Mesh-Based Inverse Kinematics

Robert W. Sumner Matthias Zwicker Craig Gotsman[†] Jovan Popović

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

[†]Harvard University

Abstract

The ability to position a small subset of mesh vertices and produce a *meaningful* overall deformation of the entire mesh is a fundamental task in mesh editing and animation. However, the class of meaningful deformations varies from mesh to mesh and depends on mesh kinematics, which prescribes valid mesh configurations, and a selection mechanism for choosing among them. Drawing an analogy to the traditional use of skeleton-based inverse kinematics for posing skeletons, we define *mesh-based inverse kinematics* as the problem of finding meaningful mesh deformations that meet specified vertex constraints.

Our solution relies on example meshes to indicate the class of meaningful deformations. Each example is represented with a feature vector of deformation gradients that capture the affine transformations which individual triangles undergo relative to a reference pose. To pose a mesh, our algorithm efficiently searches among all meshes with specified vertex positions to find the one that is closest to some pose in a nonlinear span of the example feature vectors. Since the search is not restricted to the span of example shapes, this produces compelling deformations even when the constraints require poses that are different from those observed in the examples. Furthermore, because the span is formed by a nonlinear blend of the example feature vectors, the blending component of our system may also be used independently to pose meshes by specifying blending weights or to compute multi-way morph sequences.

CR Categories: I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism—Animation

Keywords: Deformation, Geometric Modeling, Animation

Authors' contact:
{sumner|matthias|gotsman|jovan}@csail.mit.edu

1 Introduction

The shape of a polygon mesh depends on the positions of its many vertices. Although such shapes can be manipulated by displacing every vertex manually, this process is avoided because it is tedious and error-prone. Existing mesh editing tools allow the modeler to sculpt a mesh's shape by editing only a few vertices and use general numerical criteria such as detail preservation to position the remaining ones. In animation, the class of *meaningful* deformations cannot be captured by simple numerical criteria because it varies from mesh to mesh. The mesh kinematics—how the vertices are allowed to move—as well as a mechanism for choosing one out of many kinematically valid meshes must be considered when posing a mesh in a meaningful way. Skeleton-based articulation is often used in animation to approximate mesh kinematics compactly. However, skeletons cannot easily provide the rich class of deformations afforded by sculpting techniques and only allow indirect interaction with the mesh via the joint angles of the skeleton. Our method allows the user to directly position any subset of mesh vertices and produces a meaningful deformation automatically. Complex pose changes can be accomplished intuitively by manipulating only a few vertices. In analogy to traditional skeleton-based inverse kinematics for posing skeletons, we call this general problem *mesh-based inverse kinematics*, and our example solution MESHIK.

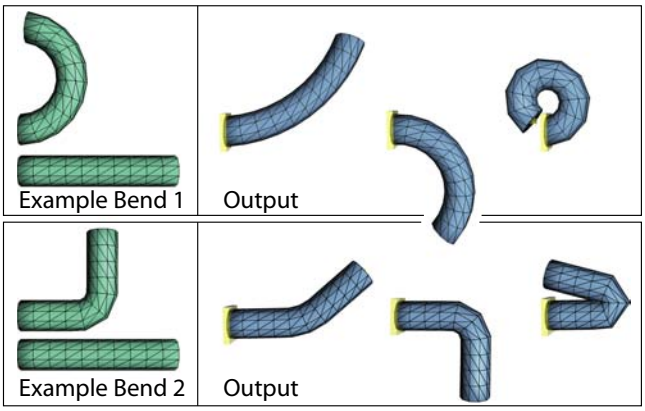

Figure 1: A simple demonstration of MESHIK. Top row: Two examples are given, shown in green in the left column. By fixing one cap in place and manipulating the other end, the bar bends like the examples. Bottom row: If a different example bend is provided, MESHIK generates the new type of bend when the mesh is manipulated.

Our MESHIK algorithm learns the space of meaningful shapes from example meshes. Using the learned space, it generates new shapes that respect the deformations exhibited by the examples, yet still satisfy vertex constraints imposed by the user. Although the user retains complete freedom to precisely specify the position of any vertex, for most tasks, only a few vertices need to be manipulated. MESHIK uses unstructured meshes—triangle meshes with no assumption about connectivity or structure—that can be scanned, hand-sculpted, designed with free-form modeling tools, or computed with arbitrarily complex procedural or simulation methods. As a result, MESHIK provides a tool that simplifies posing tasks even when traditional animation or editing methods do not apply. The animator can pose the object by moving only a few of its vertices or bring it to life by key-framing these vertex positions. Furthermore, the user always retains the freedom to choose the class of meaningful deformations for any mesh, as demonstrated by Figure 1.

