Spacetime Faces: High Resolution Capture for Modeling and Animation

Li Zhang Noah Snavely Brian Curless Steven M. Seitz
University of Washington

Abstract

We present an end-to-end system that goes from video sequences to high resolution, editable, dynamically controllable face models. The capture system employs synchronized video cameras and structured light projectors to record videos of a moving face from multiple viewpoints. A novel spacetime stereo algorithm is introduced to compute depth maps accurately and overcome over-fitting deficiencies in prior work. A new template fitting and tracking procedure fills in missing data and yields point correspondence across the entire sequence without using markers. We demonstrate a data-driven, interactive method for inverse kinematics that draws on the large set of fitted templates and allows for posing new expressions by dragging surface points directly. Finally, we describe new tools that model the dynamics in the input sequence to enable new animations, created via key-framing or texture-synthesis techniques.

CR Categories: I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Stereo, Motion, Surface Fitting; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: shape recovery, facial animation, stereo matching, shape registration, data-driven animation, expression synthesis

1 Introduction

Creating face models that look and move realistically is an important problem in computer graphics. It is also one of the most difficult, as even the most minute changes in facial expression can reveal complex moods and emotions. Yet, the presence of very convincing synthetic characters in recent films makes a strong case that these difficulties can be overcome with the aid of highly skilled animators. Because of the sheer amount of work required to create such models, however, there is a clear need for more automated techniques.

Our objective is to create models that accurately reflect the shape and time-varying behavior of a real person's face from videos. For those models, we seek real-time, intuitive controls to edit expressions and create animations. For instance, dragging the corner of the mouth up should result in a realistic expression, such as a smiling face. Rather than programming these controls manually, we wish to extract them from correlations present in the input video. Furthermore, we wish to use these controls to generate desired animations which preserve the captured dynamics of a real face. (By "dynamics," we mean the time-varying behavior, not the physics *per se*.)

Creating human face models from images is by now a proven approach, with stunning results (e.g., [Blanz and Vetter 1999]). However, the problem of accurately modeling *facial expressions* and other dynamic behavior is still in its infancy. Modeling facial dynamics is essential for creating animations, but it is more difficult to achieve due in part to limitations in current shape capture technology. In particular, laser-scanners and most other high-resolution shape capture techniques do not operate effectively on fast moving scenes (a transition to a smile can occur in a fraction of a second). Furthermore, the problem of creating animation tools that exploit captured models of 3D facial dynamics has yet to be explored.

In this paper, we present a novel, end-to-end system for producing a sequence of high-resolution, time-varying face models using off-the-shelf hardware, and describe tools that use these models for editing and animation. This paper makes several specific technical contributions. First, we introduce a novel, globally consistent spacetime stereo technique to derive high-quality depth maps from structured light video sequences. Next, we propose a new surface fitting and tracking procedure in which the depth maps are combined with optical flow to create face models with vertex correspondence. Once acquired, this sequence of models can be interactively manipulated to create expressions using a data-driven inverse kinematics technique we call *faceIK*. FaceIK blends the models in a way that is automatically adaptive to the number of user-specified controls. We also describe a representation called a *face graph* which encodes the dynamics of the face sequence. The graph can be traversed to create desired animations. While our animation results do not match the artistry of what an expert animator can produce, our approach makes it simple for untrained users to produce face animations.

1.1 Related work

Modeling and synthesizing faces is an active research field in computer graphics and computer vision. Here we review three topics most related to our work: reconstructing moving faces from images, constraint-based face editing, and data-driven face animation. Other related work is discussed throughout the paper, as appropriate.

Reconstructing moving faces from images Very few shape-capture techniques work effectively for rapidly moving scenes. Among the few exceptions are depth-from-defocus [Nayar et al. 1996] and stereo [Faugeras 1993]. Structured light stereo methods have shown particularly promising results for capturing depth maps of moving faces [Proesmans et al. 1996; Huang et al. 2003]. Using projected light patterns to provide dense surface texture, these techniques compute pixel correspondences to derive depth maps for each time instant independently. Products based on these techniques are commercially available.[1] Recent spacetime stereo methods [Zhang et al. 2003a; Davis et al. 2003] additionally integrate information over time to achieve better results. In particular, Zhang et al. [2003a] demonstrate how temporal information can be exploited for dynamic scenes. Compared to these previous structured light stereo methods the shape capture technique presented in this paper produces higher resolution shape models with lower noise.

While the aforementioned shape-capture techniques yield spatially and temporally dense depth maps, a key limitation is that they do

[1] For example, www.3q.com and www.eyetronics.com.

not capture *motion*, i.e., point correspondence over time, making it difficult to repose or reanimate the captured faces. 3D face tracking techniques address this problem by computing the deformation of a deformable 3D face model to fit a sequence of images [Essa et al. 1996; Pighin et al. 1999; Basu et al. 1998; DeCarlo and Metaxas 2002; Blanz et al. 2003] or 3D marker positions [Guenter et al. 1998]. Blanz and Vetter [1999] construct particularly high quality models, represented as linear subspaces of laser-scanned head shapes. Although subspace models are flexible, they fail to reconstruct shapes that are outside the subspace. In order to handle expression variation, Blanz and Vetter [2003] laser-scanned faces under different expressions, a time-consuming process that requires the subject to hold each expression for tens of seconds. A problem with existing face tracking methods in general is that the templates have relatively few degrees of freedom, making it difficult to capture fine-scale dimples and folds which vary from one individual to another and are important characteristic features. We instead work with a generic high resolution template with thousands of degrees of freedom to capture such fine-grain features. This approach is related to the work of Allen et al.[2003] for fitting templates to human body data, except that they rely on markers to provide partial correspondence for each range scan, whereas we derive correspondence information almost entirely from images.

An interesting alternative to traditional template-based tracking is to compute the deformable template and the motion directly from the image sequence. Torresani et al. [2001] and Brand [2001] recover non-rigid structure from a single video assuming the shape lies within a linear subspace. Although these methods are promising and work from regular video streams, they produce relatively low-resolution results, compared to, e.g., structured light stereo.

Direct 3D face editing Following Parke's pioneering work [1972] on blendable face models, most face editing systems are based on specifying blending weights to linearly combine a set of template faces. These weights can be computed indirectly from user-specified constraints [Pighin et al. 1999; Joshi et al. 2003] or fit directly to images [Blanz and Vetter 1999].

Our *faceIK* tool, as a general expression editing interface, is similar to the one in [Joshi et al. 2003]. However, Joshi et al. [2003] segment a face into a region hierarchy *a priori*, which decouples the natural correlation between different parts of the face. Zhang et al. [2003b] address this problem with a hierarchical PCA technique in which user edits may propagate between regions. Our faceIK method instead maintains the correlation across the whole face and only decouples it — automatically and adaptively — as the user introduces more constraints.

Data-driven 3D face animation A focus of our work is to use captured models of human face dynamics to drive animatable face models. Several previous authors explored performance-based methods for animating faces, using either video of an actor [Blanz et al. 2003; Chai et al. 2003], or speech [Bregler et al. 1997; Brand 1999; Ezzat et al. 2002] to drive the animation. These techniques can be considered *data-driven* in that they are based on a sequence of example faces.

Other researchers have explored data-driven animation techniques in the domains of human figure motion [Li et al. 2002; Arikan and Forsyth 2002; Kovar et al. 2002; Lee et al. 2002] and video sprites [Schödl and Essa 2002]. We adapt ideas from these other domains to devise 3D face animation tools.

1.2 System overview

Our system takes as input 6 synchronized video streams (4 monochrome and 2 color) running at 60 frames-per-second (fps), and outputs a 20 fps sequence of high-resolution 3D meshes that capture face geometry, color, and motion, for our data-driven editing and animation techniques. The videos are recorded by a camera rig shown in Figure 1. Three of the cameras capture the left side of the face, and the other three capture the right side.

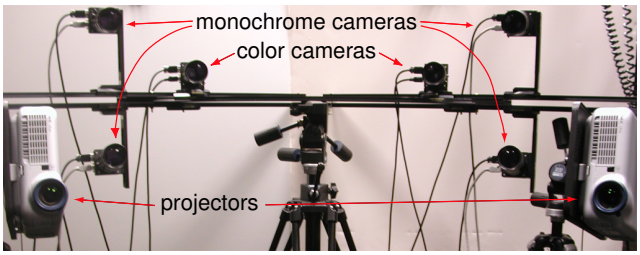


Figure 1: Our face capture rig consists of six video cameras and two data projectors. The two monochrome cameras on the left constitute one stereo pair, and the two on the right constitute a second stereo pair. The projectors provide stripe pattern textures for high quality shape estimation. The color cameras record video streams used for optical flow and surface texture.

To facilitate depth computation, we use two video projectors that project gray-scale random stripe patterns onto the face. The projectors send a "blank" pattern every three frames, which is used to compute both color texture maps and time correspondence information (optical flow). We will refer to these as "non-pattern" frames. All of the components are off-the-shelf.[2]

The following sections describe the stages in the pipeline from the input streams to high-resolution editable and animatable face models. Section 2 introduces the spacetime stereo method to recover time-varying depths maps from the left and right stereo pairs. Section 3 presents a procedure that fits a time-varying mesh to the depth maps while optimizing its vertex motion to be consistent with optical flow. Section 4 describes how this mesh sequence is used for expression editing using faceIK. Section 5 describes two animation tools that use a face graph to model the dynamics present in the captured face sequence.

