Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2007)
D. Metaxas and J. Popovic (Editors)

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

Manfred Lau[1,3] Jinxiang Chai[2] Ying-Qing Xu[3] Heung-Yeung Shum[3]

[1]Carnegie Mellon University [2]Texas A&M University [3]Microsoft Research Asia

**Abstract**

*In this paper, we present an intuitive interface for interactively posing 3D facial expressions. The user can create and edit facial expressions by drawing freeform strokes, or by directly dragging facial points in 2D screen space. Designing such an interface for face modeling and editing is challenging because many unnatural facial expressions might be consistent with the ambiguous user input. The system automatically learns a model prior from a prerecorded facial expression database and uses it to remove the ambiguity. We formulate the problem in a maximum a posteriori (MAP) framework by combining the prior with user-defined constraints. Maximizing the posterior allows us to generate an optimal and natural facial expression that satisfies the user-defined constraints. Our system is interactive; it is also simple and easy to use. A first-time user can learn to use the system and start creating a variety of natural face models within minutes. We evaluate the performance of our approach with cross validation tests, and by comparing with alternative techniques.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling-Modeling packages; I.3.6 [Computer Graphics]: Methodology and Techniques-Interaction techniques

## 1. Introduction

A long-standing challenge in computer graphics is to build an interactive system that allows users 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 paper, we present an intuitive and efficient interface for modeling 3D facial expressions (Figure 1). Our system starts with a 3D mesh model in a neutral expression. For the sketch-based interface, 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 on the 2D screen space from any camera view. The system deforms the facial model based on these inputs. The user can continue to refine the model until a desired 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 inputs are a small set of user-defined freeform strokes or control points. This information is quite low-dimensional as compared to a typical facial model, which is commonly represented by at least thousands of DOFs [PW96]. The user's inputs, therefore, cannot be used to fully determine a natural facial configuration

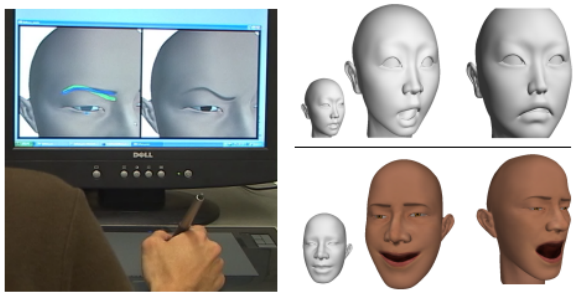

**Figure 1:** *(left) The Face Poser system in use with a graphics tablet. (right) A variety of face expressions created using the system. The lower left corner shows the neutral expression for each subject.*

Copyright © 2007 by the Association for Computing Machinery, Inc.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions Dept, ACM Inc., fax +1 (212) 869-0481 or e-mail permissions@acm.org.
SCA 2007, San Diego, California, August 04 - 05, 2007
© 2007 ACM 978-1-59593-624-4/07/0008 $ 5.00

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

We formulate the face expression modeling problem in a probabilistic framework by combining user-defined constraints with a prior from the database. We model the prior as a mixture of factor analyzers [GH97] and learn it automatically from a pre-recorded facial expression database. Maximizing the posterior (likelihood of facial expressions give user-specified constraints) generates a natural and most desirable 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 and stroke constraints. The constraints are specified on 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 naive user. We have found that a first-time user can learn to use the system in a few minutes, and be able to create desired facial expressions within tens of seconds. Figure 1 shows some examples of facial expressions generated by our system.

We evaluate the quality of synthesized facial expressions by comparison against ground truth data. We also compare alternative data-driven techniques for generating facial expressions from user-defined constraints. We compare our method against blendshape interpolation, optimization in the PCA subspace, and optimization with blendshapes.

## 2. Background

In the next section, we discuss related work in sketch-based interfaces for object modeling. Because we use prerecorded facial data in our system, we then review research utilizing examples for modeling.

### 2.1. Sketch-based interfaces

Our work is inspired by sketch-based systems that interpret the user's strokes for constructing and editing 3D models [ZHH96, IMT99]. Zeleznik and his colleagues [ZHH96] introduced a sketch-based interface to create and edit rectilinear objects. Igarashi and his colleagues [IMT99] developed the first sketch interface to interactively model and edit free-form objects. Recently, a number of researchers have explored sketch-based interfaces to edit mesh deformations [NSACO05, KG05]. Nealen and his colleagues [NSACO05] presented a sketch-based interface for laplacian mesh editing where a user draws reference and target curves on the mesh to specify the mesh deformation. Kho and Garland [KG05] demonstrated a similar interface for posing the bodies and limbs of 3D characters. Furthermore, Yang and his colleagues [YSvdP05] 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. Our goal is to develop an interface for interactively modeling and editing 3D facial expressions.

Chang and Jenkins [CJ06] presented a similar interface to edit a facial model. Their interface is different in that it works only for strokes drawn as lines on the screen. Our stroke interface allows for drawing any line, curve, shape, or region on the screen, and in any camera view. More importantly, our approach learns model priors from a prerecorded database and uses them to remove the modeling ambiguity.

