Realistic Facial Animation Using Image-Based 3D Morphing

Frederic Pighin   Joel Auslander   Dani Lischinski
David H. Salesin   Richard Szeliski[†]

May 9, 1997

University of Washington   [†]Microsoft Research

Technical report UW-CSE-97-01-03

Abstract

We present new techniques for creating a realistic textured 3D facial model from several photographs of a human subject and for performing facial animation by morphing between models corresponding to different facial expressions. Starting from several uncalibrated views of an individual, 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 individuals face. A scattered data interpolation technique is then used to deform a generic face mesh into a 3D model of the individual's face. Having recovered the camera poses and the facial geometry, we extract from the input images a texture map for the model. An optical flow technique is used for improving the registration of the input images in texture space. This process is repeated for several facial expressions of a particular individual. To animate between these facial expressions we use 3D shape morphing between the corresponding facial models, while at the same time blending the corresponding textures. Using our technique we have been able to generate highly realistic facial models and natural looking transitions between different expressions.

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, from *The Artist's Complete Guide to Facial Expression* [10]

1 Introduction

Realistic facial animation 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 [25], 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 [15], video teleconferencing [6], user-interface agents and avatars [32], and medical facial surgery planning [33, 18]. Yet it is fair to say that 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* were modeled using several thousand control points each [8]. 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. And 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). And all of these problem 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 human will immediately detect.

A number of approaches have been developed to model and animate realistic facial expressions in three dimensions, including simple geometric interpolation between digitized [25] or laser-scanned models; performance-based animation, in which measurements from real human actors are used to drive synthetic characters [4, 36]; and various forms of physically-based animation, in which musculo-skeletal controls modeled with various degrees of realism are used to create facial animations [34, 31]. However, for sheer realism, one of the most effective approaches to date has been the use of 2D morphing between photographic images [2]. Indeed, some quite 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 move about. The production of this video, however, required animators to painstakingly specify a very large set of 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"[29], 2D morphing still lacks some of the advantages of having a 3D model, such as the ability to composite the image with other 3D graphics.

In this paper, we show how 2D morphing techniques can be combined with 3D transformations of a geometric model to automatically produce true 3D facial animation with a higher degree of realism than any previous method of which we are aware. The basic steps of our process are as follows. 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 corresponding points on the face in the different views (corners of the 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. These 3D positions are then used to deform a generic 3D face mesh into a closer correspondence with the face of the particular human subject. Finally, we texture-map the photos onto the 3D model. This whole process is repeated for the same human subject, with several different facial expressions. To produce facial animations, we simply blend between two 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 facial models have been constructed.

Our approach utilizes a photogrammetric technique, in which images are used to create precise geometry. The earliest such techniques applied to facial modeling and animation employed grids that were drawn directly on the human subject's face [25, 26]. One consequence of using these grids, however, is that the images used to construct geometry can no longer be used as a valid texture maps for the subject. More recently, several methods have been proposed for modeling the face photogrammetrically without the use of grids [19, 1, 16]. These modeling methods are quite similar in their basic concept to the modeling technique described in this paper. However, we improve upon these previous methods in a number of ways, including allowing fairly arbitrary camera positions and lenses (rather than using a fixed pair that are precisely oriented); employing an optical pixel flow technique for fine, subpixel registration of texture maps (which could also be used to refine the geometric model further). Furthermore, we employ these techniques not only for creating realistic facial models, but also for performing realistic facial animations in 3D — animations that we believe are more realistic than any previous work not involving a huge degree of human interaction.

The rest of the paper is organized as follows. Section 2 describes our method for fitting a generic facial 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 facial model. Section 5 presents the results of our experiments with the proposed techniques. Section 6 concludes this paper and offers directions for further research.

2 Model fitting

The task of the model fitting stage is to adapt a generic face model to fit an individual's face and facial expression. For our work, we use a generic face model made available by Keith Waters [27] (Figure 2b). As inputs to this stage, we take several images of the face from different viewpoints (the exact camera arrangement is described in Section 5). The output of this stage is a face model which has been adapted to a given individual, as well as a precise estimate of the camera locations.

