 6). The expression gallery is a key component of our system; it is used to select and display (as thumbnails) the set of facial expressions currently available. The thumbnails can be dragged and dropped onto the timeline (to set keyframes) or onto the facial design interface (to select or add facial expressions). The timeline is used to schedule the different expression blends and the changes in viewing parameters (pose) during the animation. The blends and poses have two distinct types of keyframes. Both types of keyframes are linearly interpolated with user-controlled cubic Bézier curves. The timeline can also be used to display intermediate frames at low resolution to provide a quick feedback to the animator. A second timeline can be displayed next to the composition timeline. This feature is helpful for correctly synchronizing an animation with live video or a soundtrack. The eyes are animated separately from the rest of the face, with the gaze direction parameterized by two Euler angles.

6 Results

In order to test our technique, we photographed both a man (J. R.) and a woman (Karla) in a variety of facial expressions. The photog-

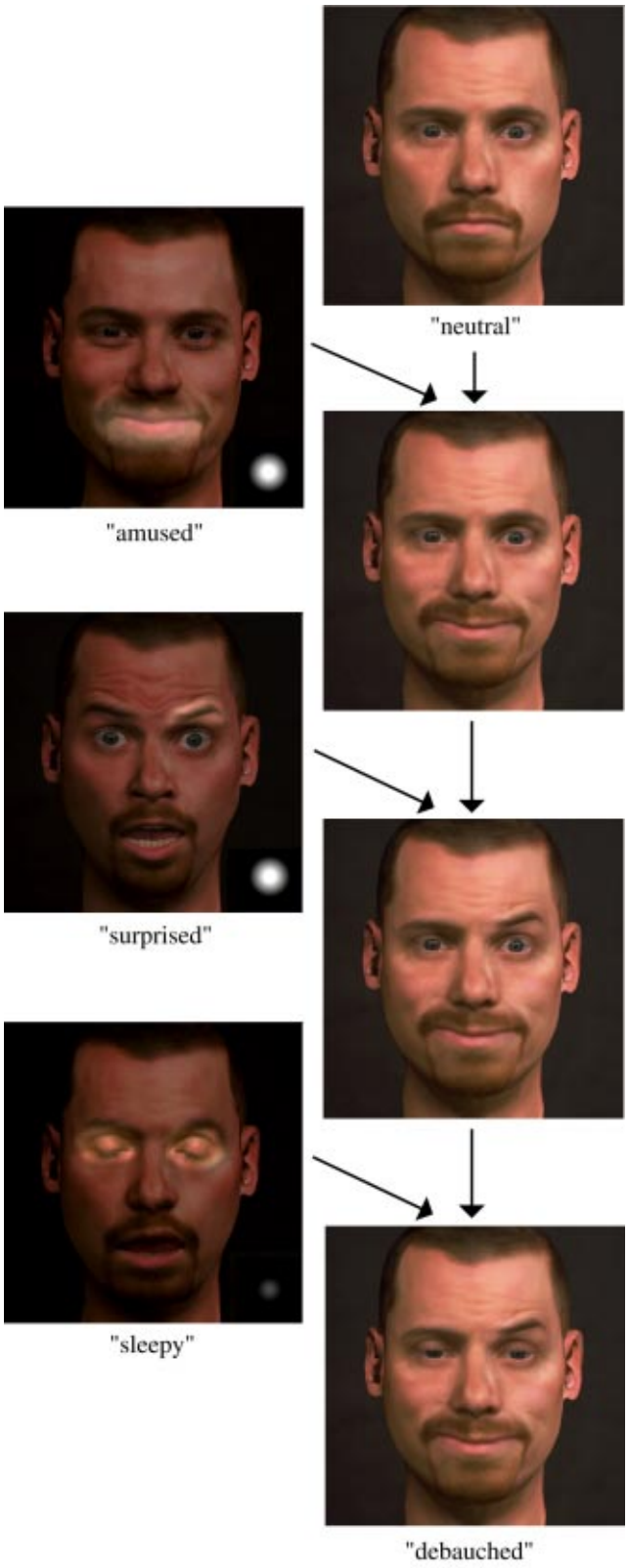


Figure 7 Painterly interface: design of a debauched smile. The right column shows the different stages of the design; the left column shows the portions of the original expressions used in creating the final expression. The "soft brush" used is shown at the bottom-right corner of each contributing expression.

raphy was performed using five cameras simultaneously. The cameras were not calibrated in any particular way, and the lenses had different focal lengths. Since no special attempt was made to illuminate the subject uniformly, the resulting photographs exhibited considerable variation in both hue and brightness. The photographs were digitized using the Kodak PhotoCD process. Five typical images (cropped to the size of the subject's head) are shown in Figure 1a.