### 2.2. Example-based modeling

Our work builds upon previous example-based modeling systems. Many example-based modeling systems are based on estimating weights to linearly combine example models [Par72, LCF00, SRC01] or eigen-models [BV99, BBPV03]. These weights can be computed from either user-specified constraints or images.

Zhang and his colleagues [ZSCS04] developed an example-based FaceIK system to edit a facial model by interactively dragging points on the face. Their face model is represented as a linear combination of pre-acquired 3D face scans. Researchers [JTDP03, ZLGS03] have also proposed to segment a model into multiple regions and represent each subregion as a convex linear combination of blend shapes. Besides facial modeling, example-based approach has also been applied to edit a skeleton structure. Sumner and Popovic [SZGP05] and Der and his colleagues [DSP06] 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.

An alternative way to estimate the weights of examples is to reconstruct them directly from images or video [BV99, PSS99, BBPV03]. Blanz and his colleagues [BV99, BBPV03] built a morphable model from 3D scans via Principal Component Analysis (PCA) [Bis96] and applied the morphable model to reconstruct a 3D model from a single image. Pighin and his colleagues [PSS99] demonstrated that they can estimate the weights of a small set of 3D morphed face models directly from images or video. Chai and his colleagues presented a real-time vision-based performance animation system which transforms a small set of automatically tracked facial features into facial animation by interpolating closest examples in a database at runtime [CXH03].

The main difference between our work and previous example-based modeling systems is that we learn a probabilistic model to represent the prior from example data and formulate the problem in a probabilistic framework by combining user-defined constraints with the prior. The system automatically learns a mixture of factor analyzers [GH97]

© Association for Computing Machinery, Inc. 2007.

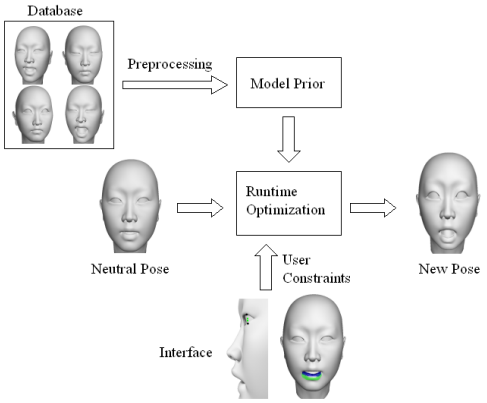


**Figure 2:** *System overview.*

from a facial database. With a collection of locally linear sub-models, our mixture of factor analyzers can model a nonlinear structure which cannot be modeled by existing linear models such as blendshapes or eigen-shapes.

A number of researchers have also developed statistical models for human poses and used them to estimate poses from kinematic constraints to solve the inverse kinematics problem. For example, Grochow and his colleagues [GMHP04] applied a global nonlinear dimensionality reduction technique, a Gaussian Process Latent Variable Model [Law04], to human motion data and then 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, it might not be appropriate for our application because the performance of the GPLVM deteriorates 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). Chai and his colleagues used a series of local statistical pose models constructed at runtime to reconstruct full body motion from continuous, low-dimensional control signals obtained from video cameras [CH05]. Online local models are more appropriate for creating animations (a sequence of poses) from continuous temporal constraints. They are not appropriate for our application because user-defined constraints are usually not continuous.

## 3. Overview

The main idea of our approach is that model priors learned from pre-recorded facial expression data can be used to create natural facial expressions that match constraints specified by the user. The combination of the model priors and the user-defined constraints provide sufficient information to produce 3D facial expression models with a natural appearance. Figure 2 shows an overview of our system.

© Association for Computing Machinery, Inc. 2007.

### 3.1. Data Preprocessing

We set up a Vicon motion capture system to record facial movement by attaching 55 reflective markers onto the face of the motion capture subject. We captured the subject performing various sets of facial actions. The data contains basic facial expressions such as anger, fear, surprise, sadness, joy, and disgust, and 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 [CXH03]. 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 example in the database as $\mathbf{x} \in R^d$, where $\mathbf{x}$ is the stacked 3D positions of all vertices in the face model and $d$ is three times the number of vertices in a facial model. Let $M$ be the total number of samples for each subject. We use Principal Component Analysis (PCA) [Bis96] to preprocess the captured data; we then obtain a reduced subspace representation for $\mathbf{x}$:

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

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 $\bar{\mathbf{x}}$ 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 [Bra02].

### 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 model $\mathbf{p}$ given the user-defined constraints $\mathbf{c}$:

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

where $pr(\mathbf{c})$ is the normalizing constant that ensures that the posterior 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 optimization problem for model $\hat{\mathbf{p}}$:

$$\hat{\mathbf{p}} = \arg\min_{\mathbf{p}} \; -\ln pr(\mathbf{c}|\mathbf{p}) - \ln pr(\mathbf{p}) \tag{3}$$

where the first term measures how well a face model matches the user-specified constraints and the second term measures the *a priori* likelihood of the face model using the knowledge embedded in the example data.