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 position in each image (where visible) is specified by hand. An example with nine selected feature points is shown in Figure 2a.

The second stage applies a scattered data interpolation algorithm to the estimated 3D coordinates of the feature points to compute the positions of the remaining facial mesh vertices. In the third 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, side view, etc.) and of 3D shape (given by the generic head model), we iteratively improve the pose and 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 presented in [30]. Unlike this previous algorithm, 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 which can be solved using very simple and numerically stable techniques [14, 28].

To formulate the pose recovery problem, we associate a rotation matrix $\boldsymbol{R}^k$ and a translation vector $\boldsymbol{t}^k$ with each of the m camera poses (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 the coordinates of the n 3D feature points as $\boldsymbol{p}_i$, and the 2D screen coordinates of point i in camera k as (x_i^k, y_i^k). In theory, we could have mn such measurements, but in practice we will have fewer since some feature points will not be visible in all views.

Assuming that the origin of the (x, y) image coordinate system lies at the optical center of each image, i.e., where to 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

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

$$y_i^k = f^k \frac{\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k}{\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i + t_z^k} \tag{2}$$

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

Instead of using the above equation, we reformulate the problem to estimate inverse distances to the object instead [30]. Let $\eta^k = 1/t_z^k$ be this inverse distance and $s^k = f^k/t_z^k$ 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 this substitution and bringing the denominators in (1–2) over to the left hand side, we get the pair of equations

$$\begin{aligned} x_i^k + x_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) - s^k (\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k) &= 0 & (3) \\ y_i^k + y_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) - s^k (\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k) &= 0. & (4) \end{aligned}$$

Notice that these equations are linear in all of the unknowns which we wish to recover, i.e., $\boldsymbol{p}_i$, t_x^k, t_y^k, η^k, s^k, and $\boldsymbol{R}$.

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: s^k; $\boldsymbol{p}_i$; $\boldsymbol{R}^k$; t_x^k and t_y^k; and η^k. This order was chosen to reflect which parameters can most reliably be recovered given our crude initial pose and shape estimates. For each set of parameters chosen, we solve for the unknowns by either computing a weighted mean (for the scalar problems of estimating s^k, t_x^k, t_y^k, and η^k), or using linear least squares (solving 3×3 systems) for estimating the $\boldsymbol{p}_i$.[1] 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.

Solving for rotation is a little trickier than for the other parameters, since $\boldsymbol{R}$ must be a valid rotation matrix. In our work, we use incremental rotations, i.e., we replace the rotation matrix $\boldsymbol{R}^k$ with $[\boldsymbol{I} + \boldsymbol{X}(\boldsymbol{v})]\boldsymbol{R}^k$, where where $\boldsymbol{X}(\boldsymbol{v})$ is the cross product operator with the incremental rotation vector $\boldsymbol{v} = (v_x, v_y, v_z)$,

$$\boldsymbol{X}(\boldsymbol{v}) = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_z & 0 \end{bmatrix}. \tag{5}$$

This leads to a 3×3 linear system in (v_x, v_y, v_z). The angular rotation can be converted to a true incremental rotation matrix using Rodriguez's formula [11]:

$$\Delta \boldsymbol{R}(\hat{\boldsymbol{n}}, \theta) = \boldsymbol{I} + \sin\theta \boldsymbol{X}(\hat{\boldsymbol{n}}) + (1 - \cos\theta) \boldsymbol{X}^2(\hat{\boldsymbol{n}}), \tag{6}$$

where $\theta = \|\boldsymbol{v}\|$ and $\hat{\boldsymbol{n}} = \boldsymbol{v}/\theta$, and the new rotation matrix computed using

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

[1] To weight the equations properly in a least-squared error sense, we have to divide each equation through by the current value of the denominator $(1 + \eta^k \boldsymbol{r}_z^k \cdot \boldsymbol{p}_i)$.