MESHIK represents each example with a *feature vector* that describes how the example has deformed relative to a reference mesh.

The *feature space*, defined as a nonlinear span of the example feature vectors, describes the space of appropriate deformations. When the user displaces a few mesh vertices, MESHIK positions the remaining vertices to produce a mesh whose feature vector is as close as possible to the feature space. This ensures that the reconstructed mesh meets the user's constraints exactly while it best reproduces the example deformations.

Our primary contribution is a formulation of mesh-based inverse kinematics that allows meaningful mesh deformations and pose changes to be achieved in an intuitive manner with only a small amount of work by the user. We present an efficient method of nonlinear, multi-way interpolation of unstructured meshes using a deformation-gradient feature space. We demonstrate an efficient optimization technique to search for meshes that meet user constraints and are close to the feature space. Our method allows interactive manipulation of moderately sized meshes with around $10,000$ vertices and 10 examples.

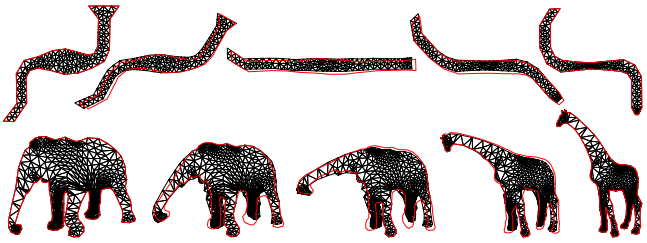

Figure 2: We compare our nonlinear feature-space interpolation scheme with our implementation of the as-rigid-as-possible method for two 2D interpolation sequences. The result of our boundary-based method is displayed as the red line segment while the as-rigid-as-possible interpolation is shown as the black triangulated region. The body of the snake and the trunk of the elephant deform in a similar, locally rigid fashion for both methods. However, our method is numerically much simpler as we only consider the boundary rather than the compatible dissection of the interior.

2 Related Work

Mesh editing allows the user to move a few vertices arbitrarily and employs some numerical objective such as detail preservation or smoothness to place the remaining ones. Subdivision and multi-resolution techniques achieve detail-preserving edits at varying scales with representations that encode mesh details as vertex offsets from topologically [Zorin et al. 1997; Kobbelt et al. 2000] or geometrically [Kobbelt et al. 1998; Guskov et al. 1999] simpler base meshes.

Other editing methods use intrinsic representations such as Laplacian (also called differential) coordinates [Alexa 2003; Lipman et al. 2004; Sorkine et al. 2004] or pyramid coordinates [Sheffer and Kraevoy 2004]. Since each vertex position is encoded by its relationship to its neighbors, local edits made to the intrinsic representation propagate to the surrounding vertices during mesh reconstruction. The editing technique of Yu and colleagues [2004] solves a Poisson equation discretized over the mesh. We use the deformation-gradient representation [Sumner and Popović 2004], which describes affine transformations that individual triangles undergo relative to a reference pose, and discuss this choice in Section 3.1. All of these intrinsic methods have high-level similarities but differ in the details. For example, we also solve a Poisson equation since the normal-equations matrix in our formulation amounts to a form of a Laplacian, and the feature vector to a guidance field.

The differences between inverse kinematics and editing are best illustrated through the typical use of both techniques. Editing sculpts meshes to create new objects, while inverse kinematics manipulates such objects to enliven them. The main implication of this difference is that editing concentrates on an object's shape (how it looks) while inverse kinematics concentrates on an object's deformation (how it moves). In the absence of a convenient numerical objective (e.g., detail preservation, smoothness) that describes how an arbitrary object moves, inverse kinematics on meshes must learn the space of desirable mesh configurations. Such a general approach is not necessary in special cases (e.g., when a skeleton expresses the space of desired configurations), but for cloth, hair, and other soft objects, the general approach of MESHIK, which infers a meaningful space from a series of user-provided examples, is required.