### 3.3. System Components

The system contains three major components (see figure 2):

**User Constraints.** The system starts with a default facial model. The user interactively edits the facial model using two kinds of constraints: point constraints and stroke constraints. These constraints are specified on the 2D screen space. The interface allows the user to iteratively apply these constraints until a desired solution is obtained.

**Model Priors.** The system automatically learns a statistical model from the captured data in the reduced PCA subspace. This model is used to constrain the generated facial expression to lie in the space of natural facial expressions.

**Runtime Optimization.** The system uses nonlinear optimization to automatically find a facial model that best satisfies the user-specified constraints while matching the statistical properties of the captured data.

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

## 4. User Constraints

Our system starts with a default 3D facial model. In our implementation, we start with the mean pose $\overline{\mathbf{x}}$. The user can specify point constraints and stroke constraints to interactively edit facial expressions. For point constraints, the user can select any facial points and specify their desired positions on the screen space. For stroke constraints, the user selects a reference freeform stroke and then draws a target freeform stroke to specify a desired deformation for the reference stroke. This section focuses on deriving objective functions and their derivatives for both constraints.

We first derive the relationship between a facial model in the PCA subspace ($\mathbf{p}$) and the projection of a chosen vertex on the 2D screen space ($\mathbf{y}_i$). Let $\mathbf{x}_i$ denote the 3D coordinate of the $i^{th}$ vertex. Let $\mathbf{y}_i$ denote the 2D projection of the $i^{th}$ vertex on the screen space under the current camera viewpoint. 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 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 current width and height of the 2D screen respectively. We further have

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

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 zero except:

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

Combining equations 1, 5, and 6 together, the 2D projection $\mathbf{y}_i$ of a 3D facial vertex can be represented as a nonlinear function of weights $\mathbf{p}$:

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

The Jacobian matrix can be computed as follows:

$$\begin{aligned} J_i(\mathbf{p}) &= \frac{\partial \mathbf{y}_i}{\partial \mathbf{p}} \\ &= \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_i} \cdot \frac{\partial \mathbf{x}_i}{\partial \mathbf{p}} \end{aligned} \tag{9}$$

The first Jacobian term can be computed as follows:

$$\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_i} = \begin{pmatrix} \frac{s_w}{2}\cdot\frac{f\mathbf{r}_1^T\cdot(\mathbf{r}_3^T\mathbf{x}_i+t_3)-\mathbf{r}_3^T\cdot(f\mathbf{r}_1^T\mathbf{x}_i+ft_1)}{(\mathbf{r}_3^T\mathbf{x}_i+t_3)^2} \\ -\frac{s_h}{2}\cdot\frac{f\mathbf{r}_2^T\cdot(\mathbf{r}_3^T\mathbf{x}_i+t_3)-\mathbf{r}_3^T\cdot(f\mathbf{r}_2^T\mathbf{x}_i+ft_2)}{(\mathbf{r}_3^T\mathbf{x}_i+t_3)^2} \end{pmatrix} \tag{10}$$

The second Jacobian term is

$$\frac{\partial \mathbf{x}_i}{\partial \mathbf{p}} = W_i \cdot B \tag{11}$$

### 4.1. Point Constraints

The first type of user-defined constraint allows the user to change the position of individual vertices on the mesh. This allows the user to have detailed control over the final result. The user first selects a set of 3D source vertices $\{\mathbf{x}_i | i = 1,...,N\}$ and then specifies a corresponding set of 2D target pixels $\{\mathbf{z}_i | i = 1,...,N\}$ of where the vertices should map to on the screen space. The user selects each 3D point by selecting a 2D screen pixel. We perform ray tracing 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. The optimization term for point constraints can be described as follows:

$$E_{point} = \frac{1}{N}\cdot\sum_i \|\mathbf{g}_i(\mathbf{p}) - \mathbf{z}_i\|^2 \tag{12}$$

The Jacobian matrix is

$$\begin{aligned} \frac{\partial E_{point}}{\partial \mathbf{p}} &= \frac{1}{N}\cdot\frac{\partial(\sum_i \|\mathbf{g}_i(\mathbf{p})-\mathbf{z}_i\|^2)}{\partial \mathbf{p}} \\ &= \frac{1}{N}\cdot\frac{\partial\sum_i \|\mathbf{y}_i-\mathbf{z}_i\|^2}{\partial \mathbf{y}_i}\cdot\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_i}\cdot\frac{\partial \mathbf{x}_i}{\partial \mathbf{p}} \\ &= \frac{1}{N}\cdot\sum_i [2(\mathbf{y}_i-\mathbf{z}_i)^T\cdot\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_i}\cdot\frac{\partial \mathbf{x}_i}{\partial \mathbf{p}}] \end{aligned} \tag{13}$$

The final Jacobian matrix is computed by combining the above equation with Equations 10 and 11.