We used the interactive modeling system described in Sections 2 and 3 to create the same set of eight face models for each subject: "happy," "amused," "angry," "surprised," "sad," "sleepy," "pained," and "neutral."

Following the modeling stage, we generated a facial animation for each of the individuals starting from the eight original expressions. We first created an animation for J. R. We then applied the very same morphs specified by this animation to the models created for Karla. For most frames of the animation, the resulting expressions were quite realistic. Figure 8 shows five frames from the animation sequence for J. R. and the purely automatically generated frames in the corresponding animation for Karla. With just a small amount of additional retouching (using the blending tools described in Section 4.2), this derivative animation can be made to look as good as the original animation for J. R.

7 Future work

The work described in this paper is just the first step towards building a complete image-based facial modeling and animation system. There are many ways to further enhance and extend the techniques that we have described:

Color correction. For better color consistency in facial textures extracted from photographs, color correction should be applied to simultaneous photographs of each expression.

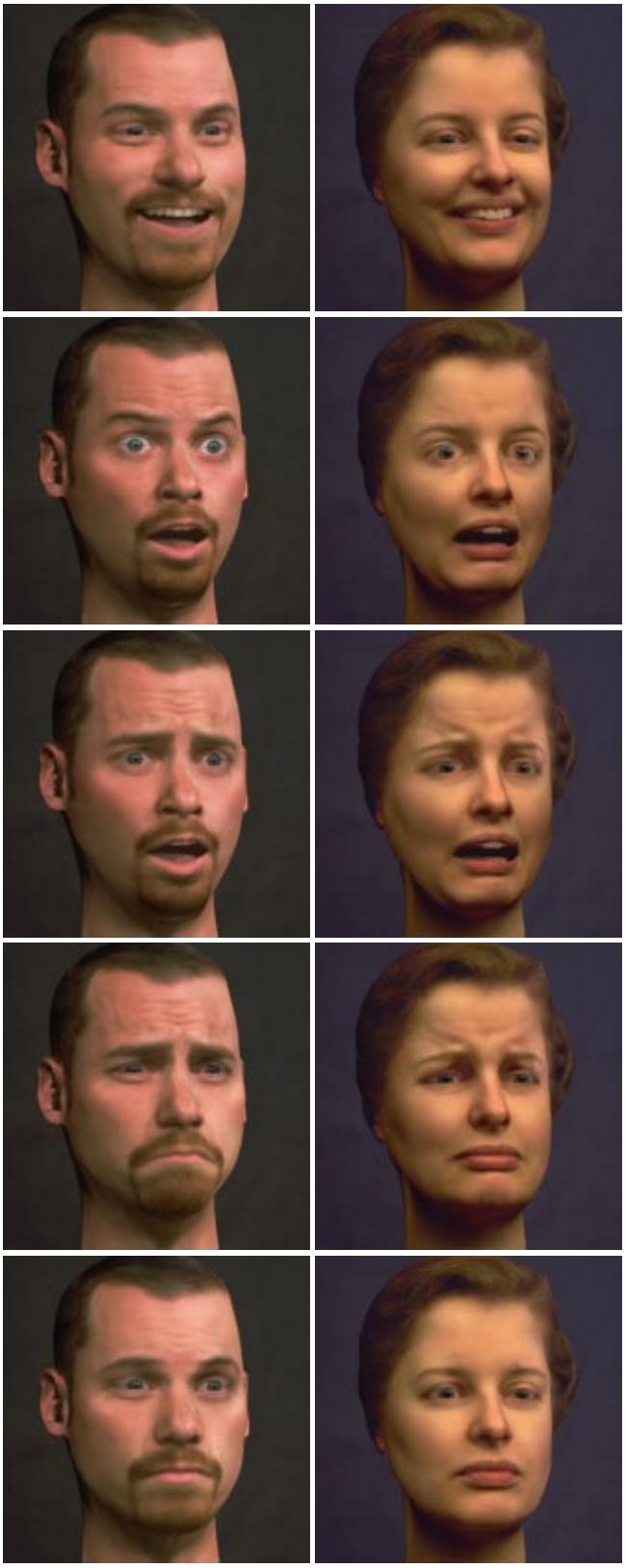
Improved registration. Some residual ghosting or blurring artifacts may occasionally be visible in the cylindrical texture map due to small misregistrations between the images, which can occur if geometry is imperfectly modeled or not detailed enough. To improve the quality of the composite textures, we could locally warp each component texture (and weight) map before blending [42].

