Face Poser: Interactive Modeling of 3D Facial Expressions Using Facial Priors

MANFRED LAU
Carnegie Mellon University and Microsoft Research Asia
JINXIANG CHAI
Texas A&M University
and
YING-QING XU and HEUNG-YEUNG SHUM
Microsoft Research Asia

This article presents an intuitive and easy-to-use system for interactively posing 3D facial expressions. The user can model and edit facial expressions by drawing freeform strokes, by specifying distances between facial points, by incrementally editing curves on the face, or by directly dragging facial points in 2D screen space. Designing such an interface for 3D facial modeling and editing is challenging because many unnatural facial expressions might be consistent with the user's input. We formulate the problem in a maximum a posteriori framework by combining the user's input with priors embedded in a large set of facial expression data. Maximizing the posteriori allows us to generate an optimal and natural facial expression that achieves the goal specified by the user. We evaluate the performance of our system by conducting a thorough comparison of our method with alternative facial modeling techniques. To demonstrate the usability of our system, we also perform a user study of our system and compare with state-of-the-art facial expression modeling software (Poser 7).

Categories and Subject Descriptors: I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Interaction techniques*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Facial modeling and animation, statistical models, 3D modeling interfaces, sketching interfaces, optimization

ACM Reference Format:

Lau, M. Chai, J., Xu, Y.-Q., and Shum, H.-Y. 2009. Face poser: Interactive modeling of 3D facial expressions using facial priors. ACM Trans. Graph. 29, 1, Article 3 (December 2009), 17 pages. DOI = 10.1145/1640443.1640446 http://doi.acm.org/10.1145/1640443.1640446

1. INTRODUCTION

A long-standing challenge in computer graphics is to build an interactive and easy-to-use system that allows a naïve user to model realistic facial expressions quickly and easily. Applications of such a system include synthesizing natural facial expressions for characters in films, games, or other virtual environments, and for facial avatars in instant communication programs. Such a system is also useful as a rapid prototyping tool: the user can efficiently create 3D face expressions for reviewing, discussions, or educational purposes.

In this article, we present an intuitive and easy-to-use interface for modeling and editing 3D facial expressions in real time. Figure 1 illustrates the basic idea of our sketching interfaces. The system starts with a 3D facial model in a neutral expression. The user first selects a reference freeform stroke on the 2D screen space; the user then draws a target freeform stroke to indicate a desired deformation for the reference stroke. Both strokes can be drawn anywhere in the 2D screen space and be viewed from any viewpoint. The system automatically deforms the 3D facial model based on the user's input. The user can continue to refine the model until a desired facial expression is achieved.

Building such an interface for 3D facial modeling is difficult because the information from the user is often ambiguous. In our system, the user's inputs are often a small set of freeform strokes, facial points, or curves. This information is quite low-dimensional compared to a typical facial model, which is commonly represented by at least thousands of Degrees of Freedom (DoF). The user's inputs,

M. Lau is also with Microsoft Research Asia.
Authors' addresses: M. Lau, Department of Computer Science, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213; J. Chai, Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843; email: jchai@cs.tamu.edu; Y.-Q. Xu, H.-Y. Shum, Microsoft Research Asia, 5/F, Beijing Sigma Center, No. 49, Zhichun Road, Hai Dian District, Beijing, China 100190.

DOI 10.1145/1640443.1640446 http://doi.acm.org/10.1145/1640443.1640446

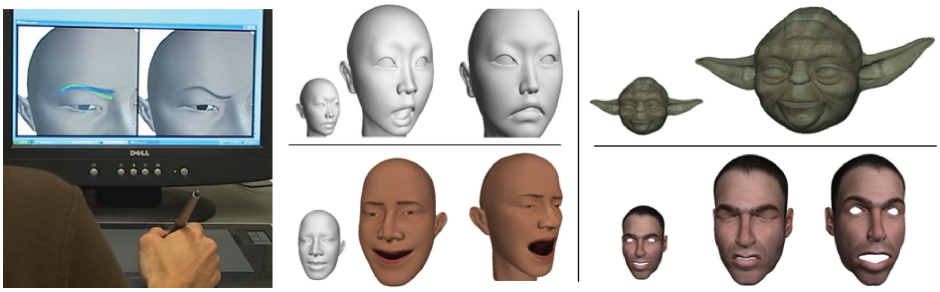


Fig. 1. Our face poser system allows intuitive modeling and editing of 3D facial expressions in real time.

therefore, cannot be used to fully determine a natural facial configuration because they will be consistent with many disparate solutions. Some solutions might correspond to unnatural facial expressions and not be what the user intends to model. We eliminate the ambiguity by imposing a prior embedded in a large database of prerecorded face expressions. The prior describes the "likelihood" function over facial expressions and measures how natural a face expression is.

We formulate the facial modeling problem in a probabilistic framework by combining the user's inputs with the facial prior embedded in precaptured facial expression data. We model the prior as a mixture of factor analyzers [Ghahramani and Hinton 1997] and learn the prior automatically from a prerecorded facial expression database. Maximizing the posteriori (i.e., likelihood of the facial expressions given the user's input) generates a natural and most likely facial expression that achieves the goal specified by the user.

We demonstrate the power and flexibility of this approach by interactive modeling of 3D facial expressions with point constraints, distance constraints, stroke constraints, and curve constraints. The constraints are specified in the 2D screen space. We thereby avoid the need for complex 3D interactions, which are common in mesh editing software and can be cumbersome for a naïve user. We have found that a first-time user can learn to use the system easily and quickly and be able to create desired facial expressions within minutes. Figure 1 shows some examples of facial expressions generated by novices.

We demonstrate potential applications of our system in trajectory keyframing, face expression transfer between different subjects, and expression posing from photos. We evaluate the quality of synthesized facial expressions by comparing against ground-truth data. In addition, we perform a thorough comparison of our method with alternative data-driven facial modeling techniques. Finally, we evaluate the usability of our system by performing a user study and comparing it against Poser 7.

1.1 Contributions

An initial version of our facial modeling framework has appeared previously in Lau et al. [2007]. While this article uses the same underlying probabilistic framework and the point/stroke constraints described in that paper, this article is a major improvement over the original one (approximately half of the material in this work is new). More specifically, this article incorporates the following major contributions/improvements over the original one.

—We have added *three new constraints* (fixed constraints, curve constraints, and distance constraints) for facial modeling and editing. These constraints were specifically added in response to comments by the users who previously tested our system. We show why each of these constraints is important by demonstrating what each one can do that others cannot. These constraints all together contribute to a *usable and complete system*. In addition, they are all designed to fit within a unified probabilistic framework.

—We have conducted a *thorough comparison* of our method against state-of-the-art facial modeling techniques, including blendshape interpolation, optimization with blendshapes, optimization in the PCA subspace, optimization in the PCA subspace with multivariate Guassian priors, and locally weighted regression. We compared cross-validation results for different types of constraints, different numbers of constraints, and different error measurements. These results not only evaluate the performance of our whole system, they also justify our use of the MFA model. Such a thorough comparison is also valuable to the whole facial modeling and animation community.

—We have introduced *two new applications* for our system, including facial expression transfer between different subjects and expression posing from photos.

—We have performed a *user study* to demonstrate the usability of our method and to compare with state-of-the-art facial modeling software (Poser 7). In addition, we have tested the system on *two additional facial models*: "Yoda" and "Simon".

2. BACKGROUND

In this section, we will discuss related work in sketching interfaces for 3D object modeling. Because we use prerecorded facial data in our system, we will also review research utilizing examples for modeling.

Our work is inspired by sketch-based systems that interpret the user's strokes for creating and editing 3D models [Zeleznik et al. 1996; Igarashi et al. 1999]. Zeleznik and his colleagues [1996] introduced a sketch-based interface to create and edit rectilinear objects. Igarashi and his colleagues [1999] developed the first sketching interface to interactively model and edit free-form objects. Recently, a number of researchers have explored sketch-based interfaces for mesh editing [Nealen et al. 2005; Kho and Garland 2005]. For example, Nealen and his colleagues [2005] presented a sketching interface for Laplacian mesh editing where a user draws reference and target curves on the mesh to specify the mesh deformation. Kho and Garland [2005] demonstrated a similar interface for posing the bodies and limbs of 3D characters. Yang and his colleagues [2005] presented a 3D modeling system to construct 3D models of particular object classes by matching the points and curves of a set of given 2D templates to 2D sketches. However, direct application of previous sketch-based modeling techniques in facial expression modeling might not work because facial

expression is a fine-grained deformation and even the slightest deviation from the truth can be immediately detected.

We present an easy-to-use sketching interface for 3D facial modeling and editing. Chang and Jenkins [2006] recently presented a similar sketching interface for facial modeling. However, their interface works only for strokes drawn as lines on the screen. Our stroke interface is more general, allowing for drawing any line, curve, shape, or region on the screen from any viewpoint. More importantly, our system learns facial priors from a large set of prerecorded facial expression data and uses them to remove the modeling ambiguity.

Our work builds upon the success of example-based facial modeling systems. Previous systems are often based on weighted combinations of examples in the original space [Parke 1972; Lewis et al. 2000; Sloan et al. 2001] or eigen-space [Blanz and Vetter 1999; Blanz et al. 2003]. These systems first compute weights from user-specified constraints or image data and then use the weights to linearly interpolate example data.

Zhang and his colleagues [2004] developed an example-based system for facial expression editing by interactively dragging points on the face. Their face model is represented as a linear combination of preacquired 3D face scans. Several researchers [Joshi et al. 2003; Zhang et al. 2003] also proposed to segment a face model into multiple regions and represented each subregion as a convex linear combination of blend shapes. Recently, Meyer and Anderson [2007] applied Principle Component Analysis (PCA) to a number of facial deformation examples and used them to select a small number of key points for facial deformation control. Example-based approaches have also been applied to edit skeletal mesh structures. For example, Sumner and Popovic [2005] and Der and his colleagues [2006] learned a reduced deformable space from a small set of example shapes, and used an inverse kinematics approach to optimize the mesh in a reduced deformable space. Their system allows the user to interactively deform a skeletal model by posing just a few vertices. Most recently, Feng and his colleagues [2008] combined a deformation regression method based on kernel Canonical Correlation Analysis (CCA) and a Poisson-based translation solving technique for easy and fast example-based deformation control.

An alternative way to compute the weights of examples is to reconstruct them directly from images or video [Blanz and Vetter 1999; Pighin et al. 1999; Blanz et al. 2003]. Blanz and his colleagues [2003] and Blanz and Vetter [1999] built a morphable model from 3D scans via Principal Component Analysis (PCA) [Bishop 1996] and applied the morphable model to reconstruct a 3D model from a single image. Pighin and his colleagues [1999] demonstrated that they can estimate the weights of 3D morphed face models directly from images or video. Chai and his colleagues [2003] presented a real-time vision-based performance interface for facial animation, which transforms a small set of automatically tracked facial features into realistic facial animation by interpolating the closest examples in a database at runtime.