© Association for Computing Machinery, Inc. 2007.

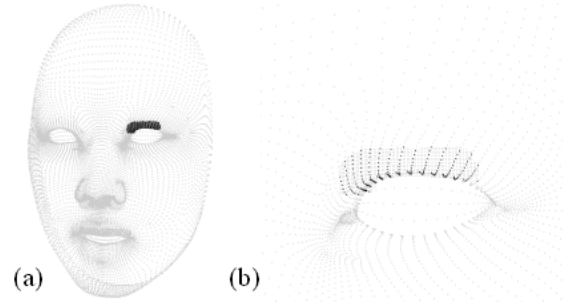


**Figure 3:** *(a) The dark gray points are 3D points on the mesh surface. They are selected by the source stroke for Stroke Constraints. (b) Closeup view. We see the individual dark gray points. They do not have to be the original vertices that make up the mesh. The black ones are also selected, and these are the original vertices in the mesh.*

### 4.2. 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 input. More specifically, the user first draws a source stroke to select a set of 3D points ($\mathbf{x}_s$'s) on the mesh (figure 3). Then the user draws a target stroke to provide a region of pixels ($\mathbf{z}_j$'s) where the 3D points should project to. These strokes can be in the form of a line, curve, or any freeform region. Unlike point constraints, the stroke constraints do not provide corresponding information between the source and target strokes.

Given a source stroke in the 2D screen space, we need to find the corresponding 3D points on the mesh efficiently. We ray traced 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 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 (figure 3). 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:

$$\mathbf{x}_s = u \cdot \mathbf{x}_u + v \cdot \mathbf{x}_v + w \cdot \mathbf{x}_w \tag{14}$$

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.

This set of 3D points on the mesh, $\mathbf{x}_s$'s, are projected back onto 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 in our experiment. 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. We measure the distance between the two strokes by computing the intensity difference between them. The optimization term for stroke constraints is

$$E_{stroke} = \sum_{r \in R} (I_y(\mathbf{y}_i) - I_z(r))^2 \tag{15}$$

We can compute its Jacobian matrix as follows:

$$\frac{\partial E_{stroke}}{\partial \mathbf{p}} = 2 \cdot \sum_{r \in R} [(I_y(\mathbf{y}_i) - I_z(r)) \cdot (\frac{\partial I_y(\mathbf{y}_i)}{\partial \mathbf{y}_i} \cdot \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_s} \cdot \frac{\partial \mathbf{x}_s}{\partial \mathbf{p}})] \tag{16}$$

The projected region of pixels, $\mathbf{y}_i$'s, are determined by forward projection of the current 3D points, $\mathbf{x}_s$'s. There might be multiple points projected to the same pixel. We choose to render the pixel using the 3D point that is closest to the camera. The partial derivative $\frac{\partial I_y(\mathbf{y}_i)}{\partial \mathbf{y}_i}$ is the image gradient computed by the Sobel operator [DH73]. We use Equation 10 to compute the partial derivative $\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_s}$. Finally based on Equation 14, $\frac{\partial \mathbf{x}_s}{\partial \mathbf{p}}$ can be computed as follows:

$$\frac{\partial \mathbf{x}_s}{\partial \mathbf{p}} = u \cdot \frac{\partial \mathbf{x}_u}{\partial \mathbf{p}} + v \cdot \frac{\partial \mathbf{x}_v}{\partial \mathbf{p}} + w \cdot \frac{\partial \mathbf{x}_w}{\partial \mathbf{p}} \tag{17}$$

where the partial derivatives on the right side of this equation can be substituted with Equation 11.

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 Equation 15 does not require the strokes to "move toward" each other. The energy value in Equation 15 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 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:

$$E_{extra} = \| \frac{\sum \mathbf{y}_i}{N_s} - \bar{\mathbf{z}} \|^2 \tag{18}$$

where $N_s$ is the number of pixels in the source stroke, $\bar{\mathbf{z}}$ is the center of the pixels in the target stroke, and the summation is for all source pixels $y_i$. The Jacobian matrix is

$$\begin{aligned} \frac{\partial E_{extra}}{\partial \mathbf{p}} &= 2 \cdot (\frac{\sum \mathbf{y}_i}{N_s} - \bar{\mathbf{z}})^T \cdot \frac{\partial (\frac{\sum \mathbf{y}_i}{N_s})}{\partial \mathbf{p}} \\ &= \frac{2}{N_s} \cdot (\frac{\sum \mathbf{y}_i}{N_s} - \bar{\mathbf{z}})^T \cdot \sum (\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_s} \cdot \frac{\partial \mathbf{x}_s}{\partial \mathbf{p}}) \end{aligned} \tag{19}$$

The weight for this additional term in the optimization

© Association for Computing Machinery, Inc. 2007.

changes depending on the objective value (Equation 18). As the distance between the centers of the two strokes decreases, this weight decreases. Intuitively, as the two strokes get closer to each other, we decrease the effect of this additional term.

## 5. Model 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 real models by imposing a prior on the generated model.