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 to create a *displacement map* which describes, the 3D displacement 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 [24]. Given a set of known displacements $\boldsymbol{u}_i = \boldsymbol{p}_i - \boldsymbol{p}_i^{(0)}$ at vertices i, we construct a function that gives us the displacement $\boldsymbol{u}_j$ for every unconstrainted vertex j. There are several general approaches to solving this problem.

The first choice we must make is the selection of an embedding space, i.e., the domain of the function we are computing (the range is vectors in 3D). The simplest choice is to use the original 3D coordinates of the points as the domain. Another choice is to use some 2D parametrization of the surface mesh, e.g., the cylindrical coordinates we use in Section 3. In our work, we interpolate over a 3D domain, i.e., we 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 $\boldsymbol{u}_j = \boldsymbol{f}(\boldsymbol{p}_j)$.

Again, several choices exist for how to construct the interpolating function [24]. We use a method based on *radial basis functions*, i.e., functions of the form,

$$\boldsymbol{f}(\boldsymbol{p}) = \sum_i \boldsymbol{c}_i \phi(\|\boldsymbol{p} - \boldsymbol{p}_i\|), \tag{7}$$

where the $\phi(r)$ are the 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).

To determine the coefficients $\boldsymbol{c}_i$, we solve the set of linear equations $\boldsymbol{u}_i = \boldsymbol{f}(\boldsymbol{p}_i)$. Observe that the x, y, and z components of the $\boldsymbol{c}_i$ vectors can be computed independently using the set of equations

$$u_i = \sum_j c_i \phi(r_{ij}), \quad \text{where} \quad r_{ij} = \|\boldsymbol{p}_i - \boldsymbol{p}_j\|,$$

i.e., by inverting the matrix $\boldsymbol{\Phi}$ with $\phi_{ij} = \phi(r_{ij})$ and multiplying $\boldsymbol{\Phi}^{-1}$ by (the x, y, or z coordinates of) $\boldsymbol{u}_i$. This matrix computation and inversion can be performed once (it depends only on the original configuration of the vertices), unless more constraints are selected during a subsequent correspondence-based shape refinement stage.

Many different functions for $\phi(r)$ have been proposed [24]. After experimenting with a number of functions, we have chosen to use $\phi(r) = e^{-r/64}$, where the diameter of the face model measured from side to side is on the order of 6 units. For now, this choice has given us reasonable results, but we would like to investigate even better interpolants in the future. Figure 2c shows the shape of the face model after we have interpolated the set of computed 3D displacements at the feature points and applied them to the whole 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 2d shows the shape of the face model after thirteen additional correspondences have been specified. Since the pose of each camera is known, we can easily compute and display the *epipolar lines* corresponding to a point selected in one image [11]. This could be used to facilitate the process of manually establishing correspondences. We could also allow the user to draw corresponding curves (e.g., the profile, which corresponds to the midline in the frontal view), and then automatically set up pointwise correspondences using the epipolar geometry.

3 Texture extraction

In this section we describe the process of extracting from the input photographs the texture maps necessary for rendering realistic images of a reconstructed facial model from various viewpoints. We first explain how to compute a single view-independent texture map, which can be used for quickly rendering the face from any viewpoint. Then we describe a view-dependent texture mapping technique that is slower, but generates images of higher fidelity.

The texture extraction stage proceeds as follows:

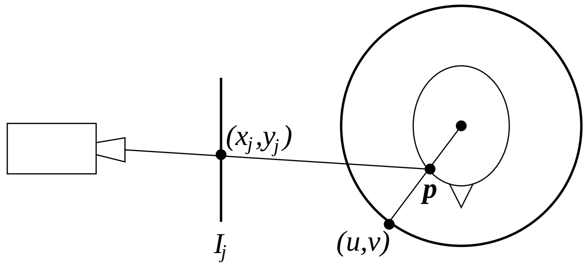

Figure 1: Geometry for texture extraction

1. A mapping between 3D coordinates on the facial mesh and a 2D texture space is defined using a cylindrical projection, similarly to several previous authors (e.g., [5, 21, 19]).