2 From videos to depth maps

In this section, we present a novel method to recover time-varying depth maps from two synchronized video streams. The method exploits time-varying structured light patterns that are projected onto the face using a standard video projector. We first provide a brief review of traditional stereo matching and prior work in spacetime stereo, to motivate our new approach.

2.1 Binocular stereo matching

There exists an extensive literature on stereo matching algorithms which take as input two images and compute a depth map as output (for a good overview, see [Scharstein and Szeliski 2002]). The key problem is to compute correspondences between points in the left and right image, by searching for pixels of similar intensity or color. Once the correspondence is known, depth values (i.e., distance from the camera) are readily computed [Faugeras 1993]. Generally, the images are assumed to be rectified after calibration,[3] so that the motion is purely horizontal and can be expressed by a 1D disparity function.

[2]We use Basler A301f/fc IEEE1394 cameras, synchronized and running at 60fps, and NEC LT260K projectors.

[3]We calibrate our stereo pairs using Bouguet's software[2001].

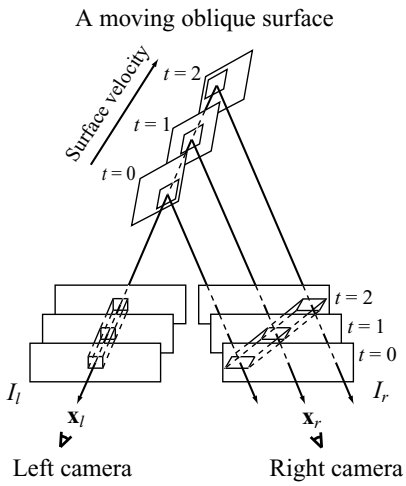


Figure 2: Illustration of spacetime stereo. Two stereo image streams are captured from fixed cameras. The images are shown spatially offset at three different times, for illustration purposes. For a moving surface, a rectangular window in the left view maps to a warped window in the right view. The best affine warp of each spacetime window along epipolar lines is computed for stereo correspondence.

More precisely, given two rectified images $I_l(x,y)$ and $I_r(x,y)$ we wish to compute a disparity function given by $d(x,y)$. For a pixel $I_l(x_0,y_0)$ in the left image, there is often more than one pixel with similar color in the right image. To resolve this ambiguity, most stereo algorithms match small windows W_0 around (x_0,y_0), assuming that the disparity function is locally nearly constant. Mathematically, this matching process involves minimizing the following error function:

$$E(d_0) = \sum_{(x,y)\in W_0} e(I_l(x,y), I_r(x-d_0,y)) \tag{1}$$

where d_0 is shorthand notation for $d(x_0,y_0)$ and $e(p,q)$ is a similarity metric between pixels from two cameras. The size and shape of the window W_0 is a free parameter, with larger windows resulting in smooth depth maps and smaller windows yielding more detailed but also noisier reconstructions.[4] $e(a,b)$ can simply be the squared difference of color differences. We use the "gain-bias" metric [Baker et al. 2003] to compensate for radiometric difference between cameras.

2.2 Spacetime stereo

Given two sequences of images, $I_l(x,y,t)$ and $I_r(x,y,t)$, a time-varying disparity map $d(x,y,t)$ may be computed by applying the above stereo matching procedure to each pair of frames independently. However, the results tend to be noisy, low-resolution (Figure 3(c,d,g,h)), and contain temporal flicker as the shape changes discontinuously from one frame to the next (see the accompanying video). More accurate and stable results are possible by generalizing stereo matching into the temporal domain.

The basic idea, as originally proposed by Zhang et al. [2003a] and Davis et al. [2003], is to assume that disparity is nearly constant over a 3D *spacetime* window $W_0 \times T_0$ around (x_0,y_0,t_0), and solve for $d(x_0,y_0,t_0)$ by minimizing the following error function

$$E(d_0) = \sum_{t\in T_0} \sum_{(x,y)\in W_0} e(I_l(x,y,t), I_r(x-d_0,y,t)) \tag{2}$$

where T_0 may be chosen to be anywhere from a few frames to the whole sequence, depending on how fast the scene is moving. As shown in [Zhang et al. 2003a], assuming locally constant disparity introduces reconstruction artifacts for oblique or moving surfaces. To model such surfaces more accurately, Zhang et al. [2003a] instead approximate the disparity variation linearly within the spacetime window as

$$\begin{aligned} d(x,y,t) &\approx \tilde{d}_0(x,y,t) \stackrel{\text{def}}{=} \\ & d_0 + d_{x_0}\cdot(x-x_0) + d_{y_0}\cdot(y-y_0) + d_{t_0}\cdot(t-t_0) \end{aligned} \tag{3}$$

where $[d_{x_0}, d_{y_0}, d_{t_0}]^{\mathrm{T}}$ is the gradient of the disparity function at (x_0,y_0,t_0). They solve for d_0 together with $[d_{x_0}, d_{y_0}, d_{t_0}]^{\mathrm{T}}$ by minimizing the following error function:

$$E(d_0, d_{x_0}, d_{y_0}, d_{t_0}) = \sum_{t\in T_0} \sum_{(x,y)\in W_0} e(I_l(x,y,t), I_r(x-\tilde{d}_0,y,t)) \tag{4}$$

Under this linearity assumption, a 3D window $W_0 \times T_0$ in I_l maps to a sheared window in I_r, as shown in Figure 2. Consequently, [Zhang et al. 2003a] developed an approach to minimize Eq. (4) by searching for the best matching sheared window at each pixel independently. The resulting depth maps are both higher-resolution and more stable than those produced using standard stereo matching as shown in Figure 3(e,i) and the companion video.

2.3 Globally consistent spacetime stereo

In practice, spacetime stereo produces significantly improved depth maps for moderately-fast moving human shapes. However, it also produces significant ridging artifacts, both evident in the original work [Zhang et al. 2003a] and clearly visible in Figure 3(e). Our analysis indicates that these artifacts are due primarily to the fact that Eq. (4) is minimized for each pixel independently, without taking into account constraints between neighboring pixels. Specifically, computing a disparity map with N pixels introduces $4N$ unknowns: N disparities and $3N$ disparity gradients. While this formulation results in a system that is convenient computationally, it is clearly over-parameterized, since the $3N$ disparity gradients are a function of the N disparities. Indeed, the estimated disparity gradients may not agree with the estimated disparities. For example, $d_x(x,y,t)$ may be quite different from $\frac{1}{2}(d(x+1,y,t) - d(x-1,y,t))$, because $d_x(x,y,t)$, $d(x+1,y,t)$, $d(x-1,y,t)$ are independently estimated for each pixel. This inconsistency between disparities and disparity gradients results in inaccurate depth maps as shown in Figure 3(e,i).

To overcome this inconsistency problem, we reformulate spacetime stereo as a global optimization problem that computes the disparity function, while taking into account gradient constraints between pixels that are adjacent in space and time. Given image sequences $I_l(x,y,t)$ and $I_r(x,y,t)$, the desired disparity function $d(x,y,t)$ minimizes

$$\Gamma(\{d(x,y,t)\}) = \sum_{x,y,t} E(d, d_x, d_y, d_t) \tag{5}$$

subject to the following constraints[5]

$$\begin{aligned} d_x(x,y,t) &= \tfrac{1}{2}(d(x+1,y,t) - d(x-1,y,t)) \\ d_y(x,y,t) &= \tfrac{1}{2}(d(x,y+1,t) - d(x,y-1,t)) \\ d_t(x,y,t) &= \tfrac{1}{2}(d(x,y,t+1) - d(x,y,t-1)) \end{aligned} \tag{6}$$

[4] Some methods allow window size to vary, and compute sizes automatically [Kanade and Okutomi 1994].

[5] At spacetime volume boundaries, we use forward or backward differences instead of central differences.