We model the prior as a mixture of factor analyzers [GH97] and learn it automatically from a pre-recorded facial expression database. The MFA model learns a probability density function (P.D.F) in the PCA subspace that provides a model 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 [HDR97] and images [BW00].

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 $\omega \sim \mathcal{N}(0,\Psi)$. $\Psi$ is a diagonal matrix. The generative model can be described as:

$$\mathbf{p} = A\tau + \omega + \mu \tag{20}$$

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

$$pr(\mathbf{p};\Theta) = \mathcal{N}(\mu, AA^T + \Psi) \tag{21}$$

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 $K$ FAs is given by:

$$pr(\mathbf{p};\Theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, A_k A_k^T + \Psi_k) \tag{22}$$

where $\pi_k$ is a mixing proportion ($\pi_k > 0$ and $\sum_{k=1}^{K} \pi_k = 1$). The system automatically learns the model parameters of the MFA model, $\Theta = \{\pi_k, \mu_k, A_k, \Psi_k | k = 1,...,K\}$, from example data via expectation maximization techniques [GH97].

We minimize the negative log of $pr(\mathbf{p})$, yielding the energy formulation:

$$E_{prior}(\mathbf{p}) = -\ln \sum_{k=1}^{K} \pi_i \mathcal{N}(\mu_k, A_k A_k^T + \Psi_k) \tag{23}$$

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.

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

**Table 1:** *Details of the data we used. $M$ 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 the number of dimensions for the latent variable $\tau_k$ in each factor of the MFA model.*

The Jacobian matrix can be computed as follows:

$$\frac{\partial E_{prior}}{\partial \mathbf{p}} = \sum_k \frac{\pi_k \mathcal{N}(\mu_k, A_k A_k^T + \Psi_k)(\mathbf{p}-\mu_k)^T (A_k A_k^T + \Psi_k)^{-1}}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, A_k A_k^T + \Psi_k)} \tag{24}$$

## 6. Runtime Optimization

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

The overall objective function is a weighted combination of the user-defined constraint terms (equations 12, 15, 18) and the model prior term (equation 23):

$$\arg\min_{\mathbf{p}\in R^r} E_{point} + \lambda_1 E_{stroke} + \lambda_2 E_{extra} + \lambda_3 E_{prior} \tag{25}$$

In our experiments, $\lambda_1$ is set to 1; we choose an adaptive weight for $\lambda_2$ as discussed in section 4.2; $\lambda_3$ is selected dynamically by the user (in our experiments, we used a value between 2 and 5). Similarly, the overall jacobian term is a weighted combination of the individual jacobian terms.

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" $\mathbf{p}$:

$$\arg\min_{\mathbf{p}\in\{\mu_1,\ldots,\mu_K\}} E_{point} + \lambda_1 E_{stroke} + \lambda_2 E_{extra} + \lambda_3 E_{prior} \tag{26}$$

We use the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization algorithm in the GSL library [GDT*03]. The solution converges rapidly due to a good starting point, a low-dimensional optimization space, and the symbolic evaluation of the jacobian terms.

## 7. Results

Table 1 shows the details of the two data sets. For the first subject, the 3D face model has 17,111 vertices and 34,168

© Association for Computing Machinery, Inc. 2007.

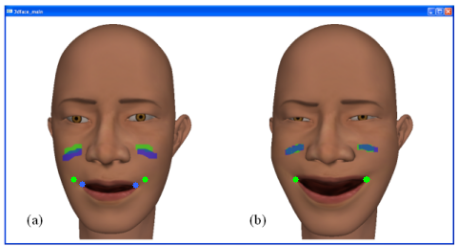


**Figure 4:** *(a) Original mesh: The user draws point and stroke constraints directly on the screen. The source point/stroke is in blue and target point/stroke is in green. (b) Result: The face model is deformed to satisfy the constraints.*

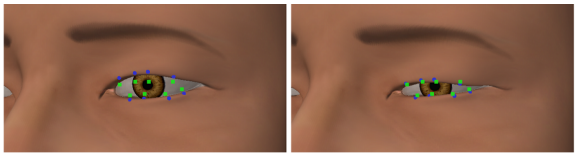

**Figure 5:** *The user iteratively adds as many constraints as desired. Starting from a neutral pose (left), the user creates a detailed eye geometry (right).*

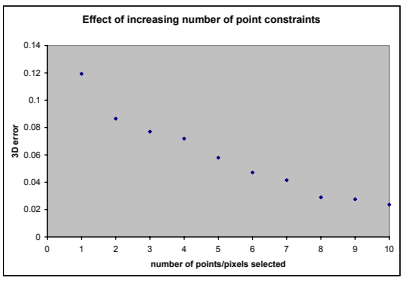


**Figure 6:** *Cross validation (section 7.3) results averaged over 5 trials.*

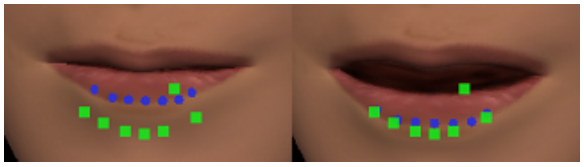

**Figure 7:** *If there are many constraints, some of them may conflict or "fight" with each other, since the system also tries to generate natural results. The inputs are shown on the left, and the synthesized result is on the right.*

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 percent of the energy in the eigenvalues.