The main difference between our work and previous example-based modeling systems is that we automatically learn a nonlinear probabilistic distribution function from a large set of prerecorded facial expression data. With a collection of locally linear submodels, our model (mixture of factor analyzers) can efficiently capture a nonlinear structure that cannot be modeled by existing linear models such as blendshapes or eigen-shapes. In addition, we formulate the facial modeling problem in a probabilistic framework by combining the user's inputs with the priors. This enables us to use a wide variety of intuitive constraints for facial modeling and editing, a capability that has not been demonstrated in previous facial modeling systems.

A number of researchers have also developed statistical models to solve the inverse kinematics problem for articulated human characters. For example, Grochow et al. [2004] applied a global nonlinear dimensionality reduction technique to human motion data and used the learned statistical pose model to compute poses from a small set of user-defined constraints. GPLVM works well for a small set of example data. However, its performance deteriorates rapidly as the size and heterogeneity of the database increases. Local statistical models are sufficient if the user provides continuous control signals (the performance animation problem). Recently, Chai and Hodgins [2005] constructed a series of local statistical pose models at runtime and reconstructed full body motion from continuous, low-dimensional control signals obtained from video cameras. Online local models are more appropriate for creating animations (a sequence of poses) from constraints that are known in advance, for example, when only the hand position will be constrained. They are not appropriate for our application because the user's input is not predefined and could be enforced at any facial points. We significantly extend the idea by constructing a statistical model from a large set of facial expression data and using it for interactive facial modeling and editing.

3. OVERVIEW

The main idea of our approach is that facial priors learned from prerecorded facial expression data can be used to create natural facial expressions that match the constraints specified by the user. The combination of the facial priors and the user-defined constraints provide sufficient information to produce 3D facial expressions with natural appearances.

3.1 Data Preprocessing

We set up a Vicon motion capture system [Vicon Systems 2007] to record facial movements by attaching 55 reflective markers to the face of a motion capture subject. We captured the subject performing a wide variety of facial actions, including basic facial expressions such as anger, fear, surprise, sadness, joy, and disgust, as well as other common facial actions such as speaking and singing. We scanned the 3D model of the subject and then converted the recorded marker motions into a set of deforming mesh models [Chai et al. 2003]. We translated and rotated each frame of the data to a default position and orientation because facial expression models should be irrelevant of head poses. We collected data for two subjects.

We denote a captured facial example in the database as $\mathbf{x} \in R^d$, where $\mathbf{x}$ is a long vector stacking the 3D positions of all facial vertices and d is three times the number of vertices in a facial model. Let M be the total number of facial examples for each subject. We first use Principal Component Analysis (PCA) [Bishop 1996] to preprocess the captured data and obtain a reduced subspace representation for $\mathbf{x}$. We have

$$\mathbf{x} = B \cdot \mathbf{p} + \overline{\mathbf{x}}, \tag{1}$$

where the vector $\mathbf{p} \in R^r$ is a low-dimensional representation of a facial model $\mathbf{x} \in R^d$. The matrix B is constructed from the eigenvectors corresponding to the largest eigenvalues of the covariance matrix of the data, and $\overline{\mathbf{x}}$ is the mean of all the examples. Due to the large dimensions of $\mathbf{x}$, we perform PCA by an incremental SVD method described by Brand [2002].

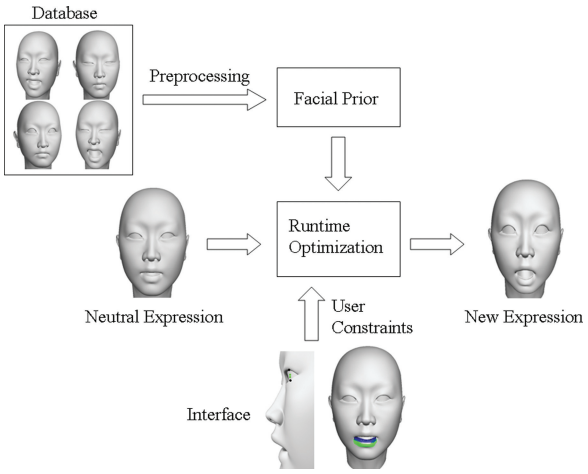


Fig. 2. System overview.

3.2 Problem Statement

We formulate our facial modeling problem in a Maximum A Posteriori (MAP) framework. From Bayes' theorem, the goal of MAP is to infer the most likely facial model $\mathbf{p}$ given the user's input $\mathbf{c}$. We have

$$\begin{aligned} \arg\max_{\mathbf{p}} \; pr(\mathbf{p}|\mathbf{c}) &= \arg\max_{\mathbf{p}} \; \frac{pr(\mathbf{c}|\mathbf{p})pr(\mathbf{p})}{pr(\mathbf{c})} \\ &\propto \arg\max_{\mathbf{p}} \; pr(\mathbf{c}|\mathbf{p})pr(\mathbf{p}) \end{aligned}, \tag{2}$$

where $pr(\mathbf{c})$ is a normalizing constant, ensuring that the posteriori distribution on the left-hand side is a valid probability density and integrates to one.

In our implementation, we minimize the negative log of $pr(\mathbf{p}|\mathbf{c})$, yielding the following energy minimization problem for facial expression modeling. We have

$$\hat{\mathbf{p}} = \arg\min_{\mathbf{p}} \quad \underbrace{-\ln pr(\mathbf{c}|\mathbf{p})}_{E_{likelihood}} + \underbrace{-\ln pr(\mathbf{p})}_{E_{prior}}, \tag{3}$$

where the first term, $E_{likelihood}$, is the *likelihood* term that measures how well a face model $\mathbf{p}$ matches the user-specified constraints $\mathbf{c}$, and the second term, E_{prior}, is the *prior* term that describes the prior distribution of the facial expressions. The prior term is used to constrain the generated facial model to stay close to the training examples. Maximizing the posteriori produces a natural facial mesh model that achieves the goal specified by the user.

3.3 System Components

The whole system contains three major components (see Figure 2).

User interaction. The user interaction step defines the likelihood term of the facial modeling framework. The system starts with a default facial model. The user interactively edits the facial model using different kinds of constraints: point, distance, stroke, and curve constraints. Furthermore, fixed constraints provide local control over the selected regions of the face. All of these constraints are specified in the 2D screen space. The interface allows the user to iteratively apply these constraints until a desired solution is obtained.

Facial priors. The system automatically learns a statistical model from precaptured facial data in the reduced PCA subspace. The constructed statistical model defines the prior term of the facial modeling framework and is used to constrain the generated facial expression to lie in the space of natural facial expressions.

Runtime optimization. The system solves the MAP problem at runtime. More specifically, the system uses gradient-based optimization techniques to automatically find a facial expression that best satisfies the user-specified constraints while matching the statistical properties of the captured data.

The facial priors are learned offline, while the other two stages are performed online based on the input from the user. We describe these components in detail in the next three sections.

4. USER INTERACTION

Our system starts with a default 3D facial model. In our implementation, we start with the mean expression $\bar{\mathbf{x}}$. The user can interactively edit facial expressions by specifying one or a combination of these types of constraints: point constraints, distance constraints, stroke constraints, and curve constraints.

For point constraints, the user can select any facial points and specify their desired positions in the screen space. For distance constraints, the user can select any two facial points and adjust the desired distance between them. For stroke constraints, the user selects a source freeform stroke on the screen and then draws a target freeform stroke to specify a desired deformation for the source stroke. For curve constraints, the user first selects a source curve by selecting and adjusting the control points of the curve. A target curve can then be incrementally edited by moving the control points. For fixed constraints, the user selects points on the face that should remain unchanged. This constraint must be used together with any of the other ones, thereby allowing for local control of the editing process. This section focuses on deriving the objective functions and their derivatives for all of these constraints.

We first derive the mathematical relationship between a facial model in the PCA subspace and the projection of a chosen vertex in the 2D screen space. Let $\mathbf{x}_i$ denote the 3D coordinates of the i^{th} vertex. Let $\mathbf{y}_i$ denote the 2D projection of the i^{th} vertex in the screen space. We have

$$\begin{pmatrix} u_i \\ v_i \\ \omega_i \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{r}_1^T & t_1 \\ \mathbf{r}_2^T & t_2 \\ \mathbf{r}_3^T & t_3 \\ \mathbf{0} & 1, \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x}_i \\ 1 \end{pmatrix}, \tag{4}$$

where f is the focal length of the virtual camera, $\mathbf{r}_i^T$ is the i^{th} row vector of the camera rotation matrix, and t_i is the corresponding camera translation component.

Let s_w and s_h denote the width and height of the 2D screen, respectively. We further have

$$\mathbf{y}_i = \begin{pmatrix} (\frac{u_i}{\omega_i} + 1) \cdot \frac{s_w}{2} \\ (1 - \frac{v_i}{\omega_i}) \cdot \frac{s_h}{2} \end{pmatrix} = \begin{pmatrix} \left(\frac{f\mathbf{r}_1^T\mathbf{x}_i + ft_1}{\mathbf{r}_3^T\mathbf{x}_i + t_3} + 1\right) \cdot \frac{s_w}{2} \\ \left(1 - \frac{f\mathbf{r}_2^T\mathbf{x}_i + ft_2}{\mathbf{r}_3^T\mathbf{x}_i + t_3}\right) \cdot \frac{s_h}{2}. \end{pmatrix}, \tag{5}$$

We can "select" the i^{th} vertex of $\mathbf{x}$ with

$$\mathbf{x}_i = W_i \cdot \mathbf{x}, \tag{6}$$

where W_i is a $3 \times 3N$ matrix, whose elements are zeros except

$$W_i(1:3, 3*i-2:3*i) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1. \end{pmatrix}, \tag{7}$$

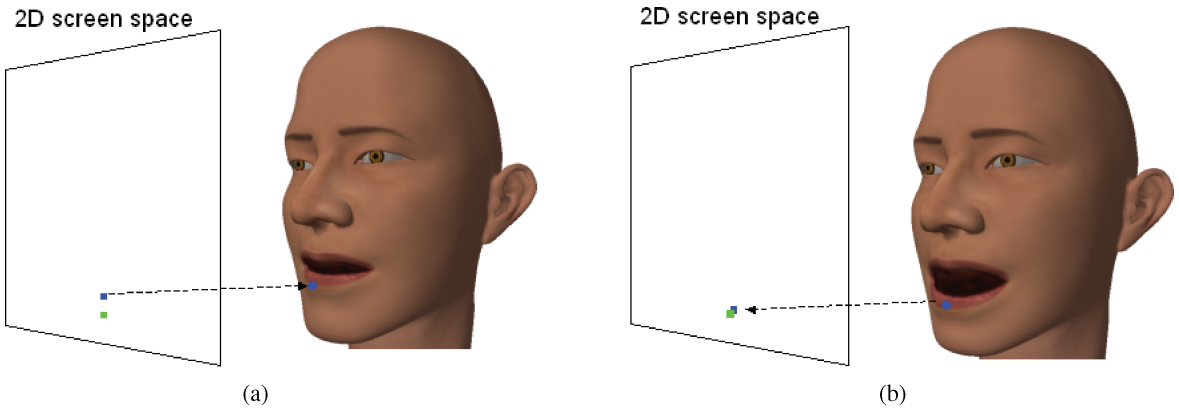


Fig. 3. Point constraints. (a) The user selects a 2D source pixel (blue) and a 2D target pixel (green) on the screen by raytracing techniques. (b) The problem is to find a face mesh so that the selected 3D vertex projects back onto the corresponding target pixel.