2. For each texture pixel centered at the cylindrical coordinates (u, v):

   (a) Compute the 3D point $\boldsymbol{p}$ on the surface of the face whose cylindrical projection is (u, v) (see Figure 1).

   (b) For each camera pose j compute (x_j, y_j), the projection of $\boldsymbol{p}$ onto the corresponding image plane, and extract the color there $I_j(x_j, y_j)$.

   (c) Set the color of the texture at (u, v) to a linear combination of the corresponding image colors:

   $$T(u, v) = \sum_j w_j(u, v)\, I_j(x_j, y_j)$$

   where $w_j(u, v)$ is the relative weight of the contribution of the j-th input image to the (u, v) pixel of the texture map, with the property that $\sum_j w_j(u, v) = 1$. The computation of the weight maps w_j is described in section 3.1.

Computing the 3D point $\boldsymbol{p}$ corresponding to point (u, v) in texture space can be accomplished by intersecting a ray with the surface of the face. In order to avoid multiple intersection tests with triangles in the facial mesh, we construct a map in texture space that specifies for each point (u, v) the triangle whose cylindrical projection covers that point. This map is constructed in a pre-processing stage by rendering the cylindrical projection of each face triangle using a unique color. (This technique is essentially a cylindrical variant of a well-known ray tracing acceleration technique, known as the *item* *buffer* [35].) Once the containing triangle is established, $\boldsymbol{p}$ is obtained by performing a single ray–plane intersection.

3.1 Weight-map construction

Constructing the weight maps for blending the input images into a single texture map is probably the trickiest and the most interesting component of our texture-mapping technique. There are several desirable properties that a weight map should have:

- If the facial surface point $\boldsymbol{p}$ corresponding to the texture map pixel (u, v) is not visible in the j-th image, the weight $w_j(u, v)$ should be zero.

- The weights in each weight map should vary smoothly, in order to ensure a seamless blend between the input images.

- The weight $w_j(u, v)$ should depend on the "positional certainty" [19] of the corresponding point $\boldsymbol{p}$ in the j-th image. The positional certainty is defined as the dot product between the surface normal at $\boldsymbol{p}$ and the direction of projection.

- Finally, to produce a view-dependent texture map, the weight $w_j(u, v)$ should also depend on the similarity between the direction of projection of $\boldsymbol{p}$ onto the j-th image and the direction of projection in the new view.

Previous authors used weighting function that satisfy only a subset of these requirements. For example, Kurihara and Arai [19] use positional uncertainty as their weighting function, but they do not account for self-occlusion. Akimoto *et al.* [1] as well as Ip and Yin [16] blend the images smoothly, but address neither self-occlusion nor positional certainty. Debevec *et al.* [7], 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.

Our method for computing the weight functions satisfies all of the properties mentioned above. We first set each entry of w_j to 1 if the corresponding point $\boldsymbol{p}$ is visible in image j. We then smoothly ramp the values from 1 to 0 in the vicinity of boundaries in the resulting binary map. This smoothing step helps eliminate seams in the final texture map. Finally, each pixel

$w_j(u, v)$ is multiplied by the positional certainty term, and the weights are normalized so that they sum to 1.

To facilitate fast visibility testing of points on the surface of the face from a particular camera pose we first render all the triangles in the mesh from a view corresponding to the camera pose and save the resulting depth map from the Z-buffer. Then we render each triangle separately and compare the depth values at the pixels covered by the triangle to the corresponding saved depth values. This process allows us to classify each triangle as totally visible, partially visible, or totally occluded. Whenever a pixel (u, v) in the weight map w_j is covered by a partially visible triangle, we project the corresponding point $\boldsymbol{p}$ using the j-th camera settings and compare the resulting depth value to the corresponding value in the j-th depth map.

