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 deformation 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.


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 sub-cutaneous 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 compose 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, Bregel 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 com-
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 and numerically stable techniques [16, 37].

To formulate the pose recovery problem, we associate a rotation matrix $R_k$ and a translation vector $t_k$ with each camera pose $k$. (The three rows of $R_k$ are $r_{1k}$, $r_{2k}$, and $r_{3k}$, and the three entries in $t_k$ are $t_{1k}$, $t_{2k}$, $t_{3k}$.) We write each 3D feature point as $p_i$, and its 2D screen coordinates in the $k$-th image as $(x_i^k, y_i^k)$.

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 be morphed independently, and we introduce a novel approach for generating morphable textures that can be used to create realistic facial expressions.
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

\[
x^k_i = f^k \frac{r^k_i \cdot p_i + t^k_i}{r^k_i \cdot p_i + t^k_i} \\
y^k_i = f^k \frac{r^k_i \cdot p_i + t^k_i}{r^k_i \cdot p_i + t^k_i}
\]  

(This is just an explicit rewriting of the traditional projection equation \( x^k_i \propto R^k p_i + t^k \) where \( x^k_i = (x^k_i, y^k_i, f^k) \).)

Instead of using (1) directly, we reformulate the problem to estimate inverse distances to the object [41]. Let \( s^k = 1/r^k \) be this inverse distance and \( s^k = f^k y^k \) 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 \( t^k \) parameters.

Performing these substitution, we obtain

\[
x^k_i = s^k \frac{r^k_i \cdot p_i + t^k_i}{1 + \eta^k r^k_i \cdot p_i} \\
y^k_i = s^k \frac{r^k_i \cdot p_i + t^k_i}{1 + \eta^k r^k_i \cdot p_i}
\]

If we let \( w^k_i = (1 + \eta^k (r^k_i \cdot p_i))^{-1} \) be the inverse denominator, and collect terms on the left-hand side, we get

\[
w^k_i (x^k_i + \eta^k r^k_i \cdot (p^k_i - p_i)) - s^k (r^k_i \cdot p_i + t^k_i) = 0 \\
w^k_i (y^k_i + \eta^k r^k_i \cdot (p^k_i - p_i)) - s^k (r^k_i \cdot p_i + t^k_i) = 0
\]

Note that these equations are linear in each of the unknowns that we wish to recover, i.e., \( p_i, t^k_i, r^k_i, \eta^k, s^k, \) and \( R^k \), if we ignore the variation of \( w^k_i \) with respect to these parameters. (The reason we keep the \( w^k_i \) 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 \( (x^k_i, y^k_i) \). 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 \( p_i, R^k, t^k_i \) and \( r^k_i \), and finally \( \eta^k \). 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 \( p_i \), 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 \( u_i = p_i - p_i^{(0)} \) away from the original positions \( p_i^{(0)} \) at every constrained vertex \( i \), construct a function that gives the displacement \( u_i \) 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 \( f(p) \) fitted to the known data \( u_i = f(p_i) \), from which we can compute \( u_j = f(p_j) \).

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
\[
f(p) = \sum_i c_i \phi([p - p_i]),
\]
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:
\[
f(p) = \sum_i c_i \phi([p - p_i]) + M p + t,
\] (3)

To determine the coefficients \(c_i\) and the affine components \(M\) and \(t\), we solve a set of linear equations that includes the interpolation constraints \(u_i = f(p_i)\), as well as the constraints \(\sum_i c_i = 0\) and \(\sum_i c_i p_i^T = 0\), 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 \(\phi(r) = e^{-r^2/64}\), 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 \(p_i\) 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 parameterization. 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 \(p\) on the face model its texture color \(T(p)\).

Each point \(p\) 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 (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 \(T(p)\) at each point on the face model can be expressed as a convex combination of the corresponding colors in the photographs:
\[
T(p) = \sum_i m^i(p) T_i(x, y),
\]

Here, \(T_i\) is the image function (color at each pixel of the \(k\)-th photograph) and \((x, y)\) are the image coordinates of the projection of \(p\) onto the \(k\)-th image plane. The weight map \(m^k(p)\) 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: \(m^k(p)\) should be zero unless \(p\) 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: \(m^k(p)\) should depend on the “positional certainty” [24] of \(p\) with respect to the \(k\)-th image. The positional certainty is defined as the dot product between the surface normal at \(p\) and the \(k\)-th direction of projection.
4. View similarity: for view-dependent texture mapping, the weight \(m^k(p)\) should also depend on the angle between the direction of projection of \(p\) 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 \( p \) 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 \( p \) 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 \( p. \) (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 \( P^k(p) \) be
the positional certainty of \( p \) with respect to the \( k \)-th image.