Combining Eqs. (1), (5), and (6) together, the 2D projection ($\mathbf{y}_i$) of the ith 3D facial vertex can be represented as a nonlinear function of the subspace facial models $\mathbf{p}$.

$$\begin{aligned}\mathbf{y}_i &= \mathbf{g}_i(\mathbf{p}) \\ &= \begin{pmatrix} \left(\frac{f\mathbf{r}_1^T W_i(B\mathbf{p}+\bar{\mathbf{x}})+ft_1}{\mathbf{r}_3^T W_i(B\mathbf{p}+\bar{\mathbf{x}})+t_3}+1\right)\cdot\frac{s_w}{2} \\ \left(1-\frac{f\mathbf{r}_2^T W_i(B\mathbf{p}+\bar{\mathbf{x}})+ft_2}{\mathbf{r}_3^T W_i(B\mathbf{p}+\bar{\mathbf{x}})+t_3}\right)\cdot\frac{s_h}{2} \end{pmatrix}\end{aligned} \tag{8}$$

4.1 Point Constraints

Point constraints allow the user to change the positions of individual vertices on the mesh. This enables the user to have detailed control over the final result. The user first selects a set of 3D source vertices $\{\mathbf{x}_i | i = 1, \ldots, N\}$ and then specifies a corresponding set of 2D target pixels $\{\mathbf{z}_i | i = 1, \ldots, N\}$ of where the vertices should map to on the screen (Figure 3(a)). The user selects each 3D point by picking a pixel in the 2D screen. We perform raytracing with this pixel to choose the point on the mesh. Given these inputs, the problem is to find a face model so that each selected 3D vertex ($\mathbf{x}_i$) projects onto the corresponding 2D screen position ($\mathbf{z}_i$) in the current camera view (Figure 3(b)).

Assuming Gaussian noise with a standard deviation of σ_{point} for the ith point constraint $\mathbf{z}_i$, we can define the likelihood term for the ith point constraint as follows.

$$\begin{aligned}E_{point} &= -\ln pr(\mathbf{z}_i|\mathbf{p}) \\ &= -\ln \frac{1}{\sqrt{2\pi}} \exp \frac{-\|\mathbf{y}_i-\mathbf{z}_i\|^2}{\sigma_{point}^2} \\ &\propto \frac{\|\mathbf{g}_i(\mathbf{p})-\mathbf{z}_i\|^2}{\sigma_{point}^2}\end{aligned} \tag{9}$$

A good match between the generated facial model ($\mathbf{p}$) and the user's input ($\mathbf{z}_i$) results in a low value for the likelihood term.

4.2 Distance Constraints

This constraint allows the user to select two facial vertices $\mathbf{x}_i$ and $\mathbf{x}_j$, and edit the distance between them in the 2D screen space. This is particularly useful for changing the width and height of the eyes or mouth. Figure 4 shows an example of the user's input. The user selects the 3D vertices by selecting 2D pixels in the same way as the point constraints. The current distances between the pairs of vertices are displayed, and the user can dynamically adjust the distances.

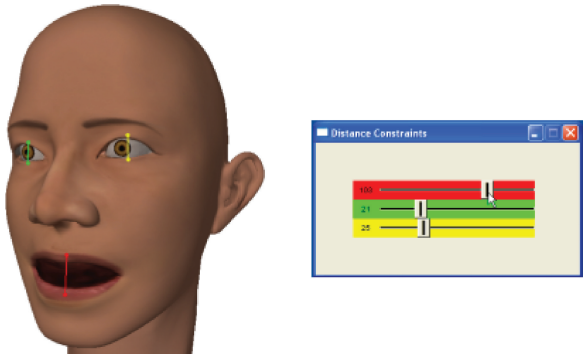


Fig. 4. Distance constraints. As input, the user selects pairs of vertices and specifies the distances between each pair.

Let d denote the user-defined target distance, and let $\mathbf{x}_i$ and $\mathbf{x}_j$ map to $\mathbf{y}_i$ and $\mathbf{y}_j$ respectively. Similarly, we assume Gaussian noise with a standard deviation of σ_{dist} for the distance constraint d, and we define the following likelihood term for the distance constraints.

$$\begin{aligned}E_{distance} &= -\ln pr(d|\mathbf{p}) \\ &= -\ln \frac{1}{\sqrt{2\pi}} \exp \frac{-(\|\mathbf{y}_i-\mathbf{y}_j\|-d)^2}{\sigma_{dist}^2} \\ &\propto \frac{(\|\mathbf{g}_i(\mathbf{p})-\mathbf{g}_j(\mathbf{p})\|-d)^2}{\sigma_{dist}^2}\end{aligned} \tag{10}$$

4.3 Stroke Constraints

This constraint allows the user to select a group of 3D points and specify where these points should collectively project to on the screen. This is designed to allow the user to make large-scale changes to the mesh with minimal user interaction. More specifically, the user first draws a 2D source stroke to select a set of 3D points ($\mathbf{x}_s$'s) on the mesh. Then the user draws a 2D target stroke to provide a region of pixels ($\mathbf{z}_j$'s) where the 3D points should project to. Figure 5 shows some examples of user-drawn strokes.

Given a source stroke in the 2D screen space, we need to find the corresponding 3D points on the mesh efficiently. We raytraced the pixels of the source stroke in a hierarchical manner. We first consider the selected pixel region as blocks of 15 by 15 pixels, and find the triangles that intersect with each block. We then consider these blocks as 3 by 3 pixels, and find the triangles that intersect with

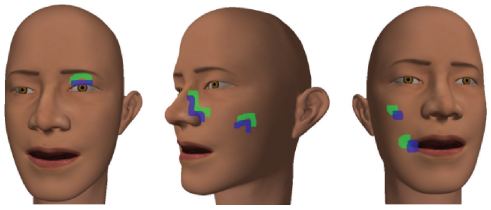


Fig. 5. Stroke constraints: different examples of source strokes (blue) and target strokes (green).

each block. For each 3 by 3 block, we only test the triangles that intersected with the corresponding 15 by 15 block. Finally, we ray-traced each pixel by testing only the triangles in the corresponding 3 by 3 block. This process selects the $\mathbf{x}_s$'s on the mesh. This process is necessary in order to allow for interactive selection of the points on the mesh. Without this hierarchical selection, it can take up to tens of seconds to select the points for each source stroke.

Since the selected 3D points on the mesh do not have to be the original vertices of the mesh, we store the barycentric coordinates of each $\mathbf{x}_s$. The position of each $\mathbf{x}_s$ depends on the positions of the three vertices of the triangle ($\mathbf{x}_u$, $\mathbf{x}_v$, $\mathbf{x}_w$) that it belongs to. When the face mesh deforms, the position of each $\mathbf{x}_s$ is recomputed as

$$\mathbf{x}_s = u \cdot \mathbf{x}_u + v \cdot \mathbf{x}_v + w \cdot \mathbf{x}_w, \tag{11}$$

where u, v, and w are the barycentric coordinates. For a given $\mathbf{x}_s$, these barycentric coordinates are fixed, while the vertices ($\mathbf{x}_u$, $\mathbf{x}_v$, $\mathbf{x}_w$) may deform.

Unlike point constraints, the stroke constraints do not provide correspondence information between the individual pixels of the source and target strokes. We could have estimated a correspondence between the individual pixels of the two strokes and then apply the point constraints approach to each pair of pixels. However, this is intuitively not what the user draws. Hence we decided to project the set of 3D points on the mesh ($\mathbf{x}_s$'s) back to the 2D screen for comparison with the target stroke ($\mathbf{z}_j$'s).

Let the projected region of pixels be $\mathbf{y}_i$'s. Let R be a constant region that contains both $\mathbf{y}_i$'s and $\mathbf{z}_j$'s. For efficiency, we choose the region R to be a joint region of $\mathbf{y}_i$'s and $\mathbf{z}_j$'s. Let I_y be a binary image over the region R whose pixel values are zeros except the projected region of pixels, $\mathbf{y}_i$'s. I_y, therefore, depends on $\mathbf{x}_s$'s. We further define I_z to be a binary image over the region R whose pixel values are zero except the target stroke, $\mathbf{z}_j$'s.

Similarly, we assume Gaussian noise with a standard deviation of σ_{stroke} for stroke constraints. The likelihood term for stroke constraints is

$$E_{stroke} = \sum_{r \in R} \frac{(I_y(\mathbf{y}_i) - I_z(r))^2}{\sigma^2_{stroke}}. \tag{12}$$

4.3.1 *Additional Distance Term for Stroke Constraints.* Our method of measuring the distance between the two strokes by computing the intensity difference is efficient. However, if the two strokes are far away from each other, the objective function in Eq. (12) does not require the strokes to "move toward" each other (see Figure 6). The energy value in Eq. (12) will reach a local minimum without "moving" the strokes. There are many ways to solve this problem. We can use a more sophisticated function to represent I_y and I_z. For example, a signed distance function would allow the strokes to "move toward" each other. But computing this function at every iteration of the optimization would be time consuming. Instead we choose to add an additional likelihood term to minimize the distance between the center of the pixels in the source stroke and the center of the pixels in the target stroke. We have

Fig. 6. If the source and target strokes are originally far away from each other, the objective function in Eq. (12) does not allow the two strokes to "move toward" each other. This motivates the need for the additional distance term for stroke constraints.

$$E_{extra} = \frac{\| \frac{\sum \mathbf{y}_i}{N_s} - \overline{\mathbf{z}} \|^2}{\sigma^2_{extra}}, \tag{13}$$

where σ_{extra} is the standard deviation for the extra constraints, N_s is the number of pixels in the source stroke, $\overline{\mathbf{z}}$ is the center of the pixels in the target stroke, and the summation is for all source pixels $y_i, i = 1, \ldots, N_s$.

4.4 Incremental Editing: Curve Constraints

Stroke constraints do not allow for incremental editing of the user-drawn strokes. This motivates curve constraints: the user can draw and adjust control points of the source and target curves. This allows for local and incremental editing of the curves and therefore the solution mesh. More specifically, the user first selects and adjusts the Bezier control points of the source curve (Figure 7(a)). The user then accepts the source curve and adjusts the control points to form the target curve (Figure 7(b)). Given the source and target curves, we can use the same framework to deform the face mesh in many different ways. In our implementation, we consider the curves as strokes and apply the same idea for stroke constraints to generate the solution.

Curve constraints allow for local and incremental control compared to stroke constraints. Local control is achieved since the Bezier control points only change the local parts of the curve. We also have incremental editing because we can continue to adjust the control points of the curve until we have a desired outcome.

The user-selected control points deterministically define a curve. These points are a subset of the Bezier control points defining the curve and they lie on the curve. The other Bezier control points do not lie on the curve. Bezier curves were chosen because the user-selected control points can lie on the curve, and this leads to a more intuitive interface. Moreover, Bezier control points can generate smooth piecewise curves. We construct piecewise Bezier curves such that the first and second derivatives of adjacent piecewise curves are the same. In addition, we construct the piecewise curves so that the second derivatives of the start and end points of the whole curve is zero. This is known as the "relaxed" property of Bezier curves and allows us to deterministically generate the whole curve from the user-selected points.