Texture relighting. Currently, extracted textures reflect the lighting conditions under which the photographs were taken. Relighting techniques should be developed for seamless integration of our face models with other elements.

Automatic modeling. Our ultimate goal, as far as the facial modeling part is concerned, is to construct a fully automated modeling system, which would automatically find features and correspondences with minimal user intervention. This is a challenging problem indeed, but recent results on 2D face modeling in computer vision [25] give us cause for hope.

Modeling from video. We would like to be able to create face models from video or old movie footage. For this purpose, we would have to improve the robustness of our techniques in order to synthesize face meshes and texture maps from images that do not correspond to different views of the same expression. Adding anthropomorphic constraints to our face model might make up for the lack of coherence in the data [48].

Complex animations. In order to create complex animations, we must extend our vocabulary for describing facial movements beyond blending between different expressions. There are several potential ways to attack this problem. One would be to adopt an action-unit-based system such as the Facial Action Coding System



(a) (b)

Figure 8 On the left are frames from an original animation, which we created for J. R. The morphs specified in these frames were then directly used to create a derivative animation for Karla, shown on the right.

(FACS) [12]. Another possibility would be to apply modal analysis (principal component analysis) techniques to describe facial expression changes using a small number of motions [25]. Finding natural control parameters to facilitate animation and developing realistic-looking temporal profiles for such movements are also challenging research problems.

Lip-synching. Generating speech animation with our keyframe animation system would require a large number of keyframes. However, we could use a technique similar to that of Bregler *et al*. [5] to automatically lip-synch an animation to a sound-track. This would require the synthesis of face models for a wide range of visemes. For example, such database of models could be constructed using video footage to reconstruct face models automatically [17].

Performance-driven animation. Ultimately, we would also like to support performance-driven animation, i.e., the ability to automatically track facial movements in a video sequence, and to automatically translate these into animation control parameters. Our current techniques for registering images and converting them into 3D movements should provide a good start, although they will probably need to be enhanced with feature-tracking techniques and some rudimentary expression-recognition capabilities. Such a system would enable not only very realistic facial animation, but also a new level of video coding and compression techniques (since only the expression parameters would need to be encoded), as well as real-time control of avatars in 3D chat systems.

8 Acknowledgments

We would like to thank Katrin Petersen and Andrew Petty for modeling the generic face model, Cassidy Curtis for his invaluable advice on animating faces, and Joel Auslander and Jason Griffith for early contributions to this project. This work was supported by an NSF Presidential Faculty Fellow award (CCR-9553199), an ONR Young Investigator award (N00014-95-1-0728), and industrial gifts from Microsoft and Pixar.

References

[1] Takaaki Akimoto, Yasuhito Suenaga, and Richard S. Wallace. Automatic Creation of 3D Facial Models. *IEEE Computer Graphics and Applications*, 13(5):16–22, September 1993.

[2] Alias | Wavefront, Toronto, Ontario. *Alias V7.0*, 1995.

[3] Thaddeus Beier and Shawn Neely. Feature-based Image Metamorphosis. In *SIGGRAPH 92 Conference Proceedings*, pages 35–42. ACM SIGGRAPH, July 1992.

[4] Philippe Bergeron and Pierre Lachapelle. Controlling Facial Expressions and Body Movements in the Computer-Generated Animated Short "Tony De Peltrie". In *SIGGRAPH 85 Advanced Computer Animation seminar notes*. July 1985.

[5] Christoph Bregler, Michele Covell, and Malcolm Slaney. Video Rewrite: Driving Visual Speech with Audio. In *SIGGRAPH 97 Conference Proceedings*, pages 353–360. ACM SIGGRAPH, August 1997.

[6] David T. Chen, Andrei State, and David Banks. Interactive Shape Metamorphosis. In *1995 Symposium on Interactive 3D Graphics*, pages 43–44. ACM SIGGRAPH, April 1995.

[7] Chang S. Choi, Kiyoharu, Hiroshi Harashima, and Tsuyoshi Takebe. Analysis and Synthesis of Facial Image Sequences in Model-Based Image Coding. In *IEEE Transactions on Circuits and Systems for Video Technology*, volume 4, pages 257 – 275. June 1994.