3. Set weight \( m^k(u, v) \) to the product \( F^k(u, v) \cdot P^k(p) \).

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 to-
gether 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 sev-
everal times, each time using a different input photograph as a texture
map, and blend the results. More specifically, for each input photo-
graph, 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 \( d \), we first select the subset of pho-
tographs 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 ob-
ject. Since our cameras were positioned roughly in the same plane,
we select just the two photographs whose view directions \( d' \) and
\( d'' \) are the closest to \( d \) and blend between the two.

In choosing the view-dependent term \( V^k(d) \) 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:

\[
V^k(d) = \begin{cases} 
    d \cdot d' & \text{if } k = \ell + 1 \\
    0 & \text{otherwise}
\end{cases}
\]

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

View-dependent texture maps have several advantages over cylin-
drical 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 tex-
ture map for these parts in every facial expression. The ears have an
intricate geometry with many folds and usually fail to project with-
out overlap on a cylinder. The hair has fine-detailed texture that is
difficult to register properly across facial expressions. For these rea-
sons, 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 eye-
lids and the mouth respectively. We approximate this shadowing by
modulating the brightness of the eye and teeth texture maps accord-
ing to the size of the eyelid and mouth openings.

### 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 correspon-
dences 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 correspondence between vertices 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.

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 photography 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 (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.

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.
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

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

$$a_j \cdot x - b_j = 0,$$

we solve for the vector $x$ by minimizing

$$\sum_j (a_j \cdot x - b_j)^2.$$  

(5)

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

$$\sum_j (a_j a_j^T) x = \sum_j b_j a_j,$$

(6)

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

$$\sum_j a_j a_j^T x = \sum_j b_j a_j.$$  

(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 $a_j$ and scalar value $b_j$ from Equation (2). For example, to solve for $p_1$, we set

$$a_{23+0} = \frac{w_1^2}{s_k} (s_k^2 w_k^3 - x_k^2), \quad b_{23+0} = w_1^2 (s_k^2 w_k^3 - x_k^2),$$

$$a_{23+1} = w_1^2 (s_k^2 w_k^3 - x_k^2), \quad b_{23+1} = w_1^2 (s_k^2 w_k^3 - y_k^2).$$

For a scalar variable like $s_k$, we obtain scalar equations

$$a_{23+0} = \frac{w_1^2}{s_k} (r_k^2 \cdot p_1 + t_k^2), \quad b_{23+0} = w_1^2 \left( \frac{r_k^2}{s_k} \cdot p_1 + t_k^2 \right),$$

$$a_{23+1} = w_1^2 (r_k^2 \cdot p_1 + t_k^2), \quad b_{23+1} = w_1^2 \left( \frac{r_k^2}{s_k} \cdot p_1 + t_k^2 \right).$$

Similar equations for $a_j$ and $b_j$ can be derived for the other parameters $t_k^3$, $t_k^3$, and $s_k$. 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 $R$ must be a valid rotation matrix. Instead of updating the elements in $R_k$ directly, we replace the rotation matrix $R^k$ with $RR^*$ [42], where $R$ is given by Rodriguez’s formula [15]:

$$R(\hat{\theta}, \hat{\phi}) = I + \sin \theta X(\hat{\phi}) + (1 - \cos \theta) X^2(\hat{\phi}),$$

(8)

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

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

(9)

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

Substituting into Equation (2) and letting $q_i = R^* p_i$, we obtain

$$w_1^2 \left( s_k^2 + s_k^2 \eta (s_k^2 - q_k) - s_k^2 (s_k^2 + q_k + t_k^3) \right) = 0$$

(10)

$$w_1^2 \left( s_k^2 + s_k^2 \eta (s_k^2 - q_k) - s_k^2 (s_k^2 + q_k + t_k^3) \right) = 0,$$

where $p_i^3 = (1, -v_x, v_y), p_i^4 = (v_x, 1, -v_y), p_i^5 = (-v_x, v_y, 1)$, are the rows of $[I + X(v)]$. This expression is linear in $(v_x, v_y, v_z)$, and hence leads to a $3 \times 3$ set of normal equations in $(v_x, v_y, v_z)$. Once the elements of $v$ have been estimated, we can compute $\theta$ and $\hat{\phi}$, and update the rotation matrix using

$$R^k = R(\hat{\theta}, \hat{\phi}) R.$$