The Face Poser system can generate a variety of facial expressions. A first-time user learns to use the system within minutes, and is able to create realistic expressions within seconds. Our users have reported that the interface is intuitive and easy to use. The accompanying video shows live screenshots of the results.

### 7.1. User-Defined Constraints

**Point Constraints** Figure 4 shows an example with point constraints, where the user poses the shape of the mouth in the smiling expression with two points. In general, this constraint is particularly useful for detailed control of individual points. We 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. In figure 5, the user iteratively adds more constraints to achieve the desired result of a more detailed eye geometry. As the number of point constraints increase, the results are more accurate (figure 6). However, if there are constraints that do not match *any* natural facial expressions represented by our statistical model, not all the constraints will be satisfied (figure 7).

**Stroke Constraints** This constraint can be used to efficiently make large-scale changes to the face model. In figure 4, we draw strokes to "raise" the cheek regions to create a smiling expression. The point constraints are used in this example to fine-tune the corners of the mouth. Our users have reported that the sketch interface allows us to draw certain features of the face such as an eyebrow (figure 1 left) more easily. Although some results might also be achieved with point constraints, such a sketch interface is much more intuitive. Some interesting results are shown in figure 8. We can see the effect of having the additional distance term for stroke constraints in figure 9. By adding this term to the optimization, the source stroke will "move towards" the target stroke, even if they are originally far away from each other.

The computational time of the system depends on the number of user-specified constraints. As the number of constraints increases, the computational time increases linearly. This is expected as the computation time for the image intensities increases proportionally with the number of pixels. For example, the computational time for stroke constraints with 900 pixels (about the size of a curve drawn on the eyebrow) is about 0.4 seconds. The time for point constraints also increases linearly with the number of points. It takes about 0.18 seconds for 10 points. The computational time of the system does not depend on the number of example data in the database because the MFA model is learned off-line.

### 7.2. The Importance of Model Prior

The model prior term allows the generated faces to be natural. Changing the weight of the term in the optimization provides a tradeoff between satisfying user constraints and creating natural faces. We can change this weight dynami-

© Association for Computing Machinery, Inc. 2007.

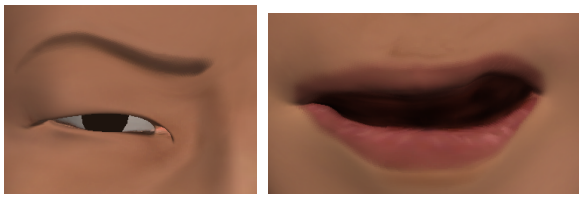

**Figure 8:** *We can create a raised eyebrow or a curled lip with stroke constraints.*

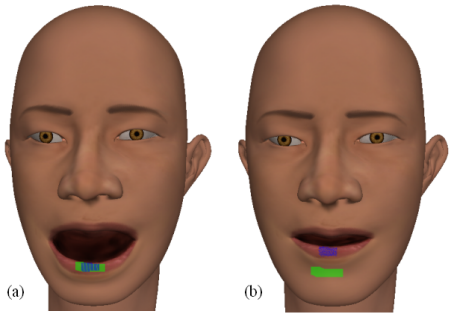


**Figure 9:** *With and without the additional distance term for stroke constraints. (a) With this term, the blue source stroke will match the green target stroke even if the two strokes are originally far away. (b) Without this term, the optimization gets stuck in a local minima.*

cally, thus providing an interesting spectrum of possibilities (figure 10) for the user to choose from.

### 7.3. Comparison Tests

We use new face models as testing data for cross validation tests. We start from a neutral pose and select the source constraints (any kind of constraints) as before. The corresponding target constraints are automatically generated based on the testing data. We then create the face models, and compare them against the original test data. In the results below, the 2D error is the value from equation 12. The 3D error is the average of the sum of the Euclidean distances between each vertex of the ground truth and synthesized 3D model.

| | Point Cons | |
|---|---|---|
| | interp | opt |
| 2D error | 18.15 | 2.28 |
| 3D error | 0.06704 | 0.03642 |

**Table 2:** *Cross validation results. Optimization produces more accurate results than interpolation. Each value is an average over 10 trials. We have similar results for other constraints.*

Table 2 shows a comparison of cross validation tests between the blend-shape interpolation and optimization methods. For interpolation, we compute a distance for each sample based on the objective functions in the optimization, and