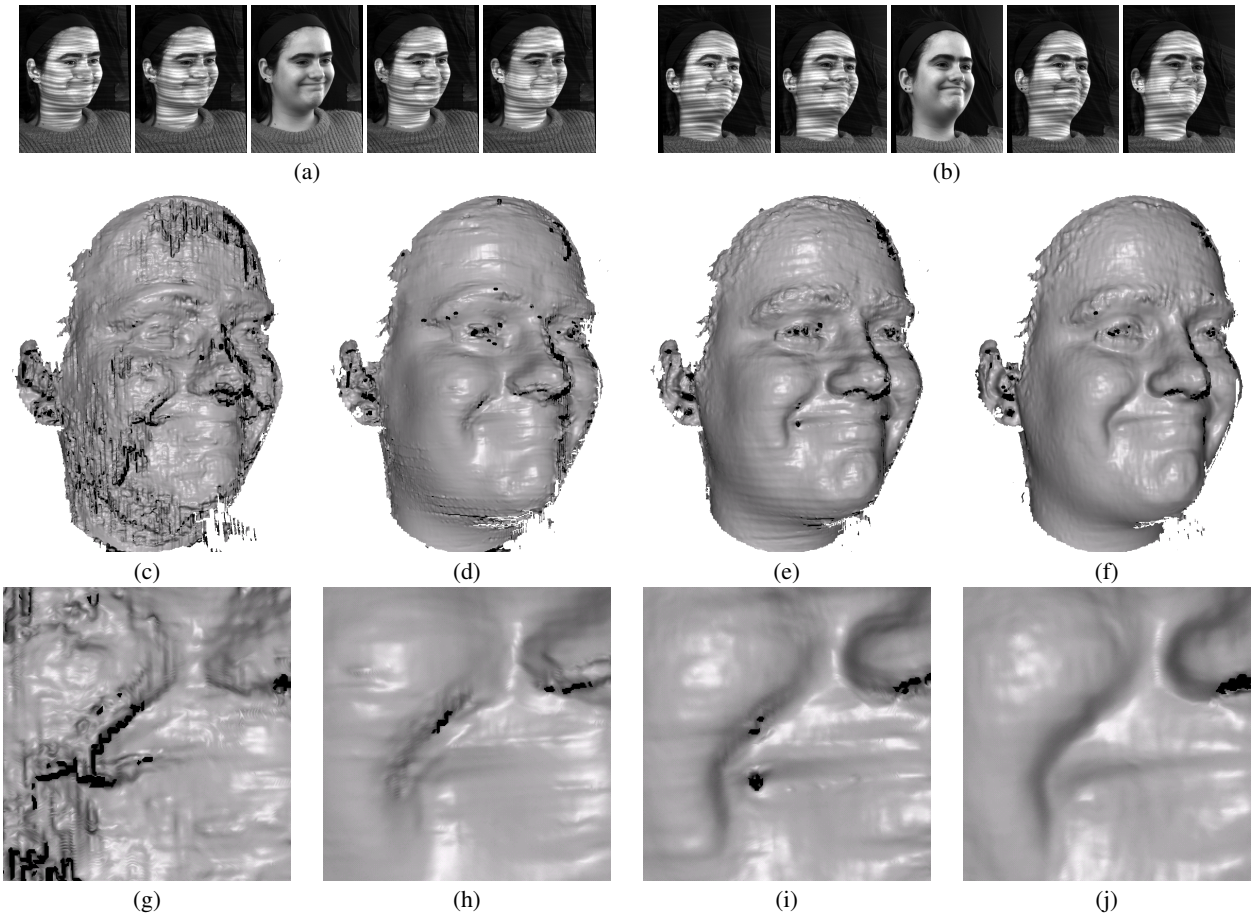


Figure 3: Comparison of four different stereo matching algorithms. (a,b) Five consecutive frames from a pair of stereo videos. The third frames are non-pattern frames. (c) Reconstructed face at the third frame using traditional stereo matching with a $[15\times15]$ window. The result is noisy due to the lack of color variation on the face. (d) Reconstructed face at the second frame using stereo matching with a $[15\times15]$ window. The result is much better because the projected stripes provide texture. However, certain face details are smoothed out due to the need for a large spatial window. (e) Reconstructed face at the third frame using local spacetime stereo matching with a $[9\times5\times5]$ window. Even though the third frame has little intensity variation, spacetime stereo recovers more detailed shapes by considering neighboring frames together. However, it also yields noticeable striping artifacts due to the over-parameterization of the depth map. (f) Reconstructed face at the third frame using our new global spacetime stereo matching with a $[9\times5\times5]$ window. The new method removes most of the striping artifacts while preserving the shape details. (g-j) Closeup comparison of the four algorithms around the nose and the corner of the mouth.

Eq. (5) defines a nonlinear least squares problem with linear constraints. We solve this problem using the Gauss-Newton method [Nocedal and Wright 1999] with a change of variables. To explain our approach, we use $\mathbf{D}$ to denote the concatenation of $d(x,y,t)$ for every (x,y,t) into a column vector. $\mathbf{D}_x$, $\mathbf{D}_y$, and $\mathbf{D}_t$ are defined similarly, by concatenating values of $d_x(x,y,t)$, $d_y(x,y,t)$, and $d_t(x,y,t)$, respectively. Given an initial value of $\mathbf{D}$, $\mathbf{D}_x$, $\mathbf{D}_y$, and $\mathbf{D}_t$, we compute the gradient $\mathbf{b}$ and local Hessian $\mathbf{J}$ of Eq. (5) using Gauss-Newton approximation. Then, the optimal updates $\delta\mathbf{D}$, $\delta\mathbf{D}_x$, $\delta\mathbf{D}_y$, and $\delta\mathbf{D}_t$ are given by

$$\mathbf{J}\begin{bmatrix}\delta\mathbf{D}\\ \delta\mathbf{D}_x\\ \delta\mathbf{D}_y\\ \delta\mathbf{D}_t\end{bmatrix} = -\mathbf{b} \tag{7}$$

Since Eqs. (6) are linear constraints, we represent them by matrix multiplication:

$$\mathbf{D}_x = \mathbf{G}_x\mathbf{D} \quad \mathbf{D}_y = \mathbf{G}_y\mathbf{D} \quad \mathbf{D}_t = \mathbf{G}_t\mathbf{D} \tag{8}$$

where $\mathbf{G}_x$,$\mathbf{G}_y$, and $\mathbf{G}_t$ are sparse matrices encoding the finite difference operations. For example, suppose $d(x,y,t)$ is the i'th component of $\mathbf{D}$, then the only nonzero columns in row i of $\mathbf{G}_x$ are j and j' which correspond to $d(x+1,y,t)$ and $d(x-1,y,t)$ and take values of 0.5 and -0.5, respectively. Substituting Eq. (8) into Eq. (7), we obtain the optimal update $\delta\mathbf{D}$ by solving

$$\begin{bmatrix}\mathbf{I}\\ \mathbf{G}_x\\ \mathbf{G}_y\\ \mathbf{G}_t\end{bmatrix}^{\mathsf{T}}\mathbf{J}\begin{bmatrix}\mathbf{I}\\ \mathbf{G}_x\\ \mathbf{G}_y\\ \mathbf{G}_t\end{bmatrix}\delta\mathbf{D} = -\begin{bmatrix}\mathbf{I}\\ \mathbf{G}_x\\ \mathbf{G}_y\\ \mathbf{G}_t\end{bmatrix}^{\mathsf{T}}\mathbf{b} \tag{9}$$

where $\mathbf{I}$ is an identity matrix of the same dimension as $\mathbf{D}$. We initialize $\mathbf{D}$ using dynamic programming with the spacetime window metric Eq. (2)[6], as described in [Zhang et al. 2003a], and set $\mathbf{D}_x$, $\mathbf{D}_y$, and $\mathbf{D}_t$ to be zero. Then we iteratively solve Eq. (9) and re-compute $\mathbf{J}$ and $\mathbf{b}$, until convergence. Figure 3(f,j) shows the resulting improvement when employing this new spacetime stereo method.

[6]When using dynamic programming for initialization, we use a $[1\times3]$ image window for frame-by-frame matching and a $[1\times3\times3]$ window for spacetime matching.

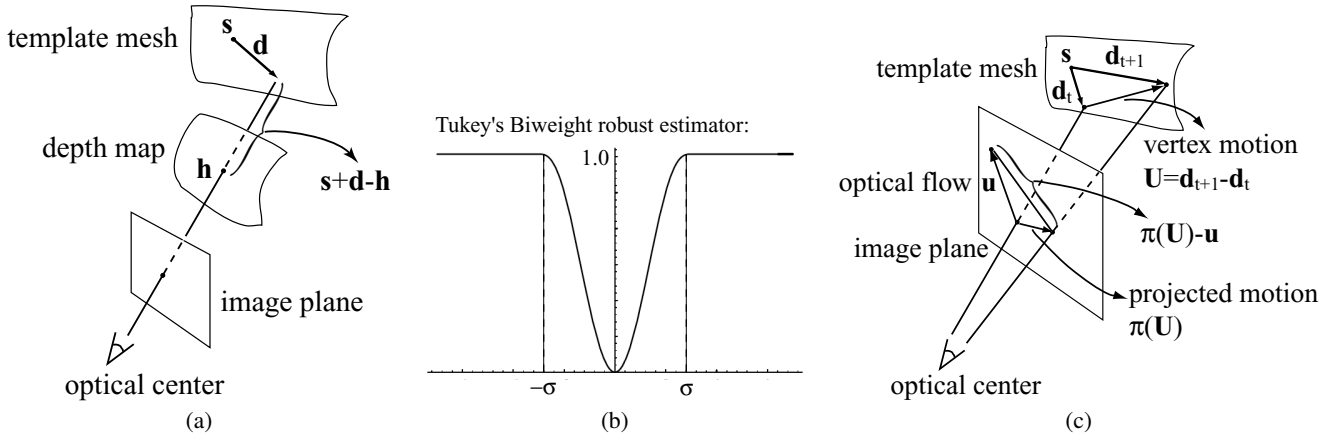


Figure 4: Illustration of the error metric of a vertex used in template fitting. (a) $\mathbf{s}$ is a vertex on the template mesh and $\mathbf{d}$ is its displacement vector. Let $\mathbf{h}$ be the intersection of the depth map, shown as a surface, and the line from optical center to $\mathbf{s}+\mathbf{d}$. $\mathbf{s}+\mathbf{d}-\mathbf{h}$ is the difference vector between $\mathbf{s}+\mathbf{d}$ and the depth map. (b) A plot of Tukey's Biweight robust estimator. (c) $\mathbf{d}_{t+1}$ and $\mathbf{d}_t$ are the displacements for a vertex $\mathbf{s}$ on the template mesh at frame t and $t+1$, respectively. $\mathbf{U}=\mathbf{d}_{t+1}-\mathbf{d}_t$ is the vertex motion from frame t to $t+1$. The projection of $\mathbf{U}$ in the image plane, $\pi(\mathbf{U})$, should be the same as the optical flow $\mathbf{u}$. $\|\pi(\mathbf{U})-\mathbf{u}\|$ is used as a metric for consistency between vertex motion and optical flow.