We take the user-selected points and compute the curve by first computing the corresponding relaxed B-spline points and then the relaxed Bezier points. We show the method with the example in

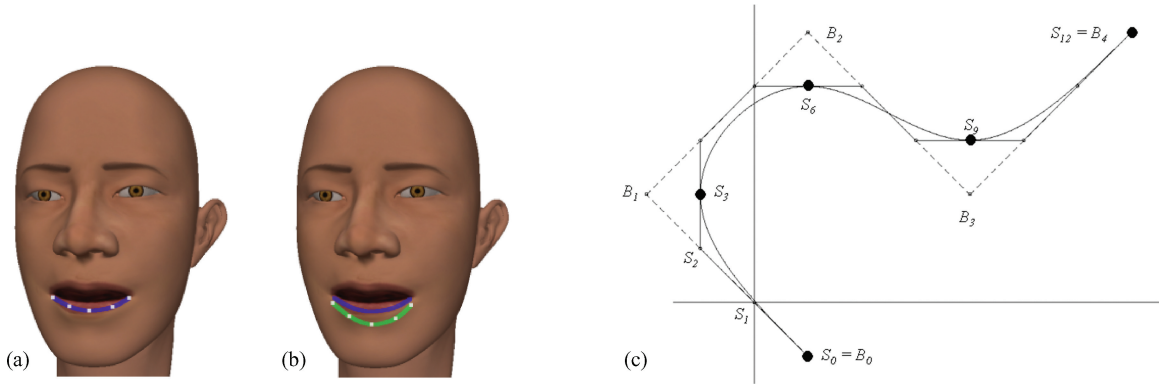


Fig. 7. Curve constraints. (a) The white points are Bezier control points of the blue source curve. The user first selects and adjusts the positions of these points on the 2D screen. (b) Once the user accepts the source curve, the same control points can be adjusted locally to form the target curve. (c) An example to show the construction of the curve from the user-selected control points (large black points).

Figure 7(c). First, the user-selected points (S_0, S_3, S_6, S_9, S_{12}) are used to compute the B-spline points (B_0, B_1, B_2, B_3, B_4).

$$\begin{pmatrix} 4 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 4 \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} 6S_3 - S_0 \\ S_6 \\ 6S_9 - S_{12} \end{pmatrix} \tag{14}$$

Also, $B_0 = S_0$ and $B_4 = S_{12}$. The B-spline points are then used to compute the Bezier points (the $S_i's$ that are not the user-selected points) that are not on the curve. For example, B_0 and B_1 are trisected to form S_1 and S_2. The Bezier points are then used to generate piecewise Bezier curves. For example, S_0, S_1, S_2, and S_3 are the Bezier control points that generate the curve from S_0 to S_3.

4.5 Local Control: Fixed Constraints

If the user edits the mesh with one of the previous constraints, the regions of the mesh that the user did not select are "soft" regions, in the sense that they may change or stay the same depending on the motion capture data. For example, if the eye region is selected and changed, the mouth may also change even if those vertices were not selected. We can use fixed constraints to allow some vertices to maintain their positions as much as possible in the optimization. In this case, the user can choose some of the vertices near the mouth to be "fixed."

More specifically, the user selects N_F total number of fixed vertices ($\mathbf{x}_i$'s). Let $\mathbf{x}_{i\_original}$ be the original 3D positions of these vertices. Similarly, the likelihood term for the fixed constraints defined on the ith vertex is

$$E_{fixed} = \frac{\|\mathbf{x}_i - \mathbf{x}_{i\_original}\|^2}{\sigma_{fixed}^2}, \tag{15}$$

where σ_{fixed} is the standard deviation for the fixed constraints. We can substitute Eqs. (1) and (6) here. Fixed constraints must be used with at least one of the other constraints described before. The overall objective function will have this term multiplied by a user-defined weight added to it. The larger the weight is, the more these vertices will try to stay in place.

5. FACIAL PRIORS

There might be many facial models that satisfy the user-defined constraints. For example, when the user selects one facial vertex to edit the whole model, there might be many results that are consistent with this constraint. To remove ambiguities, we can constrain the generated model to lie in the space of natural facial expressions by imposing a prior on the generated model.

We model the prior as a Mixture of Factor Analyzers (MFA) [Ghahramani and Hinton 1997] and construct the model automatically from a prerecorded facial expression database. The MFA model learns a Probability Density Function (P.D.F.) in the PCA subspace that provides a facial prior to measure the naturalness of facial expressions. The MFA model has also been successfully applied to model the prior for many high-dimensional nonlinear data such as handwritten digits [Hinton et al. 1997] and images [Bishop and Winn 2000].

A single Factor Analyzer (FA) assumes that an observed r-dimensional variable $\mathbf{p}$ is generated as a linear transformation of some lower q-dimensional latent variable $\tau \sim \mathcal{N}(0, I)$ plus additive Gaussian noise $\omega \sim \mathcal{N}(0, \Psi)$. Ψ is a diagonal matrix. The generative model can be described as

$$\mathbf{p} = A\tau + \omega + \mu. \tag{16}$$

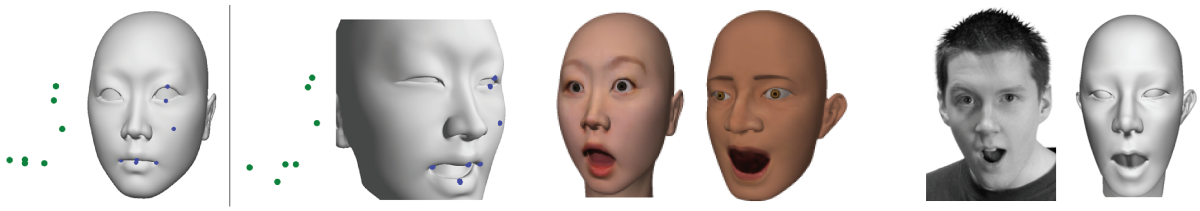
Here, $A \in R^{r \times q}$ is a factor loading matrix. μ is a mean vector. The P.D.F. of the observed data in an FA model can be obtained by

$$pr(\mathbf{p}; \Theta) = \mathcal{N}(\mu, AA^T + \Psi). \tag{17}$$

A Mixture of Factor Analyzers (MFA) is defined by a linear combination of K factor analyzers and can be thought of as a reduced dimension mixture of Gaussians. The MFA model extracts q-dimensional locally linear manifolds underlying the given high-dimensional data. The P.D.F. of the observed data by a mixture of factor analyzers is given by

$$pr(\mathbf{p}; \Theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}\Big(\mu_k, A_k A_k^T + \Psi\Big), \tag{18}$$

where π_k is a mixing proportion ($\pi_k > 0$ and $\sum_{k=1}^{K} \pi_k = 1$). The goal of our offline learning process is to automatically find the model parameters $\Theta = \{\pi_k, \mu_k, A_k, \Psi | k = 1, \ldots, K\}$ from the training data. We use an EM algorithm to fit a mixture of factor analyzers. The detailed description of the EM algorithm can be found in Ghahramani and Hinton [1997].



(a) (b) (c)

Fig. 8. Applications. (a) Trajectory keyframing: we use seven 2D target points (green) to generate the 3D model (blue points are corresponding 3D source points). (b) Expression transfer: the facial expression model on the left is the source expression, and the one on the right is the transferred facial expression. (c) Expression posing: the facial model on the right is created with the photo.

We minimize the negative log of $pr(\mathbf{p})$, yielding the energy formulation for the prior term.

$$E_{prior} = -\ln \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\mu_k, A_k A_k^T + \Psi\right) \quad (19)$$

A smaller E_{prior} value means that $\mathbf{p}$ is closer to the samples in the motion data and therefore more natural. The inverse and determinant of the covariance matrices are precomputed for each factor to achieve a faster runtime.

6. RUNTIME OPTIMIZATION

During runtime, the system optimizes in the reduced PCA subspace and finds the 3D face model ($\mathbf{p}$) that best satisfies the user-defined constraints ($\mathbf{c}$).

The overall objective function is a combination of various forms of the likelihood terms (Eqs. (9), (10), (12), (13), and (15)) and the prior term (Eq. (19)):

$$\arg\min_{\mathbf{p}\in R^r} E_{point} + E_{distance} + E_{stroke} + E_{extra} + E_{fixed} + E_{prior} \quad (20)$$

Note that curve constraints are represented here as part of the stroke constraints term. Each factor in the MFA model has a mean face mesh. We initialize the optimization with the "best" mean among all factors, by explicitly computing the "best" $\mathbf{p}$.

$$\arg\min_{\mathbf{p}\in\{\mu_1,\ldots,\mu_K\}} E_{point} + E_{distance} + E_{stroke} + E_{extra} + E_{fixed} + E_{prior} \quad (21)$$

We analytically evaluate the Jacobian terms of the objective function (see Appendix) and minimize the function using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm in the GSL library [Galassi et al. 2003]. The solution converges rapidly due to a good starting point, a low-dimensional optimization space, and the analytical evaluation of the Jacobian terms.

7. APPLICATIONS

In order to explore the effectiveness of our system, we also tested the system in the following applications.

Trajectory keyframing. Our optimization framework can be applied for trajectory keyframing. Given the 3D or 2D trajectories of a few points, we can reconstruct the 3D facial expressions using point constraints. If we just have the 2D trajectories, we also need the camera viewpoint as input. Some examples are shown in Figure 8(a). When we generate many frames for an animation, we add a smoothness term to minimize the change in velocity of the vertices between consecutive frames.

Table I. Details of the Data We Used

	First Subject	Second Subject
M	7,790	10,036
d	51,333	49,503
r	20	15
K	30	20
q	10	6

M is the number of samples, d is the number of original dimensions (d equals 3 times the number of vertices), r is the number of reduced dimensions in the PCA step, K is the number of factors, and q is total number of dimensions for the latent variable τ_k in each factor of the MFA model.

Expression transfer. We can apply our system to transfer facial expression from one subject (e.g., "A") to another (e.g., "B"). More specifically, we extract a set of distance constraints from subject "A" and use them as well as the facial prior of subject "B" to generate a corresponding facial expression for subject "B". Figure 8(b) shows some examples. These results are generated with three distance constraints, controlling the height of the mouth and eyes. Since the meshes are different for the two subjects, we scale the target distances between subjects by measuring the minimum and maximum of key features of the face. The results show that face expressions can be realistically transferred between different subjects.

Expression posing from photos. We can take photos of a human face and recreate their expressions with our system. Figure 8(c) shows an example. In this case, our system may not achieve detailed correspondences between the photo and facial model, but it allows for quick prototyping of a 3D face model.

8. RESULTS

Table I shows the details of the main datasets that we used. For the first subject, the 3D face model has 17,111 vertices and 34,168 faces. For the second subject, the 3D model has 16,501 vertices and 32,944 faces. The number of reduced dimensions for the first subject and second subject are 20 and 15, respectively, by keeping 99% of the energy in the largest eigenvalues from PCA.

8.1 User Interaction

Our system can generate a variety of facial expressions. The accompanying video shows live screenshots of the interactive system.