Work has been done in the animation community on the compact representation of sets, or animation sequences, of meshes. Alexa and Müller [2000] compress animation sequences using principal component analysis (PCA). This approximates the set by a linear subspace of mesh space. Similarly, Hauser, Shen, and O'Brien [2003] use modal analysis of linear elastic equations to infer a structure common to all linear elastic materials. Modal analysis uses eigen-analysis of mass and stiffness matrices to extract a small set of basis vectors for the high-energy vibration modes [Pentland and Williams 1989]. However, while both are well-understood and simple to implement, their inherently linear structure makes them inappropriate for describing nonlinear deformations. For example, linear interpolation of rotations will shorten the mesh. Furthermore, PCA works well for compression of existing meshes, but is less appropriate for guiding the search outside the subspace described by the principle components. Hybrid approaches avoid the problems associated with linear interpolation in the special case that the nonlinearities can be expressed in terms of skeletal deformation [Lewis et al. 2000; Sloan et al. 2001]. MESHIK generalizes these approaches with a nonlinear combination of example shapes.

This nonlinear blend can be thought of as an n-way boundary-based version of as-rigid-as-possible shape interpolation [Alexa et al. 2000]. Rather than performing a two-way interpolation based on the compatible dissection of the interior of two shapes, MESHIK interpolates the boundary of n shapes. The practical implication of this reformulation is significant. MESHIK interpolation is faster because it solves for fewer vertices and easier to apply because compatible dissection of n shape interiors is difficult without adding an extremely large number of Steiner vertices. An experimental comparison of the two methods is shown in Figure 2 and demonstrates that, for 2D polygonal shapes, MESHIK interpolation behaves reasonably despite ignoring the interior. The remaining results in the paper and our experience with MESHIK indicate that the same holds for 3D meshes. Concurrent with our work, Xu and colleagues have developed a boundary-based mesh interpolation scheme similar to our nonlinear feature space method [2005]. However, while Xu and colleagues focus on interpolation with prescribed blending weights, our primary contribution is a formulation of mesh-based inverse kinematics that hides these weights from the user behind an intuitive interaction metaphor.

Techniques closest to our approach are those that learn skeleton or mesh configurations from examples. The first such system learns the space of desirable skeleton configurations by linearly blending example motions [Rose et al. 2001]. FaceIK [Zhang et al. 2004] uses a similar approach on meshes to generate new facial expressions by linearly blending the acquired face scans. These linear approaches exhibit the same difficulties as those discussed above. Furthermore, every mesh or skeleton is confined to the linear span of the basis shapes. MESHIK blends nonlinearly *and* does not restrict a mesh to be in the nonlinear span of example shapes. Instead, it favors meshes that are close to, but not necessarily in, this nonlinear space. These design choices, at the cost of slower performance, allow MESHIK to generate compelling meshes even when they differ significantly from the example shapes. When linear blending suffices, the same principle can be used to improve its generalization outside the space explored by examples. Style-based inverse

kinematics [Grochow et al. 2004] describes an alternative nonlinear approach which learns a probabilistic model of desirable skeleton configurations. However, bridging the gap between 60 degrees of freedom in a typical skeleton and 30,000 degrees of freedom in a moderate mesh is the main obstacle to applying this promising technique to meshes.

Ngo and colleagues [2000] introduce configuration modeling as a fundamental problem in computer graphics and present a solution for describing the configuration space of two-dimensional drawings. James and Fatahalian [2003] use a similar approach to precompute numerical simulations for the most common set of control inputs. MESHIK fits in naturally with these configuration models by enhancing the reparameterization map [Ngo et al. 2000], which prescribes how to extrapolate and generalize from such example drawings and precomputed states.

3 Principles of MeshIK

MESHIK uses the example meshes provided by the user to form a space of meaningful deformations. The definition of this space is critical as it must include deformations representative of those exhibited by the examples even far from the given data. The key to designing a good space is to extract, from each example, a vector of features that encodes important shape properties. We use feature vectors that encode the change in shape exhibited by the examples on a triangle-by-triangle basis.