2.3.1 Scalable implementation

Although $\mathbf{D}_x$, $\mathbf{D}_y$, $\mathbf{D}_t$, and $\mathbf{J}$ are very sparse, solving Eq. (9) using the conjugate gradient method [Press et al. 1993] over the whole video is not practical; a 10-second video of 640×480 resolution at 60Hz comprises nearly 180 millions depth variables! To apply global spacetime stereo matching over a long video, we divide the video into a 3D $(\mathsf{X},\mathsf{Y},\mathsf{T})$ array of 80×80×90 blocks that are optimized in sequence. When optimizing a particular block, we treat as boundary conditions the disparity values in its adjacent blocks that have already been optimized. To speed up the procedure, we distribute the computation over multiple machines while ensuring that adjacent blocks are not optimized simultaneously. While many traversal orders are possible, we found that the following simple strategy suffices: We first optimize blocks with odd T values, and distribute blocks with different T values to different CPU's. On each CPU, we traverse the blocks from left to right and top to bottom. We then repeat the same procedure for blocks with even T values. Our prototype implementation takes 2 to 3 minutes to compute a depth map on a 2.8GHz CPU and each depth map contains approximately 120K depth points.

3 Shape registration

In this section, we present a novel method for computing a single time-varying mesh that closely approximates the depth map sequences while optimizing the vertex motion to be consistent with optical flow between color frames. We start by fitting a template mesh to the pair of depth maps captured in the first non-pattern frame, initialized with a small amount of user guidance. We then track the template mesh through other non-pattern frames in the sequence automatically and without the need for putting markers on the subject's face.

3.1 Template fitting

Let $M=(\mathsf{V},\mathsf{E})$ be an N-vertex triangle mesh representing a template face, with vertex set $\mathsf{V}=\{\mathbf{s}_n\}$ and edge set $\mathsf{E}=\{(n_1,n_2)\}$. Let $\{h_j(x,y)\}_{j=1}^2$ be the two depth maps at frame 1, as shown in Figure 5(a,b,c). Given the relative pose between these depth maps,[7] we wish to solve for a displacement $\mathbf{d}_n$ for each vertex such that the displaced mesh M_1, with vertex set $\{\mathbf{s}_n+\mathbf{d}_n\}$, optimally fits the depth maps. Our fitting metric has two terms: a depth matching term, E_s, and a regularization term E_r.

The depth matching term E_s measures the difference between the depths of vertices of M_1 as seen from each camera's viewpoint and the corresponding values recorded in each depth map. Specifically,

$$E_s(\{\mathbf{d}_n\})=\sum_{j=1}^{2}\sum_{n=1}^{N} w_{n,j}\rho\left(\left[\mathbf{s}_n+\mathbf{d}_n-\mathbf{h}_{n,j}\right]_{\mathbf{z}_j},\sigma_s\right) \qquad (10)$$

where $\mathbf{h}_{n,j}\in\mathbf{R}^3$ is the intersection of the depth map $h_j(x,y)$ and the line from the j'th camera's optical center to $\mathbf{s}_n+\mathbf{d}_n$, as shown in Figure 4(a); $[\cdot]_{\mathbf{z}_j}$ is the z component of a 3-d vector in the j'th camera's coordinate system; $\rho(\cdot,\sigma_s)$ is Tukey's biweight robust estimator, shown in Figure 4(b); and $w_{n,j}$ is a weight factor governing the influence of the j'th depth map on the template mesh. In experiments, we set $\sigma_s=20$, which essentially rejects potential depth map correspondences that are over 20mm away from the template mesh. For $w_{n,j}$, we use the product of the depth map confidence, computed as in [Curless and Levoy 1996], and the dot product of the normals at $\mathbf{h}_{n,j}$ and $\mathbf{s}_n+\mathbf{d}_n$ (clamped above 0). Note that, in practice, we do not need to intersect a line of sight with a surface to compute each $\mathbf{h}_{n,j}$. Instead, we project each displaced template point into the depth map $h_j(x,y)$ and perform bilinear interpolation of depth map values to measure depth differences.

In general, the shape matching objective function is underconstrained. For instance, template surface points after being displaced could bunch together in regions while still matching the depth maps closely. Further, the depth maps do not completely cover the face, and so the template can deform without penalty where there is no data. Thus, we add a regularization term E_r that penalizes large displacement differences between neighboring ver-

[7] We obtain the relative pose between depth maps using the rigid registration tool provided in Scanalyze [2002].

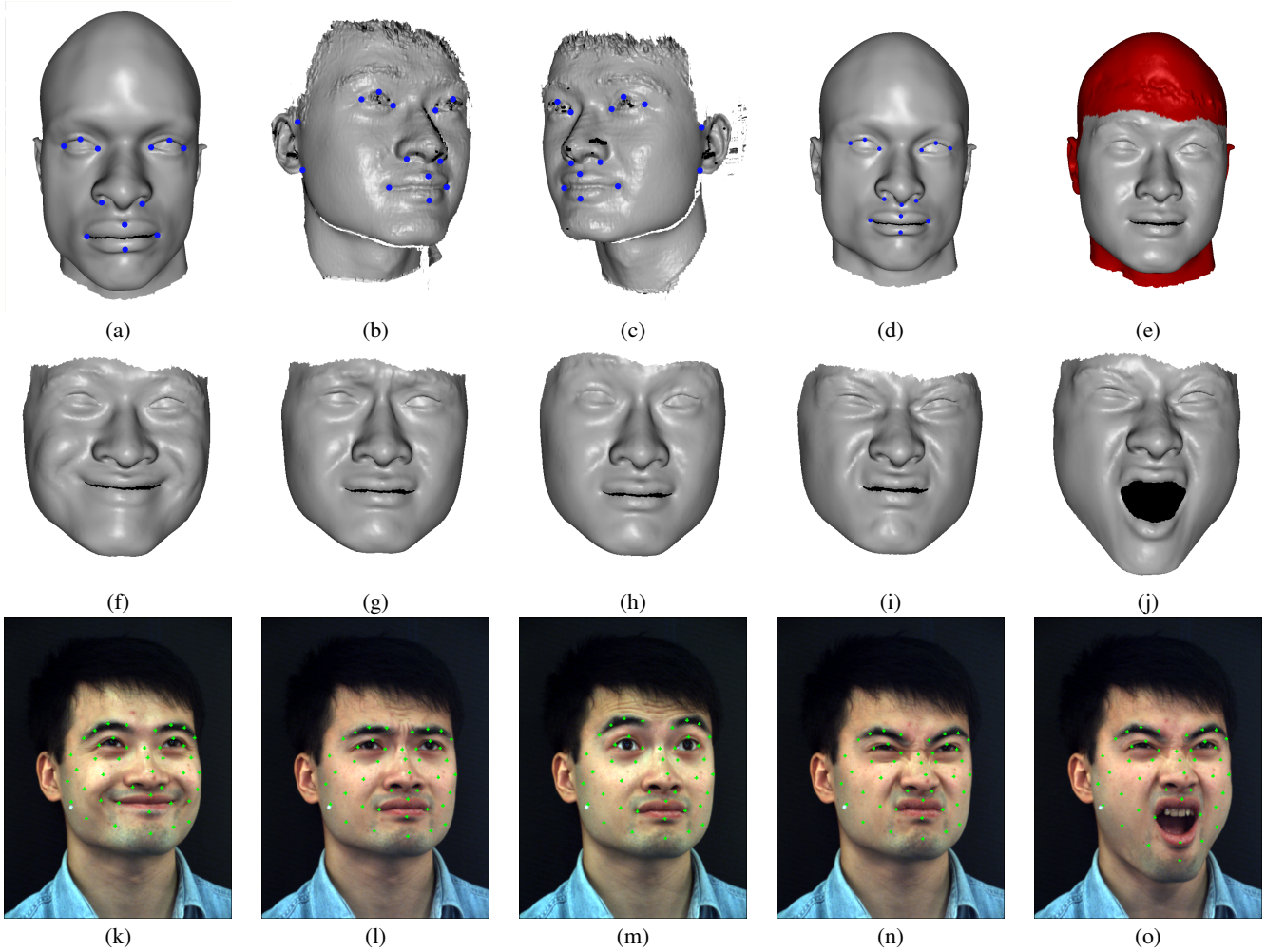


Figure 5: Illustration of the template fitting process. (a) A face template. (b-c) Depth maps from two viewpoints at the first frame. A few corresponding shape feature positions are manually identified on both the face template and the first two depth maps. (d) The template after initial global warp using the feature correspondence. (e) Initial mesh after fitting the warped template to the first two depth maps, without using the feature correspondence. The initial mesh is colored red for regions with unreliable depth or optical flow estimation. (f-j) Selected meshes after tracking the initial mesh through the whole sequence, using both depth maps and optical flows. The process is marker-less and automatic. (k-o) The projection of a set of vertices from the selected meshes on the image plane, shown as green dots, to verify that the vertex motion is consistent with visual motion. Note that the subject had no markers on his face during capture; the green dots are overlaid on the original images purely for visualization purposes.

tices on the template mesh. Specifically,

$$E_r(\{\mathbf{d}_n\}) = \sum_{(n_1,n_2)\in \mathsf{E}} \|\mathbf{d}_{n_1} - \mathbf{d}_{n_2}\|^2 / \|\mathbf{s}_{n_1} - \mathbf{s}_{n_2}\|^2 \qquad (11)$$

To fit the template mesh M to the depth maps at frame 0, we minimize a weighted sum of Eqs. (10) and (11)

$$\Phi = E_s + \alpha E_r \qquad (12)$$

with $\alpha = 2.0$ in our experiments.