3.2 View-dependent texture mapping

Rather than blending all of the input images into a single view-independent cylindrical texture map once and for all, we can defer the blending until the viewpoint from which the facial model is to be rendered has been specified. Each of the input images is warped into the cylindrical texture space and stored along with its weight map. Once the viewpoint has been specified, we increase the relative weights of those texture maps whose corresponding camera poses are closer to the new viewpoint. Blending the individual texture maps together in this fashion results in a view-dependent texture map [7]. View-dependent texture mapping is more expensive because the texture has to be reassembled once per frame, however in our experience it made little difference with respect to the faster view-independent texture mapping.

3.3 Improving texture registration

While using smooth weight maps will remove most of the visible discontinuities in the composite texture map, some residual ghosting or blurring artifacts may still be visible due to small mis-registrations between the images (e.g., the geometry may be wrong or not detailed enough). To improve the quality of the composite textures, we can locally warp each component texture (and weight) map before blending.

To compute good values for the warping (i.e., the *displacement map*), we use a classical optical flow technique [22, 3], as described in Appendix A. This algorithm registers two images I and I' by finding two fields $u(x, y)$

and $v(x, y)$ which locally minimize the squared errors in

$$I'(x + u(x, y), y + v(x, y)) - I(x, y). \tag{8}$$

A summary of our algorithm looks like this:

1. Input the two images I and I' and any initial displacement estimates (u, v) (if none are given, set $(u, v) \leftarrow (0, 0)$).

2. Resample $I'(x, y)$ to yield $I'(x', y')$, where $x' = x + u$ and $y' = y + v$, and write this image out, if desired.

3. Compute the gradients at each pixel, $f_{x,y}$ and $g_{x,y}$, and the intensity (or color) error $h_{x,y}$, and form the quantities $a_{x,y} \ldots e_{x,y}$.

4. Sum these quantities around each pixel to get aggregated values $a'_{x,y} \ldots e'_{x,y}$.

5. At each pixel, solve the linear system (12) given in Appendix A using the aggregated values $a'_{x,y} \ldots e'_{x,y}$. If the system is not of full rank or is ill-conditioned, set $(\delta u, \delta v) \leftarrow 0$.

6. Update $(u_{x,y}, v_{x,y}) \leftarrow (u_{x,y} + \delta u_{x,y}, v_{x,y} + \delta v_{x,y})$ and write out this displacement field.

7. Continue iterating at step 2 if the root mean squared correction to the displacement field is above a threshold and the maximum number of iterations has not been reached.

We have enhanced this basic algorithm in a number of ways. The weights computed during the texture map blending stage are used to weight the contributions of each pixel during the displacement map computation. This leads to smoother changes in the flow estimates near the borders of the visibility map. Since the initial misregistration may be large, we use a coarse-to-fine algorithm based on an image pyramid to improve the convergence [3].

Figures 4a–b show two texture maps, in cylindrical coordinates, that have been extracted from the original photos through projection onto the geometric model. Because the geometry is imperfect, the extraction process is inexact, which leads to local misregistrations between the two texture maps. When the texture maps are blended, the result is loss of detail and ghosting artifacts, as seen in figure 4g. We can reduce this using optical flow to warp one of the texture maps (in this case 4a) to better match the other, before we do the blend. We compute a smooth warp that minimizes the difference between the warped texture map 4a and the target texture map 4b. This warp is shown in figure 4d, with the difference before and after the warp shown in 4e and 4f. By applying this warp before blending, the resulting combined map is improved from that shown in 4g to that shown in 4h. Notice, for example, how the ghosting in the eye region has been eliminated.

4 Face morphing

In order to generate a continuous transition between key facial expressions of the same person, we first utilize the techniques described in the previous sections to construct a 3D textured facial model for each of the key expressions. A continuous transition between these key expressions can now be created as follows:

1. Obtain an intermediate geometric model of the face by 3D morphing between the facial meshes corresponding to the key expressions.