[8] Cyberware Laboratory, Inc, Monterey, California. *4020/RGB 3D Scanner with Color Digitizer*, 1990.

[9] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In *SIGGRAPH 96 Conference Proceedings*, pages 11–20. ACM SIGGRAPH, August 1996.

[10] Eben Ostby, Pixar Animation Studios. Personal communication, January 1997.

[11] Paul Ekman and Wallace V. Friesen. *Unmasking the Face. A guide to recognizing emotions fron facial clues*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[12] Paul Ekman and Wallace V. Friesen. *Manual for the Facial Action Coding System*. Consulting Psychologists Press, Inc., Palo Alto, California, 1978.

[13] Irfan Essa, Sumit Basu, Trevor Darrell, and Alex Pentland. Modeling, Tracking and Interactive Animation of Faces and Heads Using Input from Video. In *Computer Animation Conference*, pages 68–79. June 1996.

[14] Gary Faigin. *The Artist's Complete Guide to Facial Expression*. Watson-Guptill Publications, New York, 1990.

[15] Olivier Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Cambridge, Massachusetts, 1993.

[16] G. Golub and C. F. Van Loan. *Matrix Computation, third edition*. The John Hopkins University Press, Baltimore and London, 1996.

[17] Brian Guenter, Cindy Grimm, Daniel Wood, Henrique Malvar, and Frédéric Pighin. Making Faces. In *SIGGRAPH 98 Conference Proceedings*. ACM SIGGRAPH, July 1998.

[18] Pat Hanrahan and Wolfgang Krueger. Reflection from Layered Surfaces Due to Subsurface Scattering. In *SIGGRAPH 93 Conference Proceedings*, volume 27, pages 165–174. ACM SIGGRAPH, August 1993.

[19] Bright Star Technologies Inc. *Beginning Reading Software*. Sierra On-Line, Inc., 1993.

[20] Horace H. S. Ip and Lijun Yin. Constructing a 3D Individualized Head Model from Two Orthogonal Views. *The Visual Computer*, 12:254–266, 1996.

[21] Gregory Ward J., Francis M. Rubinstein, and Robert D. Clear. A Ray Tracing Solution for Diffuse Interreflection. In *SIGGRAPH 88 Conference Proceedings*, volume 22, pages 85–92. August 1988.

[22] James R. Kent, Wayne E. Carlson, and Richard E. Parent. Shape Transformation for Polyhedral Objects. In *SIGGRAPH 92 Proceedings Conference*, volume 26, pages 47–54. ACM SIGGRAPH, July 1992.

[23] Rolf M. Koch, Markus H. Gross, Friedrich R. Carls, Daniel F. von Büren, George Fankhauser, and Yoav I. H. Parish. Simulating Facial Surgery Using Finite Element Methods. In *SIGGRAPH 96 Conference Proceedings*, pages 421–428. ACM SIGGRAPH, August 1996.

[24] Tsuneya Kurihara and Kiyoshi Arai. A Transformation Method for Modeling and Animation of the Human Face from Photographs. In Nadia Magnenat Thalmann and Daniel Thalmann, editors, *Computer Animation 91*, pages 45–58. Springer-Verlag, Tokyo, 1991.

[25] A. Lanitis, C. J. Taylor, and T. F. Cootes. A Unified Approach for Coding and Interpreting Face Images. In *Fifth International Conference on Computer Vision (ICCV 95)*, pages 368–373. Cambridge, Massachusetts, June 1995.

[26] C. L. Lawson and R. J. Hansen. *Solving Least Squares Problems*. Prentice-Hall, Englewood Cliffs, 1974.

[27] Seung-Yong Lee, Kyung-Yong Chwa, Sung Yong Shin, and George Wolberg. Image Metamorphosis Using Snakes and Free-Form Deformations. In *SIGGRAPH 95 Conference Proceedings*, pages 439–448. ACM SIGGRAPH, August 1995.

[28] Seung-Yong Lee, George Wolberg, Kyung-Yong Chwa, and Sung Yong Shin. Image Metamorphosis with Scattered Feature Constraints. *IEEE Transactions on Visualization and Computer Graphics*, 2(4), December 1996.

[29] Yuencheng Lee, Demetri Terzopoulos, and Keith Waters. Realistic Modeling for Facial Animation. In *SIGGRAPH 95 Conference Proceedings*, pages 55–62. ACM SIGGRAPH, August 1995.