We minimize Eq. (12) using the Gauss-Newton method. We initialize the optimization by manually aligning the template with the depth maps. Specifically, we select several corresponding feature positions on both the template mesh and the depth maps (Figure 5(a,b,c)). Next, from these feature correspondences, we solve for an over-constrained global affine transformation to deform the mesh. Finally, we interpolate the residual displacements at the feature positions over the whole surface using a normalized radial basis function [Broomhead and Lowe 1988] as we do for vertex blending coefficients in Section 4.2. After the initial warp, the selected feature correspondences are no longer used for template fitting. As shown in Figure 5(d), the initial alignment does not have to be precise in order to lead to an accurate final fitting result, as illustrated in Figure 5(e).

3.2 Template tracking

Given the mesh M_1 at the first frame, we would now like to deform it smoothly through the rest of the sequence such that the shape matches the depth maps and the vertex motions match the optical flow computed for the non-pattern frames of the color image streams. Let $\{I_k(x,y,t)\}_{k=1}^2$ be color image sequences from two view points with pattern frames removed. We first compute optical flow $\mathbf{u}_k(x,y,t)$ for each sequence using Black and Anan-

dan's method [1993]. The flow $\mathbf{u}_k(x,y,t)$ represents the motion from frame t to $t+1$ in the k'th image plane. We measure the consistency of the optical flow and the vertex inter-frame motion $\mathbf{U}_{n,t} = \mathbf{d}_{n,t+1} - \mathbf{d}_{n,t}$, called *scene flow* in [Vedula et al. 1999], by the following metric:

$$E_m(\{\mathbf{d}_{n,t+1}\}) = \sum_{k=1}^{2}\sum_{n=1}^{N} \rho(\|\pi_k(\mathbf{U}_{n,t}) - \mathbf{u}_{n,t,k}\|, \sigma_m) \qquad (13)$$

where $\pi_k(\mathbf{U}_{n,t})$ is the image projection of $\mathbf{U}_{n,t}$ in the k'th image plane and $\mathbf{u}_{n,t,k}$ is the value of optical flow $\mathbf{u}_k(x,y,t)$ at the corresponding location, shown in Figure 4(c); $\rho(\cdot,\sigma_m)$ is the same Tukey's biweight robust estimator as in Eq. (10), with $\sigma_m = 20$ pixels.

Starting from M_1, we recursively compute M_{t+1} given M_t by optimizing a weighted sum of Eq. (12) and Eq. (13):

$$\Psi = E_s + \alpha E_r + \beta E_m \qquad (14)$$

with $\alpha = 2.0$ and $\beta = 0.5$ in our experiments.

Our mesh tracking method is fully automatic without requiring markers to be placed on the subject's face. In Figure 5(f-o), sample results for mesh tracking are shown in gray shaded rendering. The companion video shows the full sequence both as gray-shaded and color-textured rendering. Each face model has roughly 16K vertices and template tracking takes less than 1 minute per frame.

4 FaceIK

In this section we describe a real-time technique for editing a face to produce new expressions. The key property of the technique is that it exploits correlations in a set of input meshes to propagate user edits to other parts of the face. So, for instance, pulling up on one corner of the mouth causes the entire face to smile. This problem is analogous to the *inverse kinematics* (*IK*) problem in human figure animation in which the goal is to compute the pose of a figure that satisfies one or more user-specified constraints. We therefore call it *faceIK*.

Our approach is based on the idea of representing faces as a linear combination of basis shapes. While linear combinations have been widely used in face modeling [Cootes et al. 1995; Blanz and Vetter 1999; Pighin et al. 1998], the problem of generating reasonable faces using only one constraint (e.g., the corner of the mouth), or just a few, is more difficult because the problem is severely underconstrained. One solution is to compute the coefficients which maximize their likelihood with respect to the data, using principle component analysis (PCA) [Blanz and Vetter 1999; Allen et al. 2003]. The maximum likelihood criterion works well for modeling face variations under similar expressions and human body variations under similar poses. However, applying PCA to facial expressions does not produce good results unless the face is segmented *a priori* into separate regions, e.g., eyes, nose, and mouth [Blanz and Vetter 1999; Joshi et al. 2003; Zhang et al. 2003b]. Segmenting faces into regions decouples the natural correlation between different parts of a face. In practice, the appropriate segmentation is not obvious until run-time when the user decides what expressions to create. For example, to create an asymmetric expression, the left and right sides of the face must be decoupled. As discussed in Section 4.2, under- or over-segmenting the face can result in undesirable editing behavior. We instead describe a method that avoids these problems by adaptively segmenting the face into soft regions based on user edits. These regions are independently modeled using the captured face sequence, and then they are blended into a single expression. We could model these regions using PCA; however, because they are formed by user edits, we would then have to compute principal components, a costly operation, at run-time for each region. To address this problem, we introduce a fast method, *proximity-based weighting* (PBW), to model the regions. We start by describing how to use PBW to model the entire face as a single region.

4.1 Proximity-based weighting

Suppose we are given as input F meshes, each with N vertices. We use $\mathbf{s}_{n,f}$ to denote the n'th vertex in mesh f. Let $\{\mathbf{p}_l\}_{l=1}^{L}$ be user specified 3D constraints, requiring that vertex l should be at position $\mathbf{p}_l$; we call these constraints *control points*.[8] We seek a set of blend coefficients c_f such that for every l,

$$\sum_{f=1}^{F} c_f \mathbf{s}_{l,f} = \mathbf{p}_l \quad \text{and} \quad \sum_{f=1}^{F} c_f = 1 \qquad (15)$$

Because the number of constraints L is generally far fewer than the number of meshes F, we advocate weighting example faces based on *proximity to the desired expression*, i.e., nearby faces are weighted more heavily, a scheme we call proximity based weighting. Specifically, we penalize meshes whose corresponding vertices are far from the control points by minimizing

$$g(\mathbf{c}) = \sum_{f=1}^{F} \phi(\|\bar{\mathbf{s}}_f - \bar{\mathbf{p}}\|) c_f^2 \qquad (16)$$

where $\mathbf{c} = [c_1 \;\; c_2 \;\; \ldots \;\; c_F]^{\mathrm{T}}$, $\bar{\mathbf{s}}_f = [\mathbf{s}_{1,f}^{\mathrm{T}} \;\; \mathbf{s}_{2,f}^{\mathrm{T}} \;\; \ldots \;\; \mathbf{s}_{L,f}^{\mathrm{T}} \;\; 1]^{\mathrm{T}}$, $\bar{\mathbf{p}} = [\mathbf{p}_1^{\mathrm{T}} \;\; \mathbf{p}_2^{\mathrm{T}} \;\; \ldots \;\; \mathbf{p}_L^{\mathrm{T}} \;\; 1]^{\mathrm{T}}$, and $\phi(\cdot)$ is a monotonically increasing function. In our experiments, we simply use $\phi(r) = 1 + r$. Notice that $\|\bar{\mathbf{s}}_f - \bar{\mathbf{p}}\|$ equals the Euclidean distance between the control points and corresponding vertices in mesh f. Minimizing Eq. (16) subject to Eq. (15) encourages small weights for faraway meshes, and can be solved in closed-form as:

$$c_f = \frac{1}{\phi_f} \bar{\mathbf{s}}_f^{\mathrm{T}} \mathbf{a} \qquad (17)$$

where $\phi_f = \phi(\|\bar{\mathbf{s}}_f - \bar{\mathbf{p}}\|)$ and $\mathbf{a} = (\sum_{f=1}^{F} \frac{1}{\phi_f} \bar{\mathbf{s}}_f \bar{\mathbf{s}}_f^{\mathrm{T}})^{-1} \bar{\mathbf{p}}$.