Point constraints. The user can select a facial point and interactively drag it into a desired position in 2D screen space (see video for example). Even though the facial points are in 3D, this "dragging" in 2D screen space is still natural and generates good results. We can achieve a more detailed result if we increase the number of

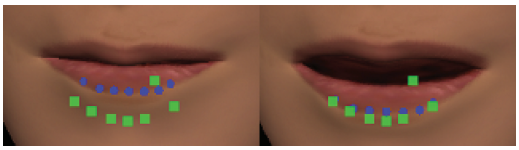


Fig. 9. If there are many constraints, some of them may conflict or "fight" with each other, since the system also tries to generate natural results. The inputs are shown on the left, and the synthesized result is on the right.

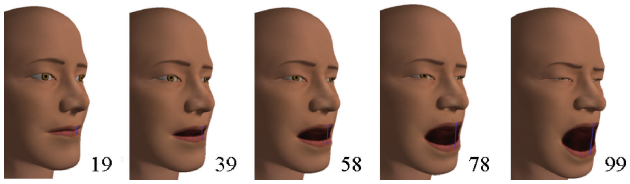


Fig. 10. Distance constraints. We can move between adjacent results interactively with just one mouse click. The mouth's height measured by the number of pixels is shown beside each model.

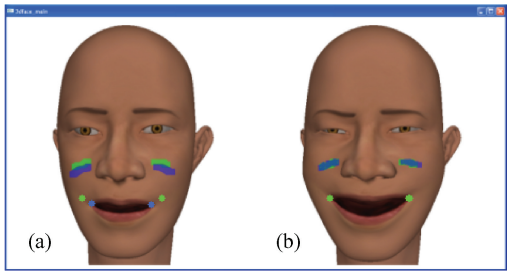


Fig. 11. Stroke and point constraints. (a) Original mesh: the user draws point and stroke constraints directly on the screen. The source points/strokes are in blue and target points/strokes are in green. (b) Result: the face model is deformed to satisfy the constraints.

constraints. In general, as the number of point constraints increases, the results are more accurate (see Section 9). However, if there are constraints that do not match any natural facial expressions represented by our statistical model, the solution will fail to satisfy all the constraints (Figure 9).

Distance constraints. Figure 4 shows a slide bar interface for controlling the distances between pairs of points. Distance constraints are particularly useful for interactively changing the height and width of the mouth and eyes. Figure 10 shows results that distance constraints can more easily generate compared to other constraints. Even though we can create an "opened" mouth with other constraints, other constraints cannot iteratively and interactively generate these types of solutions.

Stroke constraints. This constraint can be used to efficiently make large-scale changes to the face model. Figure 11 shows an example where we draw strokes to "raise" the cheek regions to create a smiling expression. The point constraints are used in this case to fine-tune the corners of the mouth. The users have reported that the sketching interface allows them to draw certain features of the face such as an eyebrow more easily (Figure 1(left)). Although some results might also be achieved with point constraints, such a pen-based sketching interface is more intuitive and more efficient than selecting a large number of points on the screen (with point constraints). Figure 5 shows different examples of strokes that can be drawn easily with simple click-and-drag operations of the mouse.

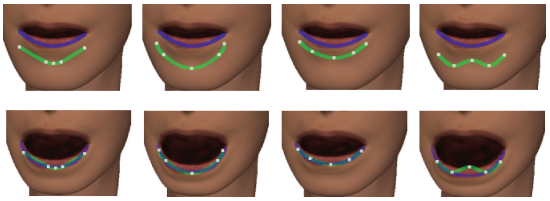


Fig. 12. Curve constraints. The top row shows the inputs and the bottom row shows the results. Each column is a different example. The source curves are the same in all four cases. Different shapes of the mouth can be obtained by curve constraints.

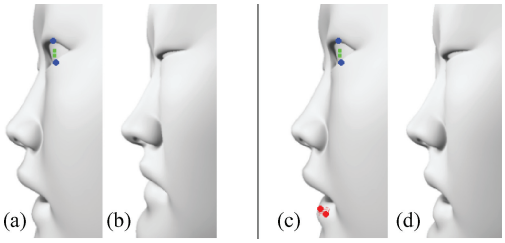


Fig. 13. Local control of the face. (Left) Without fixed constraints: only point constraints are selected in (a) and the result is shown in (b). (Right) With fixed constraints: fixed constraint (red vertices) are added in (c) and the result is shown in (d), where the eyes are closed as before while the mouth remains open.

Incremental editing: curve constraints. Curve constraints can handle local and incremental edits. Local changes can be made because the Bezier control points have a local control over the curve. We can make incremental edits to these points until a desired outcome is achieved. Figure 12 shows some examples. While the strokes in stroke constraints cannot be edited incrementally, curve constraints allow for these types of changes.

Local control: fixed constraints. Figure 13 shows the use of this constraint for local or regional control of the face. If the user is already satisfied with certain parts of the face, those parts can be selected to be "fixed" while the rest of the face is being edited. Using just the other constraints cannot provide such regional control over the whole face. The user can dynamically adjust the weight of the fixed constraint term ($1/\sigma^2_{fixed}$) in the optimization, thereby choosing a spectrum of possibilities that trade off between satisfying the "fixed" vertices and satisfying the other constraints.

8.2 Computation Time

The computation time of our method is efficient (see Figure 14). For example, the runtime for stroke constraints with 900 pixels (about the size of a stroke drawn on the eyebrow) is about 0.35 seconds. For fixed constraints, we measure the time with 5 point constraints together with different numbers of "fixed" points. These experiments were done on a Pentium 4 3GHz machine with 1GB RAM. Each value is an average over 5 trials.

As the number of constraints (points, distances, pixels, or fixed points) increases, the computation time increases linearly. This is expected as the optimization (and also Jacobian) term is a summation of the objective value for each constraint. In addition, the execution time of the system does not depend on the number of

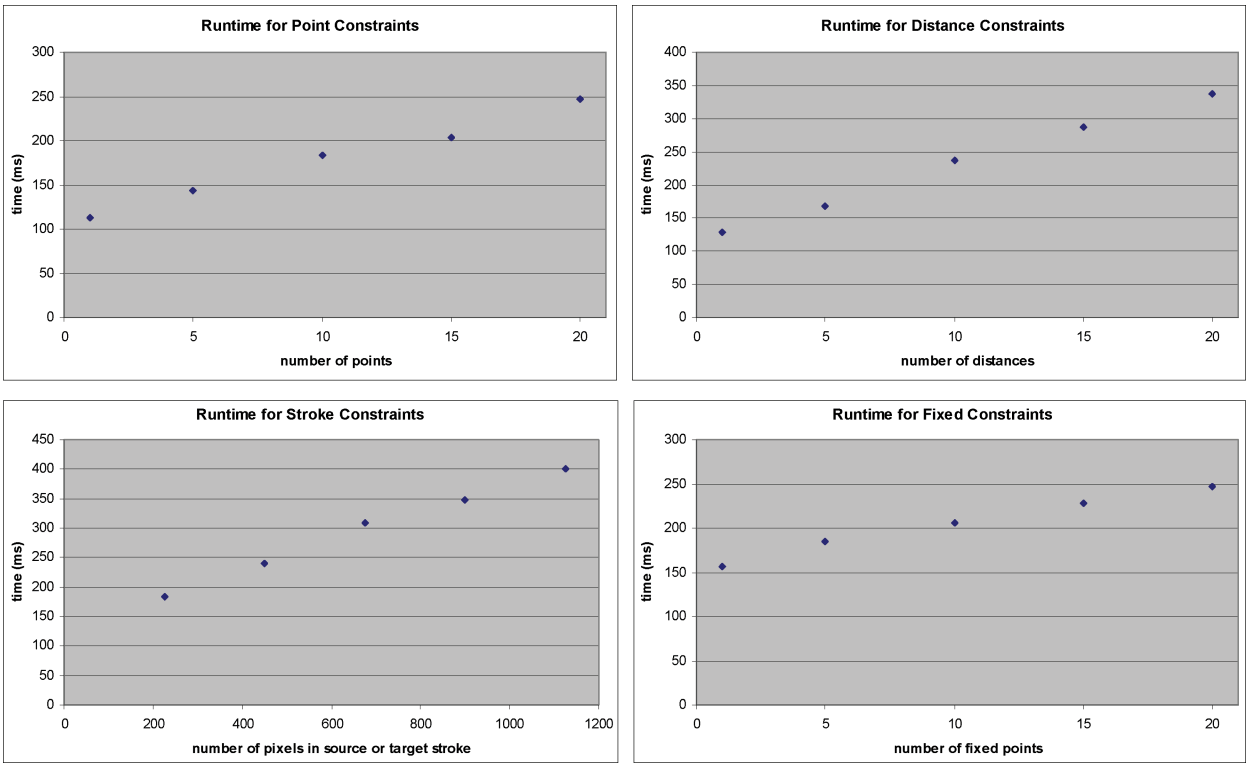


Fig. 14. Computation time for each constraint. As the number of constraints increases, the computation time increases linearly.

Fig. 15. These results show that our method works on a variety of face data. (Left) We can edit a 3D yoda model by sketching on the 2D screen. (Right) Our system works with the "Simon" head model from the Poser 7 software.

samples in the database because the MFA model is learned offline. Hence our method scales well to the size of the database.

8.3 Additional Test Data

Our method is not dependent on the number of vertices or the mesh structure. This section shows that our method works on different types of face data. Specifically, we have tested our system with a yoda head model and the "Simon" head model from the Poser 7 software.

We acquired a yoda head model (see Figure 15(left)), and manually created a number of different face expressions using the popular Maya software. Since we do not have any motion data available for this yoda model, we created thirty examples with Maya ourselves. The limitation is that we cannot create poses that are far from these original examples. The purpose is therefore to show that our method does work on mesh structures that are quite different from human head models, and our method does work surprisingly well even with a limited amount of data. The yoda model has 6,916 vertices and 13,788 faces. We reduced the number of dimensions from 20,748 (three times the number of vertices) to just 5 using PCA (keeping greater than 98% of energy).

In addition, we tested our system with the "Simon" head model (see Figure 15(right)) from the Poser 7 software. We directly exported the mesh and texture from this software. For the head model, we did not use the eyeballs, tongue, and teeth that are in the original Poser 7 model. We created thirty examples of different types of face expressions with the face morphs available in Poser 7. We also triangulated each of these meshes in Maya. Since the original thirty examples are created with the face morphs in Poser 7, one initial concern was that the expressions that we can create with these examples in Face Poser would already be similar to the expressions

that can be created in Poser 7. Our comparison of the resulting 3D models in Figure 22 might therefore be unfair. However, we found that our Face Poser system is able to interpolate, extrapolate, and/or blend between the original samples. Thus, this was not a major issue. The "Simon" head model that we use has 8,562 vertices and 16,952 faces. We reduced the number of dimensions from 25,686 (three times the number of vertices) to just 5 using PCA (keeping greater than 99% of energy).