[30] Yuencheng C. Lee, Demetri Terzopoulos, and Keith Waters. Constructing Physics-Based Facial Models of Individuals. In *Proceedings of Graphics Interface 93*, pages 1–8. May 1993.

[31] Francis H. Moffitt and Edward M. Mikhail. *Photogrammetry*. Harper & Row, New York, 3 edition, 1980.

[32] Shree K. Nayar, Katsushi Ikeuchi, and Takeo Kanade. Shape from Interreflections. *International Journal of Computer Vision*, 6:173–195, 1991.

[33] Gregory M. Nielson. Scattered Data Modeling. *IEEE Computer Graphics and Applications*, 13(1):60–70, January 1993.

[34] Frederic I. Parke. Computer Generated Animation of Faces. *Proceedings ACM annual conference.*, August 1972.

[35] Frederic I. Parke. A Parametric Model for Human Faces. PhD thesis, University of Utah, Salt Lake City, Utah, December 1974. UTEC-CSc-75-047.

[36] Frederic I. Parke and Keith Waters. *Computer Facial Animation*. A K Peters, Wellesley, Massachusetts, 1996.

[37] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, second edition, 1992.

[38] Kari Pulli, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, and Werner Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. In *Proc. 8th Eurographics Workshop on Rendering*. June 1997.

[39] Steven M. Seitz and Charles R. Dyer. View Morphing. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 21–30. ACM SIGGRAPH, August 1996.

[40] Chester C. Slama, editor. *Manual of Photogrammetry*. American Society of Photogrammetry, Falls Church, Virginia, fourth edition, 1980.

[41] Richard Szeliski and Sing Bing Kang. Recovering 3D Shape and Motion from Image Streams using Nonlinear Least Squares. *Journal of Visual Communication and Image Representation*, 5(1):10–28, March 1994.

[42] Richard Szeliski and Heung-Yeung Shum. Creating Full View Panoramic Image Mosaics and Texture-Mapped Models. In *SIGGRAPH 97 Conference Proceedings*, pages 251–258. ACM SIGGRAPH, August 1997.

[43] Demetri Terzopoulos and Keith Waters. Physically-based Facial Modeling, Analysis, and Animation. *Journal of Visualization and Computer Animation*, 1(4):73–80, March 1990.

[44] Kristinn R. Thórisson. Gandalf: An Embodied Humanoid Capable of Real-Time Multimodal Dialogue with People. In *First ACM International Conference on Autonomous Agents*. 1997.

[45] Michael W. Vannier, Jeffrey F. Marsh, and James O. Warren. Three-dimentional Computer Graphics for Craniofacial Surgical Planning and Evaluation. In *SIGGRAPH 83 Conference Proceedings*, volume 17, pages 263–273. ACM SIGGRAPH, August 1983.

[46] Keith Waters. A Muscle Model for Animating Three-Dimensional Facial Expression. In *SIGGRAPH 87 Conference Proceedings)*, volume 21, pages 17–24. ACM SIGGRAPH, July 1987.

[47] Lance Williams. Performance-Driven Facial Animation. In *SIGGRAPH 90 Conference Proceedings*, volume 24, pages 235–242. August 1990.

[48] Z. Zhang, K. Isono, and S. Akamatsu. Euclidean Structure from Uncalibrated Images Using Fuzzy Domain Knowledge: Application to Facial Images Synthesis. In *Proc. International Conference on Computer Vision (ICCV'98)*. January 1998.

A Least squares for pose recovery

To solve for a subset of the parameters given in Equation (2), we use linear least squares. In general, given a set of linear equations of the form

$$\boldsymbol{a}_j \cdot \boldsymbol{x} - b_j = 0, \tag{4}$$

we solve for the vector $\boldsymbol{x}$ by minimizing

$$\sum_j (\boldsymbol{a}_j \cdot \boldsymbol{x} - b_j)^2. \tag{5}$$

Setting the partial derivative of this sum with respect to $\boldsymbol{x}$ to zero, we obtain

$$\sum_j (\boldsymbol{a}_j \boldsymbol{a}_j^T)\boldsymbol{x} - b_j \boldsymbol{a}_j = 0, \tag{6}$$

i.e., we solve the set of *normal equations* [16]

$$\left(\sum_j \boldsymbol{a}_j \boldsymbol{a}_j^T \right) \boldsymbol{x} = \sum_j b_j \boldsymbol{a}_j. \tag{7}$$

More numerically stable methods such as *QR* decomposition or Singular Value Decomposition [16] can also be used to solve the least squares problem, but we have not found them to be necessary for our application.