4.1.1 Screen-space constraints

Rather than requiring that constraints be specified in 3D, it is often more natural to specify where the *projection* of a mesh point should move to. Given a set of user-specified 2D constraints $\{\mathbf{q}_l\}_{l=1}^{L}$, Eq. (16) is modified as follows

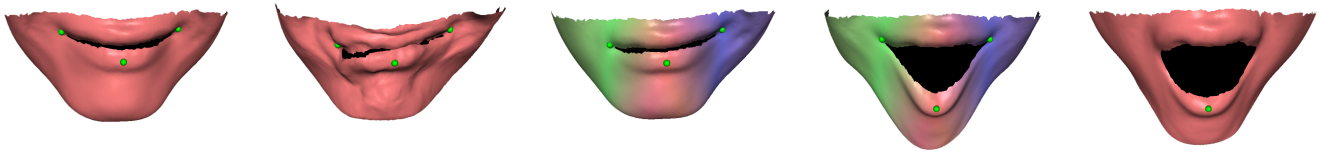
$$g(\mathbf{c}) = \sum_{f=1}^{F} \phi(\|\pi(\bar{\mathbf{s}}_f) - \bar{\mathbf{q}}\|) c_f^2 \qquad (18)$$

such that

$$\pi(\sum_{f=1}^{F} c_f \mathbf{s}_{l,f}) = \mathbf{q}_l \quad \text{and} \quad \sum_{f=1}^{F} c_f = 1 \qquad (19)$$

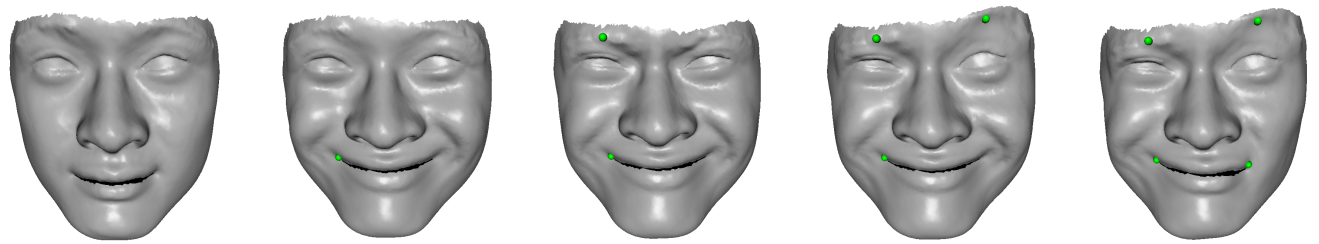
where $\pi(\cdot)$ is the image projection operator, $\bar{\mathbf{q}} = [\mathbf{q}_1^{\mathrm{T}} \;\; \mathbf{q}_2^{\mathrm{T}} \;\; \ldots \;\; \mathbf{q}_L^{\mathrm{T}} \;\; 1]^{\mathrm{T}}$, and $\pi(\bar{\mathbf{s}}_f) \stackrel{\text{def}}{=} [\pi(\mathbf{s}_{1,f})^{\mathrm{T}} \;\; \pi(\mathbf{s}_{2,f})^{\mathrm{T}} \;\; \ldots \;\; \pi(\mathbf{s}_{L,f})^{\mathrm{T}} \;\; 1]^{\mathrm{T}}$. Since π is in

[8] We assume the L constraints are for the first L mesh vertices, to simplify notation without loss of generality.



(a) (b) (c) (d) (e)

Figure 6: Advantages of adaptive face segmentation with faceIK. Many face editing techniques pre-segment a face into regions (e.g., mouth, nose, eyes) and model each region separately with PCA. (a) Three symmetric control points are used to create a symmetric smile by applying PCA on the mouth region to compute the maximum likelihood (ML) shape. (b) When the control points become asymmetric, ML behaves poorly, since all input mouth shapes are roughly symmetric. (c) For the same control point positions, faceIK creates an asymmetric smile by dividing the mouth into three soft regions (indicated by color variations) and blending the influence of each control point. Each control point influences its region using PBW in real-time. By contrast, using PCA would require computing principal components, a costly operation, at run-time for each new region. (d) With the same control vertices as in (c), if the point on the lower lip is moved by itself, the mouth opens unnaturally, because the two control points on the mouth corners decouple their correlation to the lower lip. (e) With only one control point on the lower lip, the mouth opens more naturally. These comparisons indicate that it is more desirable to adaptively segment a face into regions based on user edits, rather than *a priori*.



(a) (b) (c) (d) (e)

Figure 7: A faceIK editing session. From left to right, we show the creation of a complex expression by adding control points one at a time, starting from neutral.

general nonlinear, we approximate Eq. (19) by

$$\sum_{f=1}^{F} c_f \pi(\mathbf{s}_{l,f}) = \mathbf{q}_l \quad \text{and} \quad \sum_{f=1}^{F} c_f = 1 \tag{20}$$

This approximation works well in our experience, and minimizing Eq. (18) subject to Eq. (20) yields the closed-form solution:

$$c_f = \frac{1}{\phi_f} \pi(\bar{\mathbf{s}}_f)^{\mathsf{T}} \mathbf{a} \tag{21}$$

where $\phi_f = \phi(\|\pi(\bar{\mathbf{s}}_f) - \bar{\mathbf{q}}\|)$ and $\mathbf{a} = (\sum_{f=1}^{F} \frac{1}{\phi_f} \pi(\bar{\mathbf{s}}_f)\pi(\bar{\mathbf{s}}_f)^{\mathsf{T}})^{-1}\bar{\mathbf{q}}$.

4.2 Local influence maps

The faceIK method presented so far assumes that the entire face is created by linear interpolation of nearby meshes. However, this assumption is too limiting, since, for example, an asymmetric smile cannot be generated from a data set that contains only symmetric smiles. We therefore propose a method to blend different face regions by defining an *influence map* for each control point. Specifically, we give each control point greater influence on nearby mesh points, and then blend the influences over the entire mesh to allow for a broader range of expressions that do not exist in the input data.

Accordingly, for each of the L control points, we compute a set of blending coefficients $\mathbf{c}_l$ whose components sum to 1, by minimizing Eq. (16) or Eq. (18). This process is done independently for each control point. The resulting L meshes are then blended together, using normalized radial basis functions [Broomhead and Lowe 1988] to define spatially-varying weights. Specifically, we set the blending coefficient for vertex $\mathbf{s}_n$ as follows

$$\mathbf{c}(\mathbf{s}_n) = \sum_{l=1}^{L} B(\mathbf{s}_n, \mathbf{s}_l)\hat{\mathbf{c}}_l \tag{22}$$

where $B(\mathbf{s}_n, \mathbf{s}_l) = \frac{\exp(-\|\mathbf{s}_n - \mathbf{s}_l\|^2 / r_l^2)}{\sum_{l'=1}^{L} \exp(-\|\mathbf{s}_n - \mathbf{s}_{l'}\|^2 / r_{l'}^2)}$ with $r_l = \min_{l' \neq l} \|\mathbf{s}_l - \mathbf{s}_{l'}\|$. We prove in the appendix that the components of $\mathbf{c}(\mathbf{s}_n)$ sum to 1 given that $\sum_{l=1}^{L} B(\mathbf{s}_n, \mathbf{s}_l) = 1$. For each vertex $\mathbf{s}_n$, we use $\mathbf{c}(\mathbf{s}_n)$ to blend corresponding vertices in the mesh data set.

Figure 6 shows the advantage of using local influence maps to adaptively segment the face based on user-interaction, rather than specifying regions *a priori*. The main observation is that the optimal set of regions depends on the desired edit; for instance, generating an asymmetric smile from a set of roughly symmetric faces requires decoupling the left and right sides of the mouth. However, an edit that opens the mouth is more easily obtained *without* this decoupling. Our PBW scheme supports the adaptive segmentation in real-time.

Figure 7 shows a sequence of edits that leads from a neutral face to a complex expression. As shown in the accompanying video, our faceIK tool runs in real time, providing interactive direct-manipulation editing.

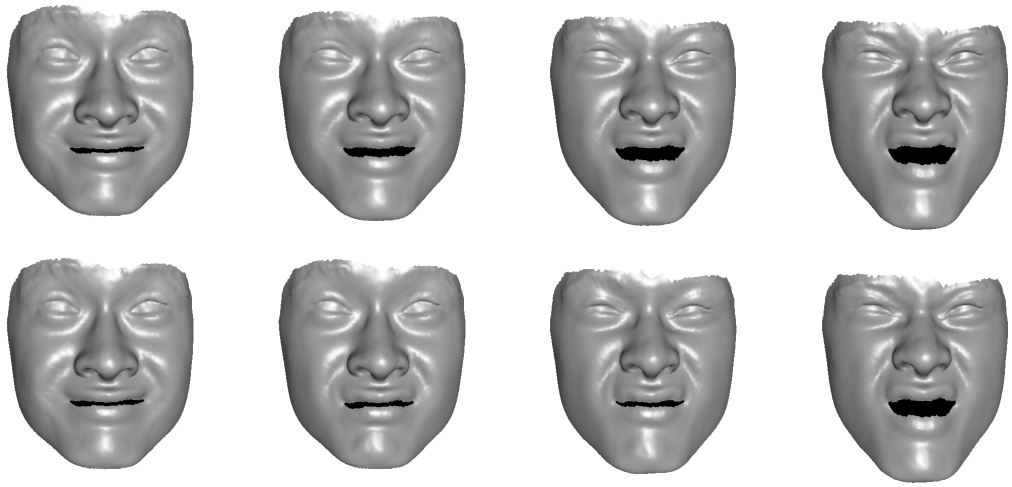


Figure 8: Illustration of linear interpolation (top row) vs. data driven interpolation (bottom row), with the first and last columns as key frames. Linear interpolation makes the mouth and the eyes move synchronously, which looks less realistic when played as an animation. Data driven interpolation, instead, first purses the mouth, then squints the eyes, and finally opens the mouth. The sequential nature of the data-driven interpolation for this example arose naturally because that is the way the real subject behaved.