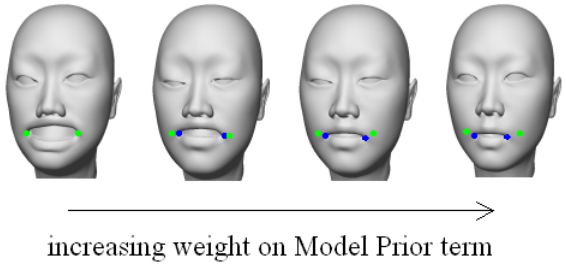


**Figure 10:** *As the weight increases, the expression becomes more natural. However, the constraints become less satisfied, and the 2D and 3D cross validation errors increase. The leftmost figure has a zero weight, and the rightmost figure has a high weight.*

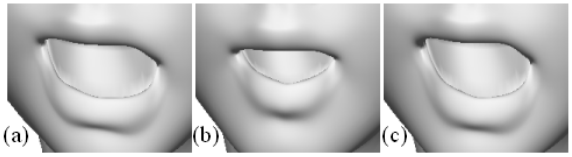


**Figure 11:** *Cross validation results. Optimization produces visually better results than interpolation. (a) Ground truth. (b) Interpolation, 2D error = 161.8, 3D error = 0.1921. (c) Optimization, 2D error = 10.5, 3D error = 0.0998.*

then use each distance to weigh (as in [AMS7a]) that sample accordingly. Figure 11 shows a visual comparison of the results.

We compare the MFA model used in this work against other methods (table 3). "Interp-all" takes all the samples, computes a weight on each sample based on equation 12, and blends the samples to find the solution [Par72, LCF00, SRC01]. "Opt-blend" represents the solution as a sum of weighted blendshapes and optimizes these weights [ZSCS04]. PCA performs the optimization in the PCA space [BV99, BBPV03]. LWR first finds $k$ examples in the database that best matches the user constraints; it then performs a locally weighted regression [AMS7a, CXH03] where each sample is weighted based on equation 12. All optimization methods start with the same initial pose. Figure 12 shows a visual side-by-side comparison of one result.

| | Interp-all | Opt-blend | PCA | LWR | MFA |
|---|---|---|---|---|---|
| 1 point | 0.197 | 0.160 | 0.157 | 0.155 | 0.060 |
| 4 points | 0.191 | 0.156 | 0.154 | 0.149 | 0.047 |
| time (ms) | 53753 | 26954 | 103 | 52133 | 183 |

**Table 3:** *A comparison of our MFA method with other techniques, for different numbers of point constraints. The values are the 3D errors described above; each one is an average over 3 trials.*

In table 3, "Interp-all" and "Opt-blend" do not produce

© Association for Computing Machinery, Inc. 2007.


**Figure 12:** *A visual comparison of cross validation results. (a) Ground truth. (b) Optimization with PCA. (c) Optimization with MFA.*

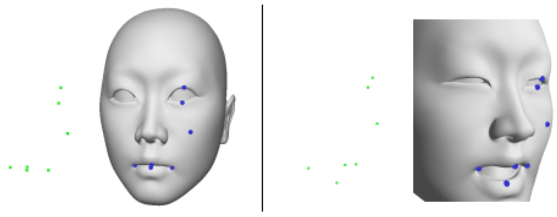

**Figure 13:** *Trajectory keyframing: We use seven 2D target points (green) to generate the 3D model (blue points are corresponding 3D source points). Two examples with different viewpoints are shown here. See the video for an animation.*

good results because the solution depends on all the samples. PCA also produces a large error because the number of constraints in our system is usually small and they are not sufficient to fully determine the weights of **p**. For LWR, the model provided by the closest samples might not be efficient because they do not provide any temporal coherence. "Interp-all" and LWR are slow since these methods have to iterate through every sample in the database. "Opt-blend" is also slow because there are many parameters in the optimization problem due to the large number of samples. PCA and MFA optimizes in a low-dimensional space and is therefore most efficient. Their models of the dataset are learned off-line. These results show that our technique produces a smaller error and is more efficient than previous methods.

### 7.4. Application: Trajectory Keyframing

Our optimization framework can be applied for trajectory keyframing. Given the trajectories of a few points, we can re-create the 3D face models using point constraints. Some examples are shown in figure 13. 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.

## 8. Discussion

We have presented an approach for generating facial models from two kinds of user constraints (point and stroke constraints) while matching the statistical properties of a database of 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 model 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 goals specified by the user.

The quality of the generated model depends on both model priors and user-defined constraints. Without the use of the model priors, the system would not generate natural models 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 model database does not include highly detailed facial geometry such as wrinkles, our system will not generate wrinkles on the face model. We might re-use our database to pose facial expressions for different facial models by re-targeting the original data with expression cloning methods [NN01].

The quality of the generated 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 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 model is also influenced by the weight of the model prior term, which provides a tradeoff 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 could work with other types of input devices. For example, the user can specify desired facial deformation by dragging multiple facial points on a large touch screen or track a small set of facial points using a vision based interface.

© Association for Computing Machinery, Inc. 2007.

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.

## References


[AMS7a] ATKESON C. G., MOORE A. W., SCHAAL S.: Locally weighted learning. In *Artificial Intelligence Review*. 1997a. 11:11–73.