To update one of the parameters, we simply pull out the relevant linear coefficient $\boldsymbol{a}_j$ and scalar value b_j from Equation (2). For example, to solve for $\boldsymbol{p}_i$, we set

$$\begin{aligned} \boldsymbol{a}_{2k+0} &= w_i^k (x_i^k \eta^k \boldsymbol{r}_z^k - s^k \boldsymbol{r}_x^k), & b_{2k+0} &= w_i^k (s^k t_x^k - x_i^k) \\ \boldsymbol{a}_{2k+1} &= w_i^k (y_i^k \eta^k \boldsymbol{r}_z^k - s^k \boldsymbol{r}_y^k), & b_{2k+1} &= w_i^k (s^k t_y^k - y_i^k). \end{aligned}$$

For a scalar variable like s^k, we obtain scalar equations

$$\begin{aligned} \boldsymbol{a}_{2k+0} &= w_i^k (\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k), & b_{2k+0} &= w_i^k \left(x_i^k + x_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) \right) \\ \boldsymbol{a}_{2k+1} &= w_i^k (\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k), & b_{2k+1} &= w_i^k \left(y_i^k + y_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) \right). \end{aligned}$$

Similar equations for $\boldsymbol{a}_j$ and b_j can be derived for the other parameters t_x^k, t_y^k, and η^k. Note that the parameters for a given camera k or 3D point i can be recovered independently of the other parameters.

Solving for rotation is a little trickier than for the other parameters, since $\boldsymbol{R}$ must be a valid rotation matrix. Instead of updating the elements in $\boldsymbol{R}_k$ directly, we replace the rotation matrix $\boldsymbol{R}^k$ with $\tilde{\boldsymbol{R}}\boldsymbol{R}^k$ [42], where $\tilde{\boldsymbol{R}}$ is given by Rodriguez's formula [15]:

$$\tilde{\boldsymbol{R}}(\hat{\boldsymbol{n}}, \theta) = \boldsymbol{I} + \sin\theta \boldsymbol{X}(\hat{\boldsymbol{n}}) + (1 - \cos\theta)\boldsymbol{X}^2(\hat{\boldsymbol{n}}), \tag{8}$$

where θ is an incremental rotation angle, $\hat{\boldsymbol{n}}$ is a rotation axis, and $\boldsymbol{X}(\boldsymbol{v})$ is the cross product operator

$$\boldsymbol{X}(\boldsymbol{v}) = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}. \tag{9}$$

A first order expansion of $\tilde{\boldsymbol{R}}$ in terms of the entries in $\boldsymbol{v} = \theta\hat{\boldsymbol{n}} = (v_x, v_y, v_z)$ is given by $\boldsymbol{I} + \boldsymbol{X}(\boldsymbol{v})$.

Substituting into Equation (2) and letting $\boldsymbol{q}_i = \boldsymbol{R}^k \boldsymbol{p}_i$, we obtain

$$\begin{aligned} w_i^k \left(x_i^k + x_i^k \eta^k (\tilde{\boldsymbol{r}}_z^k \cdot \boldsymbol{q}_i) - s^k (\tilde{\boldsymbol{r}}_x^k \cdot \boldsymbol{q}_i + t_x^k) \right) &= 0 \\ w_i^k \left(y_i^k + y_i^k \eta^k (\tilde{\boldsymbol{r}}_z^k \cdot \boldsymbol{q}_i) - s^k (\tilde{\boldsymbol{r}}_y^k \cdot \boldsymbol{q}_i + t_y^k) \right) &= 0, \end{aligned} \tag{10}$$

where $\tilde{\boldsymbol{r}}_x^k = (1, -v_z, v_y)$, $\tilde{\boldsymbol{r}}_y^k = (v_z, 1, -v_x)$, $\tilde{\boldsymbol{r}}_z^k = (-v_y, v_x, 1)$, are the rows of $[\boldsymbol{I} + \boldsymbol{X}(\boldsymbol{v})]$. This expression is linear in (v_x, v_y, v_z), and hence leads to a 3×3 set of normal equations in (v_x, v_y, v_z). Once the elements of $\boldsymbol{v}$ have been estimated, we can compute θ and $\hat{\boldsymbol{n}}$, and update the rotation matrix using

$$\boldsymbol{R}^k \leftarrow \tilde{\boldsymbol{R}}(\hat{\boldsymbol{n}}^k, \theta^k)\boldsymbol{R}^k.$$