5 Data-driven animation

Producing realistic animations of the human face is extremely challenging, as subtle differences in dynamics and timing can have a major perceptual effect. In this section, we exploit the facial dynamics captured in our reconstructed 3D face sequences to create tools for face animation. In particular we describe two such tools, one for producing random infinite face sequences, and another for data-driven interpolation of user-specified key frames.

Before introducing these tools, we first describe our model of face dynamics using a graph representation. Our approach adapts related graph techniques used in video textures [Schödl et al. 2000] and character animation [Kovar et al. 2002; Arikan and Forsyth 2002; Lee et al. 2002] to the domain of face animation.

5.1 Face graphs

Let $M_1, \ldots, M_F$ be a sequence of face meshes. We represent face dynamics using a fully connected graph with F nodes corresponding to the faces; we call this the *face graph*. The weight of the edge between nodes i and j specifies the cost of a transition from M_i to M_j in an animation sequence. This cost should respect the dynamics present in the input sequence, balanced with a desire for continuity. Given two frames M_i and M_j in the input sequence, we define the weight w of edge (i, j) as

$$w(i,j) = \texttt{dist}(M_{i+1}, M_j) + \lambda \texttt{dist}(M_i, M_j) \qquad (23)$$

where `dist` is a distance metric between meshes. (We use the L_2 norm, summed over all the vertices.) The first term prefers following the input sequence, while the second term penalizes large jumps.

5.2 Random walks through face space

Video textures [Schödl et al. 2000] generates non-repeating image sequences of arbitrary length. The same technique can be used to generate random, continuously-varying face animations. To do so we simply perform a random walk through the face graph. As in [Schödl et al. 2000], we define the probability of a transition from mesh M_i to mesh M_j to be $P_{ij} = e^{-w(i,j)/\sigma}$, normalizing so that the sum of P_{ij} over all j is 1. The parameter σ is used to define the frequency of transitions between different parts of the input sequence; lower values create animations that closely follow the input sequence, whereas higher values promote more random behavior.

As in [Schödl et al. 2000], we disguise transitions between two meshes that are not consecutive in the input sequence by a weighted blend of the two subsequences across the transition. Results are shown in the companion video.

5.2.1 Animation with regions

A limitation of the method described so far is that the frames composing the animation are constrained to lie within the set of input meshes. We therefore generalize this approach by defining regions on the face, animating the regions separately using the above method, and then blending the results into a single animation.

Rather than grouping the vertices of the meshes into disjoint regions, we create "soft" regions using control points to define a set of weights for each vertex, as described in Section 4.2. The influence maps are taken into account in the computation of the cost graph by defining $d(M_i, M_j)$ to be weighted sum-of-squared distance, with per-vertex weights defined by the influence map.

The companion video shows an animation generated from this method using two control points. While a majority of the resulting sequence looks natural and compelling, it also contains some unnatural frames and transitions, due to the fact that different parts of the face are animated independently.

5.3 Data-driven keyframe animation

While the random walk technique produces animation very easily, it does not provide a mechanism for user control. However, the same concepts may be used to support traditional keyframe animations, in which in-between frames are automatically generated from user-specified constraint frames. The in-between frames are generated using a data-driven interpolation method, which seeks to follow minimum-cost paths through the graph [Kovar et al. 2002; Arikan and Forsyth 2002; Lee et al. 2002; Schödl and Essa 2002].

Suppose that an animator has a sequence of meshes available, and wants to animate a transition between two expressions that appear in the sequence. In the simplest case, the expressions comprise the endpoints of a subsequence of the input sequence. More generally, the interpolation must blend two or more noncontiguous subsequences.

To find a path between two keyframes M_i and M_j, we construct the graph defined above, then search for the shortest path connecting M_i and M_j using Dijkstra's algorithm [Kozen 1992]. The result is a sequence of meshes. We then compute a per-vertex cubic-spline interpolation of the mesh sequence to generate a continuous animation, which is sampled using a user-specified parameterization to produce a desired set of in-between frames with desired timing.

5.3.1 Keyframe animation with regions

The keyframe interpolation method is extended to work with multiple regions using the same technique as described for random walks. In particular, a separate graph is defined for each region. Optimal paths on each graph are computed independently, and the resulting animations are blended together using the influence functions to produce a composite key-frame animation. Figure 8 shows an example keyframe animation, comparing our data-driven interpolation to traditional linear interpolation. We have also created a forty three second animation (shown in the companion video) using our data-driven technique. The animation uses nineteen keyframes, and each keyframe has three control points.

6 Discussion and future work

We have presented an end-to-end system that takes several video sequences as input and generates high resolution, editable, dynamically controllable face models. The capture system employs synchronized video cameras and structured light projectors to capture streams of images from multiple viewpoints. Specific technical contributions include first a novel spacetime stereo algorithm that overcomes over-fitting deficiencies in prior work. Second, we described a new template fitting and tracking procedure that fills in missing data and brings the surfaces into correspondence across the entire sequence without the use of markers. Third, we demonstrated a data-driven, interactive method for face editing that draws on the large set of fitted templates and allows specification of expressions by dragging surface points directly. Finally, we described new tools that model the dynamics in the input sequence to enable new animations, created via key-framing or texture-synthesis techniques.

There are many important topics to explore in future work. First, the resolution of our template mesh is only about one-eighth of the depth maps, and we only fit the template to one-third of the depth maps (at non-pattern frames). A natural extension of our current work is to employ a hierarchical fitting approach to use templates whose resolutions are comparable with the depth maps, and also interpolate color along optical flow to obtain face models at 60Hz.

Our capture technique requires illuminating the subject's face with two bright projectors, and involves a relatively large rig with multiple mounted cameras. One could imagine, however, embedding six small cameras on the sides of a monitor, and use imperceptible structured light [Raskar et al. 1998] to make the capture process less objectionable.

Our registration technique depends on optical flow estimation which can be unreliable for textureless regions. Although regularization helps to produce smooth meshes, we did observe some "vertex swimming" artifacts over the textureless regions. In the future, we hope to incorporate temporal coherence.

Our faceIK method generates natural results for control points that are relatively near the input faces, but can produce bizarre (and even disturbing) results for larger extrapolations. Although this behavior is not surprising, the user must explore the space of faces by trial and error. More useful would be if the tool could constrain the edits to the range of *reasonable* faces, by learning a set of good controls. Another limitation of faceIK is that, although it allows the user to segment the face adaptively, within an animation the segmentation cannot be changed. Therefore, before creating an animation, the user must specify enough control points so that any desired keyframe can be created. This limitation did not pose a problem when we created the animation in the accompanying video; three controls (on both eyebrows and the lower lip) were enough to create the expressions we wanted. However, it would be desirable to allow the controls to vary across keyframes.

The animation tools presented in this paper are quite simple and could be extended and improved in several ways. One limitation is that we assume that the key frames are blends of the input frames. Although different regions can be controlled separately, the approach does not provide good support for extrapolated faces, since such faces may not be part of a motion sequence (i.e., there is no natural successor frame). Another limitation is that our data-driven interpolation technique requires a rich face sequence in order to produce natural-looking transitions between all the captured expressions. If our technique fails to find a desired transition in the face sequence, it may choose to use linear interpolation or an unnatural backwards transition instead. In addition, eye blinks may occur at inopportune moments, which complicates animation, and gaze direction is fixed by the input sequence; more careful modeling of the eyes, as well as insertion of teeth, would improve the quality of resulting animations. Finally, more intuitive control of timing would also help produce more realistic keyframe animations. All of these problems are important areas for future work.

While we have focused on animating a single face, it would be interesting to explore variations in dynamic behaviors among different human faces, similar in spirit to what has been done for static shape variations [Blanz and Vetter 1999; Allen et al. 2003].

Acknowledgements

We would like to thank Kiera Henning, Terri Moore, and Ethel Evans for allowing us to use their face data. We also thank Jiwon Kim, Kiera Henning, Brett Allen, Yung-Yu Chuang, and Wilmot Li for their help in preparing the paper and companion video. This work was supported in part by National Science Foundation grants CCR-0098005, IIS-0049095, and EIA-0321235, an Office of Naval Research YIP award, the UW Animation Research Labs, and Microsoft Corporation.

Appendix

In this appendix, we prove that the blending coefficients for vertex $\mathbf{s}$, $\mathbf{c}(\mathbf{s})$, sum up to 1 given that $\sum_{l=1}^{L} B(\mathbf{s},\mathbf{s}_l) = 1$. Our proof is based on two facts. To state the facts succinctly, we first introduce two concepts. A vector is called *normalized* if its components sum to 1. A matrix is called *normalized* if all of its rows are normalized.

FACT1. If an invertible square matrix $\mathbf{A} = [a_{i,j}]$ is normalized, then $\mathbf{B} = \mathbf{A}^{-1}$ is also normalized.