The simplest feature space is just the linear span of the feature vectors of the example poses. Although this space is not what we will ultimately use, we describe it first because it is simple, fast, and may still be valuable in applications where linearity assumptions are sufficient [Blanz and Vetter 1999; Ngo et al. 2000] or where artifacts can be avoided by dense sampling [Bregler et al. 2002]. Our more powerful nonlinear span is required in the general case when the natural interpolation of the example deformations is not linear (e.g., for rotations).

An edited mesh can be reconstructed from a feature vector by solving a least squares problem for the free vertices while enforcing constraints for each vertex that the user has positioned. Because the feature vector is an intrinsic representation of the mesh geometry, the error incurred by fixing some vertices as constraints will propagate across the mesh, rather than being concentrated at the constrained vertices. Our algorithm couples the constrained reconstruction process with a search within feature space so that it finds the position in feature space that has the minimal reconstruction error.

3.1 Feature Vectors

An obvious and explicit way to represent the geometry of a triangle mesh is with the coordinates of its vertices in the global frame. However, this representation, while simple and direct, is a poor choice for any mesh editing operation as the coordinates in the global frame do not capture the local shape properties and relationships between vertices [Sorkine et al. 2004].

For manipulating meshes, it is more useful to describe a mesh as a vector in a different feature space based on properties of the mesh. The components of the feature vector relate the geometry of nearby vertices and capture the short-range correlations present in the mesh. MESHIK uses deformation gradients or, as Barr [1984] refers to them, local deformations, as the feature vector. Deformation gradients describe the transformation each triangle undergoes relative to a reference pose. They were used by Sumner and Popović [2004] to transfer deformation from one mesh to another and are similar to the representation used by Yu et al. [2004].

Deformation Gradient Given a reference mesh P_0 and a deformed mesh P, each containing n vertices and m triangles in the same connectivity structure, we would like to compute the feature vector $\mathbf{f}$ corresponding to P. A *deformation gradient* of a triangle of P is the Jacobian of the affine mapping of the vertices of the triangle from their positions in P_0 to their positions in P. Since the positions of the triangle's vertices in P_0 and P do not uniquely define an affine mapping in $\mathbb{R}^3$, we add to each triangle a fourth vertex, as proposed by Sumner and Popović [2004]. This strategy ensures that the affine map scales the direction perpendicular to the triangle in proportion to the length of the edges. For simplicity, when we discuss matrix dimensions in terms of the variable n, we mean for n to include these added vertices.

Denote by Φ_j the affine mapping of the j-th triangle that operates on a point $\mathbf{p} \in \mathbb{R}^3$ as follows:

$$\Phi_j(\mathbf{p}) = \mathbf{T}_j\mathbf{p} + \mathbf{t}_j.$$

The 3×3 matrix $\mathbf{T}_j$ contains the rotation, scaling, and skewing components, and the vector $\mathbf{t}_j$ defines the translation component of the affine transformation.

The deformation gradient is the Jacobian matrix $\mathrm{D}_{\mathbf{p}}\Phi_j(\mathbf{p}) = \mathbf{T}_j$, which is computed from the positions of the four vertices $\{\bar{\mathbf{v}}_k^j\}$ and $\{\mathbf{v}_k^j\}$, $(1 \le k \le 4)$, in P_0 and P respectively:

$$\mathbf{T}_j = \begin{bmatrix} \mathbf{v}_1^j - \mathbf{v}_4^j & \mathbf{v}_2^j - \mathbf{v}_4^j & \mathbf{v}_3^j - \mathbf{v}_4^j \end{bmatrix} \cdot \begin{bmatrix} \bar{\mathbf{v}}_1^j - \bar{\mathbf{v}}_4^j & \bar{\mathbf{v}}_2^j - \bar{\mathbf{v}}_4^j & \bar{\mathbf{v}}_3^j - \bar{\mathbf{v}}_4^j \end{bmatrix}^{-1}. \quad (1)$$