[BBPV03] BLANZ V., BASSO C., POGGIO T., VETTER T.: Reanimating faces in images and video. In *Computer Graphics Forum* (2003). 22(3):641–650.

[Bis96] BISHOP C.: *Neural Network for Pattern Recognition*. Cambridge University Press, 1996.

[Bra02] BRAND M.: Incremental singular value decomposition of uncertain data with missing values. In *Proceedings of ECCV* (2002). 707–720.

[BV99] BLANZ V., VETTER T.: A morphable model for the synthesis of 3d faces. In *Proceedings of ACM SIGGRAPH* (1999). 187–194.

[BW00] BISHOP C. M., WINN J. M.: Non-linear bayesian image modelling. In *Proceedings of ECCV*. 2000. 3–17.

[CH05] CHAI J., HODGINS J.: Performance animation from low-dimensional control signals. In *ACM Transactions on Graphics* (2005). 24(3):686–696.

[CJ06] CHANG E., JENKINS O. C.: Sketching articulation and pose for facial animation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2006). 271-280.

[CXH03] CHAI J., XIAO J., HODGINS J.: Vision-based control of 3d facial animation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003). 193–206.

[DH73] DUDA R., HART P.: *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[DSP06] DER K. G., SUMNE R. W., POPOVIĆ J.: Inverse kinematics for reduced deformable models. In *ACM Transactions on Graphics* (2006). 25(3):1174–1179.

[GDT*03] GALASSI M., DAVIES J., THEILER J., GOUGH B., JUNGMAN G., BOOTH M., ROSSI F.: *GNU Scientific Library Reference Manual - Revised Second Edition*. Network Theory Ltd., 2003. ISBN 0954161734.

[GH97] GHAHRAMANI Z., HINTON G. E.: The EM algorithm for mixtures of factor analyzers, 1997.

[GMHP04] GROCHOW K., MARTIN S. L., HERTZMANN A., POPOVIĆ Z.: Style-based inverse kinematics. In *ACM Transactions on Graphics* (2004). 23(3):522–531.

[HDR97] HINTON G. E., DAYAN P., REVOW M.: Modeling the manifolds of images of handwritten digits. In *IEEE Transactions on Neural Networks*. 1997. 8(1):65–74.

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3d freeform design. In *Proceedings of ACM SIGGRAPH* (1999). 409–416.

[JTDP03] JOSHI P., TIEN W. C., DESBRUN M., PIGHIN F.: Learning controls for blendshape based realistic facial animation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Aimation* (2003). 187–192.

[KG05] KHO Y., GARLAND M.: Sketching mesh deformations. In *Proceedings of ACM Symposium on Interactive 3D Graphics and Games* (2005). 147–154.

[Law04] LAWRENCE N. D.: Gaussian process latent variable models for visualization of high dimensional data. In *Advances in Neural Information Processing Systems 16* (2004). 329–336.

[LCF00] LEWIS J. P., CORDNER M., FONG N.: Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of ACM SIGGRAPH*. 2000. 165–172.

[NN01] NOH J., NEUMANN U.: Expression cloning. In *Proceedings of ACM SIGGRAPH* (2001). 277–288.

[NSACO05] NEALEN A., SORKINE O., ALEXA M., COHEN-OR D.: A sketch-based interface for detail-preserving mesh editing. In *ACM Transactions on Graphics* (2005). 24(3):1142–1147.

[Par72] PARKE F. I.: Computer generated animation of faces. In *Proc. ACM National Conference* (1972). 1:451–457.

[PSS99] PIGHIN F., SZELISKI R., SALESIN D.: Resynthesizing facial animation through 3d model-based tracking. In *International Conference on Computer Vision*. 1999. 143–150.

[PW96] PARKE F. I., WATERS K.: Computer facial animation. A.K. Peter, Wellesley, 1996.

[SRC01] SLOAN P.-P., ROSE C., COHEN M. F.: Shape by example. In *ACM Symposium on Interactive 3D Graphics* (2001). 135-143.

[SZGP05] SUMNER R. W., ZWICKER M., GOTSMAN C., POPOVIĆ J.: Mesh-based inverse kinematics. In *ACM Transactions on Graphics* (2005). 24(3):488–495.

[YSvdP05] YANG C., SHARON D., VAN DE PANNE M.: Sketch-based modeling of parameterized objects. In *2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2005).

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: Sketch: an interface for sketching 3d scenes. In *Proceedings of ACM SIGGRAPH* (1996). 163–170.

[ZLGS03] ZHANG Q., LIU Z., GUO B., SHUM H.: Geometry-driven photorealistic facial expression synthesis. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003). 177-186.

[ZSCS04] ZHANG L., SNAVELY N., CURLESS B., SEITZ S. M.: Spacetime faces: high resolution capture for modeling and animation. In *ACM Transactions on Graphics* (2004). 23(3):548–558.


© Association for Computing Machinery, Inc. 2007.