9. EVALUATION

In this section, we provide a thorough comparison of our method against alternative data-driven techniques: blendshape interpolation (Interp-all), optimization with blendshapes (Opt-blend), optimization in the PCA subspace (PCA), optimization in the PCA subspace with a multivariate Gaussian distribution for facial priors (Probabilistic PCA or PPCA), and locally weighted regression (LWR). We compare cross-validation results for different types of user constraints, for different numbers of each constraint, and for different error measurements.

9.1 Explanation of Other Techniques, Cross-Validation, 3D and 2D Error

We compare the MFA model used in this work against other methods. The alternative techniques can be classified into two groups.

Interpolation. "Interp-all" takes all the examples in the database, uses the objective function (i.e., Eq. (9) for point constraints) to compute an appropriate weight for each example, and blends the examples with the computed weights [Parke 1972; Lewis et al. 2000; Sloan et al. 2001]. "LWR" extends the idea of "Interp-all" by considering closest examples in the database [Chai et al. 2003]. It finds k examples that are closest to the user's input and performs a locally weighted regression based on the closest examples.

Optimization. "Opt-blend" represents the solution as a weighted combination of examples in the database and optimizes the weights based on user-defined constraints [Zhang et al. 2004]. "PCA" significantly reduces the solution space of "opt-blend" by performing an optimization in the PCA subspace [Blanz and Vetter 1999; Blanz et al. 2003]. "PPCA" further reduces the solution space by incorporating a "Gaussian" prior term into the objective function. "PPCA" assumes that the examples in the database are represented by a multivariate Gaussian distribution. All optimization methods start with the same initial pose.

For cross-validation tests, we use new face models as testing data. We start from a neutral pose and select the source constraints (points, distances, strokes). The corresponding target constraints are automatically generated based on the ground-truth test data. For example, if 3D point x is chosen and it maps to the 2D pixel y for the test data, y is used as the target constraint for the original neutral pose. Given the source and target, we can then generate the solution and compare it against the ground-truth data (i.e., the test data).

The 3D error is the average of the Euclidean distances between each vertex of the ground-truth mesh and the synthesized mesh. For point constraints, the 2D error is the difference between the projected pixel (from the selected source vertex) and the target pixel. If there is more than one pair of source vertices and target pixels, we average these differences. This is essentially Eq. (9) with $\mathbf{p}$ being the solution mesh. The 2D error for the other constraints are computed similarly by their respective optimization objective functions. Each value reported for 3D error, 2D error, and runtime is an average over 10 trials.

9.2 Comparison of 3D Reconstruction Error

The left side of Figure 16 shows the comparison results of 3D errors for point, distance, and stroke constraints, respectively. As the number of constraints increases (in all cases), the 3D error decreases. This is because the increase in the number of constraints provides more information about the ground-truth sample. This leads to a more accurate result in the solution mesh compared to the ground truth, and therefore the 3D error decreases.

"Opt-blend" and "Interp-all" do not produce good results because the solution depends on all the samples in the database. PCA produces a larger error because the number of user-specified constraints is usually small even when compared to the reduced dimensions of the PCA subspace. Hence the constraints are not sufficient to fully determine the weights of $\mathbf{p}$. PPCA can remove the mapping ambiguity from the low-dimensional constraints to the reduced subspace dimensions by the facial prior. However, the face models are not as well approximated by a multivariate Gaussian distribution. For LWR, the model provided by the k closest samples are usually not sufficient because they do not provide any temporal coherence. Our method (MFA) produces a smaller 3D error both across the three types of constraints and across the different number of points, distances, and pixels for each constraint.

LWR and PCA tend to be closest to our method in terms of the cross-validation 3D errors. We therefore present some visual results of these two techniques compared with our method. Figures 17, 18, 19, and 20 show the side-by-side comparisons for different examples: MFA can produce better perceptual results than LWR and PCA.

9.3 Comparison of 2D Fitting Error

The right side of Figure 16 shows the comparison results of 2D fitting errors for point, distance, and stroke constraints, respectively. As the number of constraints increases (in all cases), the 2D error remains approximately the same. This is because the 2D error is an average over the number of points, distances, and pixels for the three types of constraints.

LWR and PCA also tend to be closest to our method in terms of the 2D errors. We again refer to Figures 17, 18, 19, and 20 to show MFA can produce better visual results than LWR and PCA.

In particular, the 2D error for PCA is usually smaller than MFA. This is because PCA exactly tries to minimize this error without considering how natural the result might be. Figure 21 shows an example where PCA has a zero 2D error, but it produces an unnatural result. For the same inputs, MFA has a 2D error that is relatively large, but it produces a natural solution. Therefore, even though PCA can have a lower 2D error than MFA, the visual solutions that it produces may not be desirable.

9.4 Comparison of Runtime

We show a comparison of the runtime of point, distance, and stroke constraints for all the techniques (Table II). We use 4 points, 4 distances, and 450 pixels for each constraint, respectively. The runtime for MFA is comparable to PCA and PPCA, while being significantly better than the other three techniques.

"Interp-all" and LWR are very inefficient since these methods have to iterate through every sample in the database. "Opt-blend" is even more inefficient because the number of parameters in the optimization problem is equal to the total number of samples. PPCA,

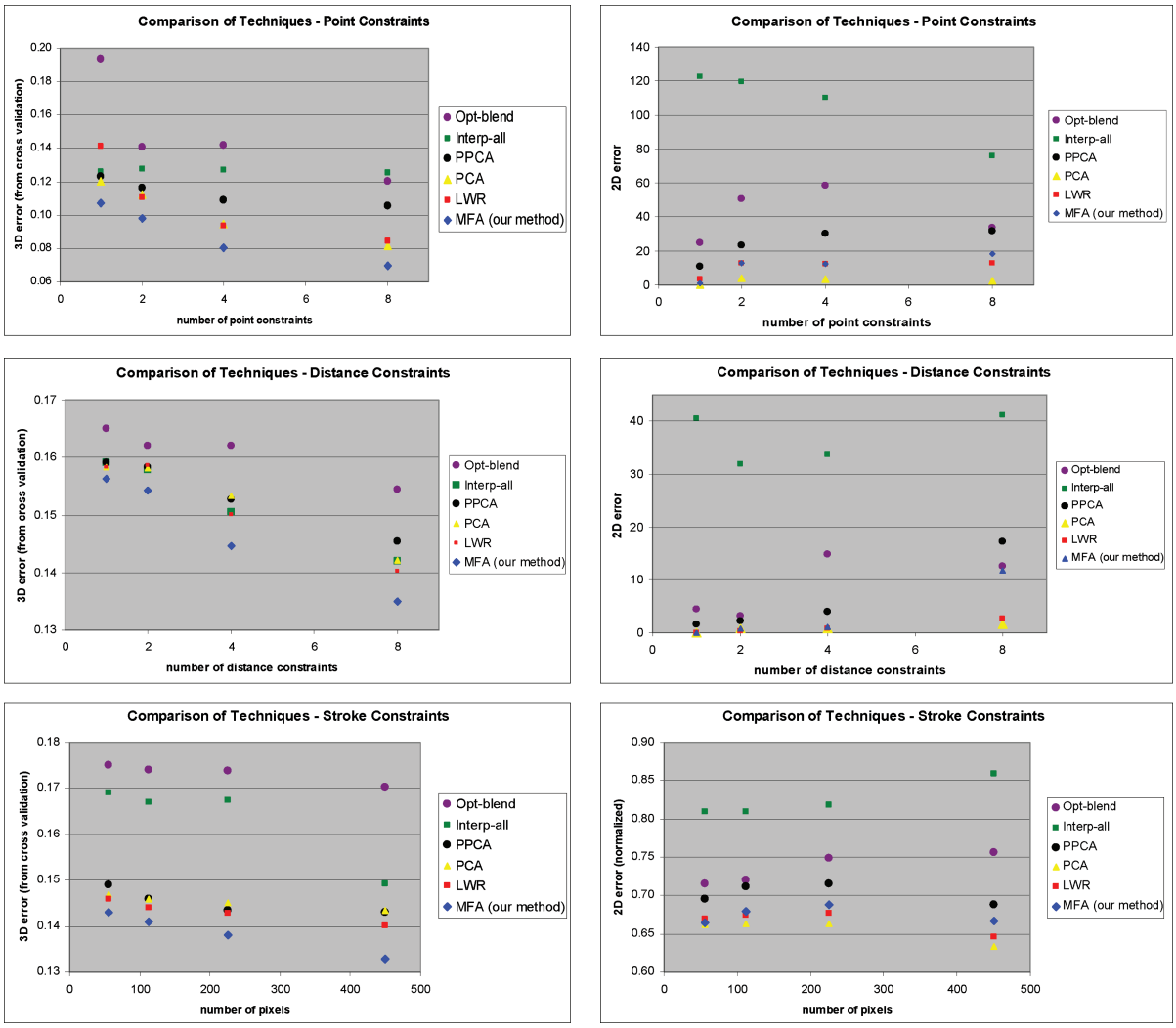


Fig. 16. Comparison with other techniques. (Top) Point constraints. (Middle) Distance constraints. (Bottom) Stroke constraints. Figures in the left and right show 3D and 2D errors, respectively.

PCA, and MFA optimize in a low-dimensional space and are much more efficient. These three methods learn the models of the data offline, and therefore can scale to the size of the database. MFA is slightly slower than PPCA and PCA; this might be due to the more complex model that is used by MFA to represent the data.

9.5 User Study

The goal of our user study is: (i) to compare our system with a common existing software tool for creating and editing face expressions; and (ii) to ask users to identify any main strengths and weaknesses of our system that we may not be aware of. We compared our system with "Poser 7". This software tool is used by animators and artists for animating, modeling, and rendering human-like figures. For the purpose of face modeling, this software allows the user to adjust face morphs to edit specific parts of the face. For example, one can set the value for "Smile" from 0.0 to 1.0 to create a smiling expression on the face.

For the user study, we first briefly show each user the functions of both Poser 7 and our Face Poser system. We then give the users examples of face expressions, and ask them to create these expressions (starting from a neutral expression) using both systems. These examples are in the form of images or photos of faces. Figure 22(a) shows some examples. These are meant to serve as a guideline for the users. We decided to give these images or photos as guidelines instead of specific descriptions such as "happy" or "surpise", since different people may have different interpretations of these terms. We tested fifteen users, and each user has either little or no previous experience with 3D modeling. We asked each user to create these expressions: joy, angry, digust, fear, and surprise. Figure 22(b) and (c) show examples of models created by users. After creating each expression with each system, we asked the user to provide a score (from 1 to 9) of how easy it is to create that expression with that system. We also recorded the time it took the user to create each expression.

The results show that the 3D models created by the users with either Poser 7 or Face Poser are similar. This can be seen in

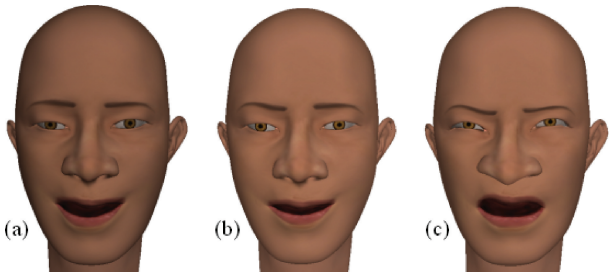


Fig. 17. A side-by-side comparison of cross-validation results for point constraints. (a) Ground truth. (b) MFA. (c) LWR.