2. Obtain a texture map for the intermediate expression by image morphing between the corresponding texture maps.

3. Render the resulting texture-mapped facial model from the desired viewpoint.

The problem of morphing between general polygonal meshes in 3D is a difficult one [17]. In our case, however, the topology of the two key face meshes is identical. Thus, the correspondences between vertices in the key meshes need not be computed. Furthermore, for most individuals, and most pairs of facial expressions, we can assume that the geometry of the meshes is fairly similar. Thus, a satisfactory morphing sequence can be obtained by simple linear interpolation between the geometric coordinates of corresponding vertices in each of the two face meshes.

In order to morph between a pair of textures we also need correspondences between the two texture images [2]. These correspondences are given to us by means of the texture coordinates of the facial mesh vertices in each of the two meshes. Thus, in order to morph between the two textures, we should compute the cylindrical coordinates for each vertex in the intermediate facial mesh, warp each of the two textures to the new coordinates, and blend them together.

Note that if the texture coordinates were the same for each corresponding pair of vertices in the facial meshes, there would be no need to warp the textures before blending them. However, using the same set of texture coordinates for each facial mesh is inconvenient, as it means that we would not be able to simply use cylindrical coordinates as texture coordinates. As a consequence, the computations to extract the textures would become more complicated . Also,

Warping high-resolution textures can be time-consuming. Fortunately, explicitly morphing between the textures is not always necessary. Typically, we are interested in producing only a single image of each intermediate expression from a particular viewpoint. In this case, instead of morphing between the two key texture maps, we simply render the intermediate facial mesh twice, once for each key expression, using the appropriate texture map. The resulting images are then blended together. The blending can be performed rapidly using the accumulation buffer in OpenGL [23].

5 Results

In order to put our technique to the test, we photographed a female and a male model 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. Five typical images (cropped to the size of the subject's head) are shown in Figure 2a. The photographs were digitized using the Kodak PhotoCD process.

We used the interactive modeling system described in sections 2 and 3 to create four facial models: (a) the female subject (Karla) with a neutral expression, (b) Karla with an expression of fear mixed with pain, (c) the male subject (Steve) with a neutral face, and (d) Steve smiling. Despite the lack of hair, the resulting textured facial models look very lifelike and bear a great deal of resemblance to the individuals in the photographs, as demonstrated by the first and the last column of Figure 5.

Following the modeling stage, we generated a 3D morphing sequence for each of the individuals using the technique described in section 4. The two morphing sequences are shown on the accompanying video tape. The second and the fourth rows in Figure 5 show a few frames from the transition between the expressions. In the video sequence, this transition appears very natural and realistic, possessing a considerably less synthetic look than typical 2D image morphing sequences.

6 Summary and future work

In this paper, we have presented a new approach to image-based realistic facial animation. Rather than relying on pure 2D image morphing, we construct a 3D model of the face and combine 3D shape morphing with 2D image morphing to obtain realistic transitions between facial expressions. Our method has the advantage over previous similar approaches that it extracts textured facial models from photographs taken with uncalibrated cameras from arbitrary locations. In addition, our method computes from these photographs both view-independent and view-dependent textures, using carefully designed weight functions. We have also described an optical pixel flow technique that we use for improving the registration between the images in texture space. This technique could also be used for adaptively improving the geometric accuracy of the facial model. With the aid of these techniques, we have been able to generate some of the most realistic 3D facial models to date. Furthermore, since these models have built-in correspondences between them, we are able to generate realitic transitions between facial expressions automatically, using a simple and inexpensive 3D morphing technique.

The results that we have obtained with our current implementation are indeed encouraging; however, we see this work as just the beginning of a challenging research project to build a complete image-based facial animation system. Building this first prototype has given us several insights into where the inherent difficulties lie.

Constructing accurate 3D models of the face from images has proven to be a non-trivial task. One reason is that the current model we use only covers the face and not the complete head, and establishing correspondences at the edge of the face (especially near the ears) can be quite difficult. A second reason is that our current scattered data interpolant is not sufficiently flexible. We believe that it could be significantly improved by better tuning the radial basis functions and adding an affine registration component. A third reason is that finding accurate correspondences in the relatively smooth regions such as the cheeks is quite difficult (this is where active rangefinders such as laser scanners have a big advantage).