PROOF. Let $\mathbf{B} = [b_{i,j}]$. $\mathbf{B} = \mathbf{A}^{-1} \Rightarrow \forall i, \forall j, \sum_k b_{i,k} a_{k,j} = \delta_{i,j} \Rightarrow$ $\forall i, 1 = \sum_j \delta_{i,j} = \sum_j \sum_k b_{i,k} a_{k,j} = \sum_k \sum_j b_{i,k} a_{k,j} = \sum_k b_{i,k} \sum_j a_{k,j} = \sum_k b_{i,k}$.

FACT2. If a m by n matrix $\mathbf{A} = [a_{i,j}]$ is normalized and a m dimensional vector $\mathbf{b} = [b_i]$ is normalized, then the n dimensional vector $b^{\mathrm{T}}\mathbf{A}$ is also normalized.

PROOF. $b^{\mathrm{T}}\mathbf{A} = [\sum_i b_i a_{i,j}] \Rightarrow \sum_j \sum_i b_i a_{i,j} = \sum_i \sum_j b_i a_{i,j} = \sum_i b_i \sum_j a_{i,j} = \sum_i b_i = 1$.

Let $\mathbf{C} = [\mathbf{c}_1\ \mathbf{c}_2\ \ldots\ \mathbf{c}_L]^{\mathrm{T}}$ and $\hat{\mathbf{C}} = [\hat{\mathbf{c}}_1\ \hat{\mathbf{c}}_2\ \ldots\ \hat{\mathbf{c}}_L]^{\mathrm{T}}$. We know from the construction of RBF that $\mathbf{C} = \left[B(\mathbf{s}_{l'}, \mathbf{s}_l)\right]\hat{\mathbf{C}}$. Because both $\mathbf{C}$ and $\left[B(\mathbf{s}_{l'}, \mathbf{s}_l)\right]$ are normalized, the matrix $\hat{\mathbf{C}} = \left[B(\mathbf{s}_{l'}, \mathbf{s}_l)\right]^{-1}\mathbf{C}$ is also normalized, according to FACT1 and FACT2. Again, from the definition of RBF, we know that $\mathbf{c}(\mathbf{s})^{\mathrm{T}} = [B(\mathbf{s}, \mathbf{p}_l)]^{\mathrm{T}}\hat{\mathbf{C}}$. Because both vector $[B(\mathbf{s}, \mathbf{p}_l)]$ and matrix $\hat{\mathbf{C}}$ are normalized, $\mathbf{c}(\mathbf{s})$, the blending coefficient for vertex $\mathbf{s}$, is also normalized.

References

ALLEN, B., CURLESS, B., AND POPOVIC, Z. 2003. The space of human body shapes: reconstruction and parameterization from range scans. In *SIGGRAPH Conference Proceedings*, 587–594.

ARIKAN, O., AND FORSYTH, D. A. 2002. Synthesizing constrained motions from examples. In *SIGGRAPH Conference Proceedings*, 483–490.

BAKER, S., GROSS, R., AND MATTHEWS, I. 2003. Lucas-kanade 20 years on: A unifying framework: Part 3. Tech. Rep. CMU-RI-TR-03-35, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, November.

BASU, S., OLIVER, N., AND PENTLAND, A. 1998. 3d lip shapes from video: A combined physical-statistical model. *Speech Communication 26*, 1, 131–148.

BLACK, M. J., AND ANANDAN, P. 1993. Robust dense optical flow. In *Proc. Int. Conf. on Computer Vision*, 231–236.

BLANZ, V., AND VETTER, T. 1999. A morphable model for the synthesis of 3D faces. In *SIGGRAPH Conference Proceedings*, 187–194.

BLANZ, V., BASSO, C., POGGIO, T., AND VETTER, T. 2003. Reanimating faces in images and video. In *Proceedings of EUROGRAPHICS*, vol. 22, 641–650.

BOUGUET, J.-Y. 2001. *Camera Calibration Toolbox for Matlab*. http://www.vision.caltech.edu/bouguetj/calib_doc/index.html.

BRAND, M. 1999. Voice puppetry. In *SIGGRAPH Conference Proceedings*, 21–28.

BRAND, M. 2001. Morphable 3D models from video. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 456–463.

BREGLER, C., COVELL, M., AND SLANEY, M. 1997. Video rewrite: Visual speech synthesis from video. In *SIGGRAPH Conference Proceedings*, 353–360.

BROOMHEAD, D. S., AND LOWE, D. 1988. Multivariable functional interpolation and adptive networks. *Complex Systems 2*, 321–355.

CHAI, J., JIN, X., AND HODGINS, J. 2003. Vision-based control of 3d facial animation. In *Proceedings of Eurographics/SIGGRAPH Symposium on Computer Animation*, 193–206.

COOTES, T. F., TAYLOR, C. J., COOPER, D. H., AND GRAHAM, J. 1995. Active shape models—their training and application. *Computer Vision and Image Understanding 61*, 1, 38–59.

CURLESS, B., AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *SIGGRAPH Conference Proceedings*, 303–312.

DAVIS, J., RAMAMOORTHI, R., AND RUSINKIEWICZ, S. 2003. Spacetime stereo: A unifying framework for depth from triangulation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 359–366.

DECARLO, D., AND METAXAS, D. 2002. Adjusting shape parameters using model-based optical flow residuals. *IEEE Trans. on Pattern Analysis and Machine Intelligence 24*, 6, 814–823.

ESSA, I., BASU, S., DARRELL, T., AND PENTLAND, A. 1996. Modeling, tracking and interactive animation of faces and heads using input from video. In *Proceedings of the Computer Animation*, IEEE Computer Society, 68–79.

EZZAT, T., GEIGER, G., AND POGGIO, T. 2002. Trainable videorealistic speech animation. In *SIGGRAPH Conference Proceedings*, 388–398.

FAUGERAS, O. 1993. *Three-Dimensional Computer Vision*. MIT Press.

GUENTER, B., GRIMM, C., WOOD, D., MALVAR, H., AND PIGHIN, F. 1998. Making faces. In *SIGGRAPH Conference Proceedings*, 55–66.

HUANG, P. S., ZHANG, C. P., AND CHIANG, F. P. 2003. High speed 3-d shape measurement based on digital fringe projection. *Optical Engineering 42*, 1, 163–168.

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

KANADE, T., AND OKUTOMI, M. 1994. A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE Trans. on Pattern Analysis and Machine Intelligence 16*, 9, 920–932.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *SIGGRAPH Conference Proceedings*, 473–482.

KOZEN, D. C. 1992. *The Design and Analysis of Algorithms*. Springer.

LEE, J., CHAI, J., REITSMA, P. S. S., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. In *SIGGRAPH Conference Proceedings*, 491–500.

LI, Y., WANG, T., AND SHUM, H.-Y. 2002. Motion texture: A two-level statistical model for character motion synthesis. In *SIGGRAPH Conference Proceedings*, 465–472.

NAYAR, S. K., WATANABE, M., AND NOGUCHI, M. 1996. Real-time focus range sensor. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*, 12, 1186–1198.

NOCEDAL, J., AND WRIGHT, S. J. 1999. *Numerical Optimization*. Springer.

PARKE, F. I. 1972. Computer generated animation of faces. In *Proceedings of the ACM annual conference*, ACM Press, 451–457.

PIGHIN, F., HECKER, J., LISCHINSKI, D., SALESIN, D. H., AND SZELISKI, R. 1998. Synthesizing realistic facial expressions from photographs. In *SIGGRAPH Conference Proceedings*, 75–84.

PIGHIN, F., SALESIN, D. H., AND SZELISKI, R. 1999. Resynthesizing facial animation through 3D model-based tracking. In *Proc. Int. Conf. on Computer Vision*, 143–150.

PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., AND VETTERLING, W. T. 1993. *Numerical Recipes in C : The Art of Scientific Computing*, 2nd ed. Cambridge University Press.

PROESMANS, M., GOOL, L. V., AND OOSTERLINCK, A. 1996. One-shot active 3D shape acquization. In *Proc. Int. Conf. on Pattern Recognition*, 336–340.

PULLI, K., AND GINZTON, M. 2002. *Scanalyze*. http://graphics.stanford.edu/software/scanalyze/.

RASKAR, R., WELCH, G., CUTTS, M., LAKE, A., STESIN, L., AND FUCHS, H. 1998. The office of the future: A unified approach to image-based modeling and spatially immersive displays. In *SIGGRAPH Conference Proceedings*, 179–188.

SCHARSTEIN, D., AND SZELISKI, R. 2002. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. on Computer Vision 47*, 1, 7–42.

SCHÖDL, A., AND ESSA, I. A. 2002. Controlled animation of video sprites. In *Proceedings of Eurographics/SIGGRAPH Symposium on Computer Animation*, ACM Press, 121–127.

SCHÖDL, A., SZELISKI, S., SALESIN, D. H., AND ESSA, I. 2000. Video textures. In *SIGGRAPH Conference Proceedings*, 489–498.

TORRESANI, L., YANG, D. B., ALEXANDER, E. J., AND BREGLER, C. 2001. Tracking and modeling non-rigid objects with rank constraints. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 493–500.

VEDULA, S., BAKER, S., RANDER, P., COLLINS, R., AND KANADE, T. 1999. Three-dimensional scene flow. In *Proc. Int. Conf. on Computer Vision*, 722–729.

ZHANG, L., CURLESS, B., AND SEITZ, S. M. 2003. Spacetime stereo: Shape recovery for dynamic scenes. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 367–374.

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