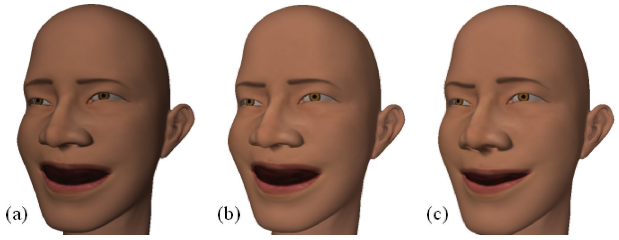


Fig. 18. A side-by-side comparison of cross-validation results for point constraints. (a) Ground truth. (b) MFA. (c) PCA.

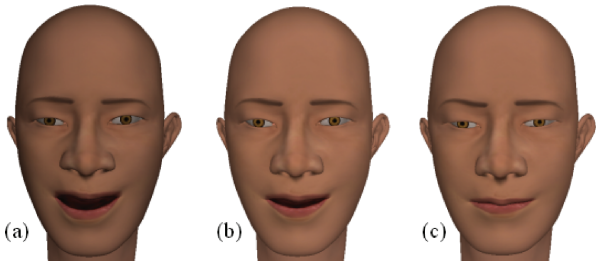


Fig. 19. A side-by-side comparison of cross-validation results for stroke constraints. (a) Ground truth. (b) MFA. (c) LWR.

Fig. 20. A side-by-side comparison of cross-validation results for stroke constraints. (a) Ground truth. (b) MFA. (c) PCA.

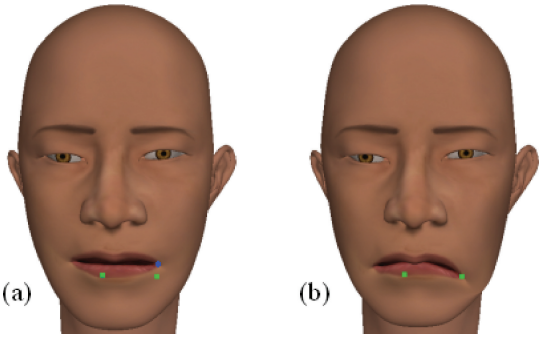


Fig. 21. A side-by-side comparison of results for MFA and PCA. The same inputs are used in both cases. (a) MFA has a larger 2D error, but the solution is natural. (b) PCA has a zero 2D error, but the result is not natural.

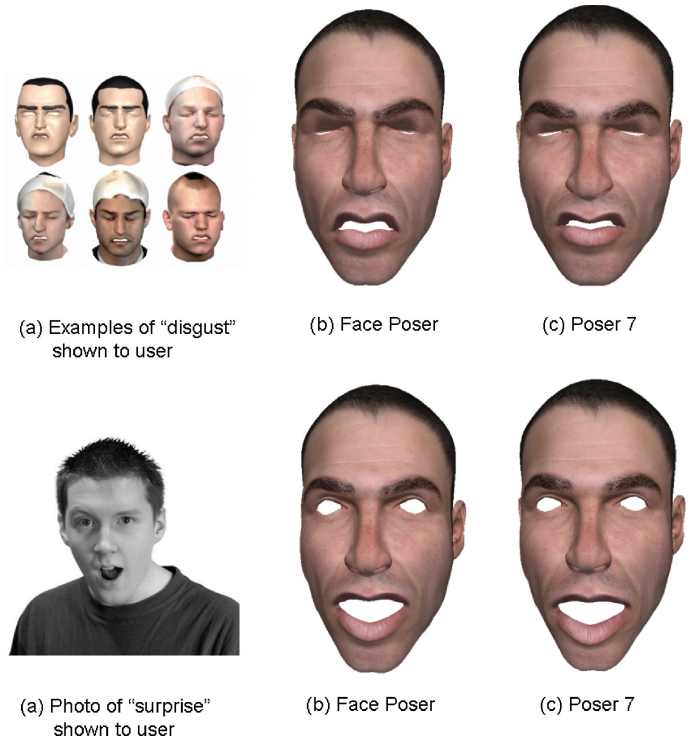


Fig. 22. Example inputs and results from the user study. The top row shows the case for "disgust" expressions, and the bottom row shows the case for "surprise" expressions. Parts (a) show the images or photo shown to the users. Parts (b) and (c) show the 3D models created by users with Face Poser and Poser 7, respectively.

Table II. Comparison of Runtime for Different Techniques and Different Constraints

	Time (ms) for Each Constraint		
	Point	Distance	Stroke
Opt-blend	73,741	72,526	142,836
Interp-all	63,539	68,648	78,187
PPCA	59	84	92
PCA	58	78	104
LWR	30,086	32,057	51,797
MFA	114	147	159

Figure 22(b) and (c). The "easiness score" and "time to create each expression" are averaged over seventy-five cases (fifteen users and five expressions each). Figure 23(left) shows the average easiness score provided by the users for each system. For Poser 7, the average score is 5.6 and the standard deviation is 1.6. For Face Poser, the average score is 7.3 and the standard deviation is 1.3. Figure 23(right) shows the average time to create each expression for each system. For Poser 7, the average time is 4.0 minutes and the standard deviation is 1.2. For Face Poser, the average score is 2.4 minutes and the standard deviation is 1.2. At the end of each user session, we asked each user to provide qualitative comments about the strengths and weaknesses of each system. The following is a summary of these comments.

—*Poser 7*. If the user knows the morph parameters well, this system is useful. However, some morph parameters are not intuitive. It is difficult to know what values to set for some of the morphs, and the editing process is sometimes based on a series of trial-and-error. There are many morph parameters: the advantage is that it is possible to make many types of changes, while the disadvantage is that it might be confusing to decide which sets of morphs should be used.

—*Face Poser*. This method takes less time to create or edit a face expression. It is nice that there are no menus, and one only needs to sketch "on the 3D model." It is sometimes difficult to make changes to some specific parts of the face. It is good that we do not have to know what the morphs do in advance.

Our results show that if an animator already has expertise with 3D face modeling and is trying to edit face expressions on a specific 3D model, then the Poser 7 system would be a good choice. The reason is that the morphs can be defined in advance for that specific 3D model, and the animator can spend time to adjust the morphs until the desired deformation is achieved. However, if a user is someone who has little or no experience with 3D modeling and he/she needs to edit face expressions on one or more models that is unfamiliar to him/her, then our Face Poser system would be a good choice. The reason is that the sketching interface is fast and requires almost no learning time. After just one demonstration of the sketching interface, users can immediately sketch new expressions on their own.

10. DISCUSSION

We have presented an approach for generating facial models from different kinds of user constraints (point, distance, stroke, curve, fixed) while matching the statistical properties of a database of

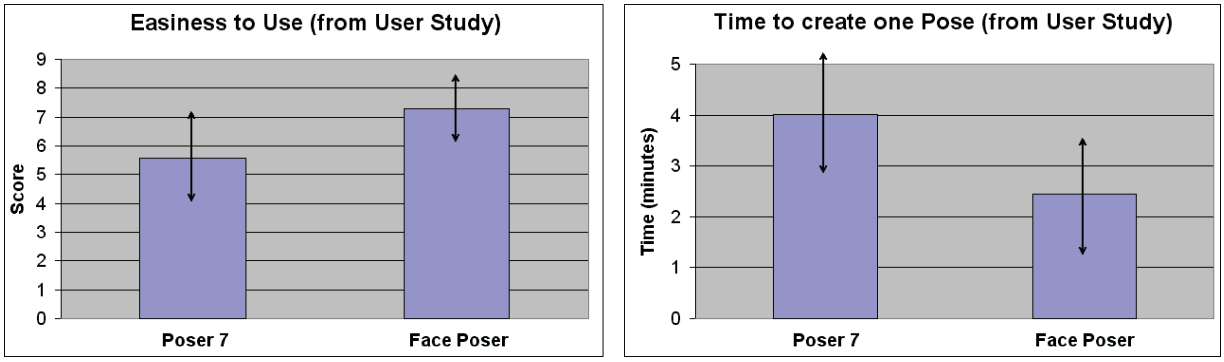


Fig. 23. Comparison with Poser 7. (Left) We asked the users to give a score (1-9) of how easy it is to create each expression with each system. This graph shows the results of these scores. (Right) We recorded the time it takes the users to create each expression with each system. This graph shows the timing results.

example models. The system first automatically learns a statistical model from example data and then enforces this as a prior to generate/edit the model. The facial prior, together with user-defined constraints, comprise a problem of maximum a posteriori estimation. Solving the MAP problem in a reduced subspace yields an optimal, natural face model that achieves the goal specified by the user.

The quality of the generated model depends on both facial priors and user-defined constraints. Without the use of the facial priors, the system would not generate natural facial expressions unless the user accurately specifies a very detailed set of constraints. One limitation of the approach, therefore, is that an appropriate database must be available. If a facial expression database does not include highly detailed facial geometry such as wrinkles, our system will not generate wrinkles on the face model.

The quality of the generated facial model also depends on the naturalness of the constraints. Constraints are "natural" when there exists at least one natural facial model consistent with them. The user might not create a natural facial expression if the constraints do not match any natural expression in the database or if the constraints are not consistent with each other.

The appearance of the final facial model is also influenced by the weight of the facial prior term, which provides a trade-off between the prior and the user-defined constraints. Instead of choosing a fixed weight, we allow the user to choose this weight dynamically; we can provide this capability because of the speed of the system.

The system allows for a "click done" mode and a "dragging" mode to create and edit a facial model. The user can choose the desired constraints and then click a button to generate the solution with the current constraints. This allows for placing multiple points and/or strokes in one optimization step. This can lead to large-scale changes, but all the constraints may not be satisfied if they come in conflict with allowing for natural poses. The "dragging" mode provides a manipulation interface where the user can see the changes continuously. It allows for more detailed changes over the local region of the dragged point.

Our system allows the user to generate facial models from various types of user-defined constraints. Any kinematic constraints can be integrated into our statistical optimization framework as long as the constraints can be expressed as a function of 3D positions of vertices.

We tested our system with a keyboard/mouse interface and an electronic pen/tablet interface. The system is simple and intuitive, and appeals to both beginning and professional users. Our system greatly reduces the time needed for creating natural face models compared to existing 3D mesh editing software. The system can work with other types of input devices. For example, the user can specify the desired facial deformation by dragging multiple facial points on a large touch screen or tracking a small set of facial points using a vision-based interface.

A possible future extension is to model the face as separate regions, generate each region separately, and blend the regions back together. This might allow for fine-grained control over local geometry and improve the generalization ability of our model.

APPENDIX: JACOBIAN EVALUATION

In this section, we show the derivations of the Jacobian terms. Their corresponding optimization terms are shown in the "User Interaction" section.