To improve the 3D model construction, we are developing an algorithm to adjust the model geometry based on the local misregistration errors computed during the texture map alignment process. In principle, small changes in geometry correspond to motions in the (u, v) coordinate space, so it should be possible to incrementally improve our 3D vertex positions by minimizing misregistration errors. This idea is similar to model-based stereo matching

[12], and will use optimization techniques similar to those used in through-the-lens camera control [13].

In addition to refining the positions of the face vertices, misregistrations error could also be used to guide a localized subdivision step, which would enable us to refine the generic face model to better preserve fine facial features. Note, however, that such a refinement must be done identically for all 3D models between which we may wish to morph (say, all models of a given person), if we are to use the same simple morphing algorithm described in this paper.

Our ultimate goal, as far as the modeling stage 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 [20] give us cause for hope. If we are successful in our quest, our techniques could then be applied to a wider range of 3D image-based reconstruction problems.

Creating photorealistic models from images is just one of the challenges that stand before us. An even greater challenge is creating complex realistic animations. Here, we must develop a rich vocabulary for describing facial movements. Such a vocabulary will require the ability to blend between several expressions, and to do so in a localized fashion (so that different parts of the face can have different expressions or actions).

There are several potential ways to attack this problem. One would be to adopt and adapt an action unit-based system such as the Facial Action Coding System (FACS) [9]. Another possibility would be to apply modal analysis (principal component analysis) techniques to describe facial expression changes using a small number of motions [20]. Finding natural control parameters to facilitate animation, and developing realistic looking temporal profiles for such movements, is also challenging.

While constructing realistic facial animations is our first goal, ultimately we would like to also 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 and facile 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.

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] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 35–42, July 1992.

[3] J. R. Bergen, P. Anandan, K. J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *Second European Conference on Computer Vision (ECCV'92)*, pages 237–252, Santa Margherita Liguere, Italy, May 1992. Springer-Verlag.

[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] David T. Chen, Andrei State, and David Banks. Interactive shape metamorphosis. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 43–44. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7.

[6] C. S. Choi, H. Harashima, and T. Takebe. Highly accurate estimation of head motion and facial action information on knowledge-based image coding. *IEICEJ*, PRU90-68:1–8, October 1990.

[7] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 11–20. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

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

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

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

[11] O. Faugeras. *Three-dimensional computer vision: A geometric viewpoint.* MIT Press, Cambridge, Massachusetts, 1993.

[12] P. Fua and Y. G. Leclerc. Registration without correspondences. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 121–128, Seattle, Washington, June 1994. IEEE Computer Society.

[13] M. Gleicher and A. Witkin. Through-the-lens camera control. *Computer Graphics (SIGGRAPH'92)*, 26(2):331–340, July 1992.

[14] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, Maryland, second edition, 1989.

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

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

[17] James R. Kent, Wayne E. Carlson, and Richard E. Parent. Shape transformation for polyhedral objects. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 47–54, July 1992.

[18] R. M. Koch, M. H. Gross, F. R. Carls, D. F. von Büren, G. Fankhauser, and Y. Parish. Simulating facial surgery using finite element methods. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 421–428. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[19] 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.

[20] 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.

[21] Yuencheng Lee, Demetri Terzopoulos, and Keith Waters. Realistic modeling for facial animation. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 55–62. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[22] B. D. Lucas and T. Kanade. An iterative image registration technique with an application in stereo vision. In *Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, pages 674–679, Vancouver, 1981.

[23] Jackie Neider, Tom Davis, and Mason Woo. *OpengGL Programming Guide*. Addison Wesley, 1993.

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

[25] Frederic I. Parke. Computer generated animation of faces. *Proc. ACM annual conf.*, August 1972.

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

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

[28] 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.

[29] Steven M. Seitz and Charles R. Dyer. View morphing. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 21–30. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[30] R. Szeliski and S. B. 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.