The Jacobian matrix $J_i(\mathbf{p})$ can be evaluated as follows. We have

$$\begin{aligned} J_i(\mathbf{p}) &= \frac{\partial \mathbf{y}_i}{\partial \mathbf{p}}, \\ &= \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_i} \cdot \frac{\partial \mathbf{x}_i}{\partial \mathbf{p}}, \end{aligned} \tag{22}$$

where the first Jacobian term can be computed as

$$\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_i} = \begin{pmatrix} \frac{s_w}{2} \cdot \frac{f\mathbf{r}_1^T \cdot (\mathbf{r}_3^T \mathbf{x}_i + t_3) - \mathbf{r}_3^T \cdot (f\mathbf{r}_1^T \mathbf{x}_i + ft_1)}{(\mathbf{r}_3^T \mathbf{x}_i + t_3)^2} \\ -\frac{s_h}{2} \cdot \frac{f\mathbf{r}_2^T \cdot (\mathbf{r}_3^T \mathbf{x}_i + t_3) - \mathbf{r}_3^T \cdot (f\mathbf{r}_2^T \mathbf{x}_i + ft_2)}{(\mathbf{r}_3^T \mathbf{x}_i + t_3)^2} \end{pmatrix} \tag{23}$$

and the second Jacobian term is

$$\frac{\partial \mathbf{x}_i}{\partial \mathbf{p}} = W_i \cdot B. \tag{24}$$

The Jacobian matrix for E_{point} is

$$\begin{aligned} \frac{\partial E_{point}}{\partial \mathbf{p}} &= \frac{\partial (\|\mathbf{g}_i(\mathbf{p}) - \mathbf{z}_i\|^2)}{\partial \mathbf{p}} \\ &= \frac{\partial \|\mathbf{y}_i - \mathbf{z}_i\|^2}{\partial \mathbf{y}_i} \cdot \frac{\partial \mathbf{y}_i}{\partial \mathbf{p}} \\ &= 2(\mathbf{y}_i - \mathbf{z}_i)^T \cdot J_i(\mathbf{p}). \end{aligned} \tag{25}$$

The Jacobian matrix for $E_{distance}$ can be computed as follows.

$$\begin{aligned}
&\frac{\partial E_{distance}}{\partial \mathbf{p}} \\
&= \frac{\partial(\|\mathbf{y}_i-\mathbf{y}_j\|-d)^2}{\partial \mathbf{p}} \\
&= 2(\|\mathbf{y}_i-\mathbf{y}_j\|-d)\cdot\frac{1}{2\sqrt{(\mathbf{y}_i-\mathbf{y}_j)^T(\mathbf{y}_i-\mathbf{y}_j)}}\cdot\frac{\partial(\mathbf{y}_i-\mathbf{y}_j)^T(\mathbf{y}_i-\mathbf{y}_j)}{\partial \mathbf{p}} \\
&= \frac{(\|\mathbf{y}_i-\mathbf{y}_j\|-d)}{\sqrt{(\mathbf{y}_i-\mathbf{y}_j)^T(\mathbf{y}_i-\mathbf{y}_j)}}\cdot\left(2(\mathbf{y}_i-\mathbf{y}_j)^T\cdot\frac{\partial(\mathbf{y}_i-\mathbf{y}_j)}{\partial \mathbf{p}}\right) \\
&= \frac{2(\|\mathbf{y}_i-\mathbf{y}_j\|-d)(\mathbf{y}_i-\mathbf{y}_j)^T}{\sqrt{(\mathbf{y}_i-\mathbf{y}_j)^T(\mathbf{y}_i-\mathbf{y}_j)}}\cdot(J_i(\mathbf{p})-J_j(\mathbf{p}))
\end{aligned} \tag{26}$$

The Jacobian matrix for E_{stroke} can be evaluated as

$$\frac{\partial E_{stroke}}{\partial \mathbf{p}} = 2\cdot\sum_{r\in R}\left[\frac{(I_y(\mathbf{y}_i)-I_z(r))}{\sigma^2_{stroke}}\cdot\left(\frac{\partial I_y(\mathbf{y}_i)}{\partial \mathbf{y}_i}\cdot\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_s}\cdot\frac{\partial \mathbf{x}_s}{\partial \mathbf{p}}\right)\right], \tag{27}$$

where the partial derivative $\frac{\partial I_y(\mathbf{y}_i)}{\partial \mathbf{y}_i}$ is the image gradient computed by the Sobel operator [Duda and Hart 1973]. We use Eq. (23) to compute the partial derivative $\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_s}$.

Finally based on Eq. (11), $\frac{\partial \mathbf{x}_s}{\partial \mathbf{p}}$ can be computed as

$$\frac{\partial \mathbf{x}_s}{\partial \mathbf{p}} = u\cdot\frac{\partial \mathbf{x}_u}{\partial \mathbf{p}} + v\cdot\frac{\partial \mathbf{x}_v}{\partial \mathbf{p}} + w\cdot\frac{\partial \mathbf{x}_w}{\partial \mathbf{p}}, \tag{28}$$

where the partial derivatives on the right side of this equation can be substituted with Eq. (24).

The Jacobian matrix for E_{extra} is

$$\begin{aligned}
\frac{\partial E_{extra}}{\partial \mathbf{p}} &= \frac{2}{\sigma^2_{extra}}\cdot\left(\frac{\sum \mathbf{y}_i}{N_s}-\overline{\mathbf{z}}\right)^T\cdot\frac{\partial\left(\frac{\sum \mathbf{y}_i}{N_s}\right)}{\partial \mathbf{p}} \\
&= \frac{2}{\sigma^2_{extra}N_s}\cdot\left(\frac{\sum \mathbf{y}_i}{N_s}-\overline{\mathbf{z}}\right)^T\cdot\sum\left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_s}\cdot\frac{\partial \mathbf{x}_s}{\partial \mathbf{p}}\right),
\end{aligned} \tag{29}$$

where $\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_s}$ can be evaluated with Eq. (23) and $\frac{\partial \mathbf{x}_s}{\partial \mathbf{p}}$ can be evaluated with Eq. (28).

The Jacobian matrix for E_{fixed} is

$$\frac{\partial E_{fixed}}{\partial \mathbf{p}} = \frac{(\mathbf{x}_i-\mathbf{x}_{i\_original})^T}{\sigma^2_{fixed}}\cdot W_i B. \tag{30}$$

The Jacobian matrix of the prior term E_{prior} can be computed as follows.

$$\frac{\partial E_{prior}}{\partial \mathbf{p}} = \sum_k \frac{\pi_k\mathcal{N}\left(\mu_k, A_kA_k^T+\Psi_k\right)(\mathbf{p}-\mu_k)^T\left(A_kA_k^T+\Psi_k\right)^{-1}}{\sum_{k=1}^K \pi_k\mathcal{N}\left(\mu_k, A_kA_k^T+\Psi_k\right)} \tag{31}$$

REFERENCES

BISHOP, C. M. AND WINN, J. M. 2000. Non-Linear bayesian image modeling. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 3–17.

Bishop, C. 1996. *Neural Network for Pattern Recognition*. Cambridge University Press.

BLANZ, V. AND VETTER, T. 1999. A morphable model for the synthesis of 3d faces. In *Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*. 187–194.

BLANZ, V., BASSO, C., POGGIO, T., AND VETTER, T. 2003. Reanimating faces in images and video. *Comput. Graph. Forum. 22*, 3, 641–650.

BRAND, M. 2002. Incremental singular value decomposition of uncertain data with missing values. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 707–720.

CHAI, J. AND HODGINS, J. 2005. Performance animation from low-dimensional control signals. *ACM Trans. Graph. 24*, 3, 686–696.

CHAI, J., XIAO, J., AND HODGINS, J. 2003. Vision-Based control of 3D facial animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 193–206.

CHANG, E. AND JENKINS, O. C. 2006. Sketching articulation and pose for facial animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 271–280.

DER, K. G., SUMNER, R. W., AND POPOVIC, J. 2006. Inverse kinematics for reduced deformable models. *ACM Trans. Graph. 25*, 3, 1174–1179.

DUDA, R. AND HART, P. 1973. *Pattern Classification and Scene Analysis*. John Wiley and Sons.

FENG, W.-W., KIM, B.-U., AND YU, Y. 2008. Real-Time data driven deformation using kernel canonical correlation analysis. *ACM Trans. Graph. 27*, 3, Article no. 91.

GALASSI, M., DAVIES, J., THEILER, J., GOUGH, B., JUNGMAN, G., BOOTH, M., AND ROSSI, F. 2003. *GNU Scientific Library Reference Manual, Revised 2nd Ed.* Network Theory Ltd.

GHAHRAMANI, Z. AND HINTON, G. E. 1997. The EM algorithm for mixtures of factor analyzers. Tech. rep. CRG-TR-96-1.

GROCHOW, K., MARTIN, S. L., HERTZMANN, A., AND POPOVIC, Z. 2004. Style-Based inverse kinematics. *ACM Trans. Graph. 23*, 3, 522–531.

HINTON, G. E., DAYAN, P., AND REVOW, M. 1997. Modeling the manifolds of images of handwritten digits. *IEEE Trans. Neural Netw. 8*, 1, 65–74.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3D freeform design. In *Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*. 409–416.

JOSHI, P., TIEN, W. C., DESBRUN, M., AND PIGHIN, F. 2003. Learning controls for blendshape based realistic facial animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Aimation*. 187–192.

KHO, Y. AND GARLAND, M. 2005. Sketching mesh deformations. In *Proceedings of the ACM Symposium on Interactive 3D Graphics and Games*. 147–154.

LAU, M., CHAI, J.-X., XU, Y.-Q., AND SHUM, H.-Y. 2007. Face poser: Interactive modeling of 3D facial expressions using model priors. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. 161–170.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*. 165–172.

MEYER, M. AND ANDERSON, J. 2007. Key point subspace acceleration and soft caching. *ACM Trans. Graph. 26*, 3, Article no. 74.

NEALEN, A., SORKINE, O., ALEXA, M., AND COHEN-OR, D. 2005. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph. 24*, 3, 1142–1147.

PARKE, F. I. 1972. Computer generated animation of faces. In *Proceedings of the ACM National Conference*. 1, 451–457.

PIGHIN, F., SZELISKI, R., AND SALESIN, D. 1999. Resynthesizing facial animation through 3D model-based tracking. In *Proceedings of the International Conference on Computer Vision*. 143–150.

SLOAN, P.-P., ROSE, C., AND COHEN, M. F. 2001. Shape by example. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*. 135–143.

SUMNER, R. W., ZWICKER, M., GOTSMAN, C., AND POPOVIC, J. 2005. Mesh-Based inverse kinematics. *ACM Trans. Graph. 24*, 3, 488–495.

VICON SYSTEMS. 2007. http://www.vicon.com.

YANG, C., SHARON, D., AND VAN DE PANNE, M. 2005. Sketch-Based modeling of parameterized objects. In *Proceedings of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling*.

ZELEZNIK, R. C., HERNDON, K. P., AND HUGHES, J. F. 1996. Sketch: An interface for sketching 3D scenes. In *Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*. 163–170.

ZHANG, Q., LIU, Z., GUO, B., AND SHUM, H. 2003. Geometry-Driven photorealistic facial expression synthesis. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer An- imation*. 177–186.

ZHANG, L., SNAVELY, N., CURLESS, B., AND SEITZ, S. M. 2004. Spacetime faces: High resolution capture for modeling and animation. *ACM Trans. Graph. 23*, 3, 548–558.

Received January 2008; revised April 2009; accepted August 2009