[31] D. Terzopoulos and K. Waters. Physically-based facial modeling, analysis, and animation. *J. of Visualization and Computer Animation*, 1(4):73–80, March 1990.

[32] K. R. Thórisson. Gandalf: An embodied humanoid capable of real-time multimodal dialogue with people. In *First ACM International Conference on Autonomous Agents*, 1997. Mariott Hotel, Marina del Rey, California, February 5-8.

[33] M. W. Vannier, J. F. Marsh, and J. O. Warren. Three-dimentional computer graphics for craniofacial surgical planning and evaluation. *Computer Graphics*, 17(3):263–273, 1983.

[34] Keith Waters. A muscle model for animating three-dimensional facial expression. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 17–24, July 1987.

[35] H. Weghorst, G. Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, January 1984.

[36] Lance Williams. Performance-driven facial animation. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 235–242, August 1990.

A Optic flow algorithm

In this appendix, we summarize the derivation of the optic flow algorithm first presented in [22] (for a more recent description with some useful generalizations, see [3]).

To register two images I and I', we compute a *displacement map*, i.e., a 2-D displacement at each point, $u(x, y)$ and $v(x, y)$. We wish to compute values of u and v which satisfy

$$I'(x + u(x, y), y + v(x, y)) = I(x, y) \tag{9}$$

In principle, we have twice as many unknowns as measurement. However, we can turn this into an overconstrained problem by choosing a neighborhood around each pixel (say 5×5), and solving the set of linear equations to yield (u, v). To do this, we use a Taylor series expansion of (u, v) around its current estimate,

$$\begin{aligned} I'(x + u + \delta u, y + v + \delta v) - I(x, y) &\approx \\ I'(x', y') + I'_x(x', y')\delta u + I'_y(x', y')\delta v - I(x, y) &= 0 \end{aligned} \tag{10}$$

where $(x', y') = (x+u, y+v)$ are the warped pixel coordinates, and $I'_x(x', y')$ and $I'_y(x', y')$ are the horizontal and vertical intensity gradients. To resample the image I' at location (x', y'), we use bilinear pixel interpolation. To simplify computation, we can also replace the gradient estimates $I'_x(x', y')$ and $I'_y(x', y')$ by $I_x(x, y)$ and $I_y(x, y)$.

Recognizing that we are operating on a discrete pixel grid, let us re-write the above equations as

$$f_{x,y}\delta u + g_{x,y}\delta v = h_{x,y} \tag{11}$$

where $f_{x,y} = I'_x(x', y')$, $g_{x,y} = I'_y(x', y')$, and $h_{x,y} = I(x, y) - I'(x', y')$ is the current intensity error. It is straightforward to show that the normal equations arising from the above linear system have the form

$$\sum_{(x,y)} \begin{bmatrix} a_{x,y} & b_{x,y} \\ b_{x,y} & c_{x,y} \end{bmatrix} \begin{bmatrix} \delta u \\ \delta v \end{bmatrix} = \sum_{(x,y)} \begin{bmatrix} d_{x,y} \\ e_{x,y} \end{bmatrix}, \tag{12}$$

where $a_{x,y} = f^2_{x,y}$, $b_{x,y} = f_{x,y}g_{x,y}$, $c_{x,y} = g^2_{x,y}$, $d_{x,y} = f_{x,y}h_{x,y}$, and $e_{x,y} = g_{x,y}h_{x,y}$, are outer products of the gradients and errors, and the summation is performed over a neighborhood surrounding each pixel.[2] For better conditioning, we typically add a small stabilizing term λ to the diagonal elements of the left hand side, $a'_{x,y}$ and $b'_{x,y}$.

The local summation in (12) can also be implemented by computing the quantities $a_{x,y} \dots h_{x,y}$ in parallel, and then applying a convolution operator. For increased efficiency, a separable convolution with a box filter (using running sums) can be used.

[2]If we are working with color images, the computation is repeated for each color band and the results are summed to product the 2×2 and 2×1 matrices appearing in the normal equations.