Transformation $\mathbf{T}_j$ is linear in the vertices $\{\mathbf{v}_k^j\}$. Thus, assuming the vertices of the reference pose are fixed, the linear operator $\mathbf{G}$ extracts a feature vector from the deformed mesh P:

$$\mathbf{f} = \mathbf{G}\mathbf{x}. \quad (2)$$

The vector $\mathbf{x} = (x_1, .., x_n, y_1, .., y_n, z_1, .., z_n) \in \mathbb{R}^{3n}$ stacks the coordinates of mesh vertices $\{\mathbf{v}_i\}_{i=1}^n$ of P. The coefficients of $\mathbf{G}$ depend only on the vertices of the reference mesh P_0 and come from the inverted matrix in Eq. (1). The matrix $\mathbf{G}$ is built such that the feature vector $\mathbf{f} \in \mathbb{R}^{9m}$ that results from the multiplication $\mathbf{G}\mathbf{x}$ will be the unrolled and concatenated elements of the deformation gradients $\mathbf{T}_j$ for all m triangles. Hence, the expression $\mathbf{G}\mathbf{x}$ is equivalent to evaluating Eq. (1) for each triangle and packaging the resulting 3×3 matrices for each of the m triangles into one tall vector.

The linear operator $\mathbf{G}$ is a $9m \times 3n$ matrix. But, because the computation is separable in the three spatial coordinates, $\mathbf{G}$ has a simple block-diagonal structure:

$$\mathbf{G} = \begin{bmatrix} G & & \\ & G & \\ & & G \end{bmatrix}.$$

Each block G is a sparse $3m \times n$ matrix with four nonzero entries in each row corresponding to the four vertices referenced by each application of Eq. (1).

Mapping a feature vector $\mathbf{f}$ of some mesh P back to its global representation $\mathbf{x}$ involves solving Eq. (2) for $\mathbf{x}$, or inverting $\mathbf{G}$. But, because our feature vectors are invariant to global translations of the mesh, one vertex position must be held constant to make the solution unique. This results in the following least squares problem:

$$\mathbf{x} = \arg\min_{\mathbf{x}} \|\tilde{\mathbf{G}}\mathbf{x} - (\mathbf{f} + \mathbf{c})\|. \quad (3)$$

The modified operator $\tilde{\mathbf{G}}$ is void of the three columns that multiply the fixed vertex, and the constant vector $\mathbf{c}$ contains the result of this multiplication. In fact, the least-squares inversion in Eq. (3) may be applied when constraining an arbitrary number of vertices. Additional vertex constraints only affect $\tilde{\mathbf{G}}$ and $\mathbf{c}$. In what follows, however, we drop the distinction between $\tilde{\mathbf{G}}$ and $\mathbf{G}$ for notational simplicity.

This inversion provides a general method for editing unstructured triangle meshes by constraining a subset of vertices to desired locations and inverting a feature vector to compute the edited shape. For example, we can retrieve the reference mesh with an identity feature $\mathbf{f}_{\text{id}}$ of m unrolled and concatenated 3×3 identity matrices. More generally, Yu and colleagues [2004] describe an algorithm to set the features by propagating the transformation of a handle curve across the mesh. Similarly, Sumner and Popović [2004] enable deformation transfer between source and target meshes by reconstructing the target with feature vectors extracted from the source.

Alternatives We choose deformation gradients over alternatives because they are a linear function of the mesh vertices and lead to a natural decomposition into rotations and scale/shears which facilitates our nonlinear interpolation. Pyramid coordinates [Sheffer and Kraevoy 2004] provide a promising representation and editing framework, but the required nonlinear reconstruction step is too slow for our interactive application. Laplacian coordinates [Lipman et al. 2004] are a valid alternative since they are linear in the mesh vertices and efficiently inverted. However, an interpolation scheme for Laplacian coordinates that generates natural-looking results is needed. The encoding scheme used by the Poisson mesh editing technique [Yu et al. 2004] is an alternative to ours that may perform well for our problem since it was successfully used for mesh interpolation [Xu et al. 2005]. These issues indicate that the design of a more compact and efficient feature space is an area of future work.

3.2 Linear Feature Space

A feature space defines the space of desirable deformations. The simplest feature space is the linear span of the features extracted from the example meshes. A member $\mathbf{f}_{\mathbf{w}}$ of this space is parameterized by the coefficients in the vector $\mathbf{w}$:

$$\mathbf{f}_{\mathbf{w}} = \mathbf{M}\mathbf{w},$$

where $\mathbf{M}$ is a matrix whose columns are the feature vectors $\mathbf{d}_i$ corresponding to the example meshes ($1 \le i \le l$).

In practice, our algorithm computes the mean $\bar{\mathbf{d}}$ and uses the mean centered feature vectors $\{\bar{\mathbf{d}}_i\}$:

$$\mathbf{M}\mathbf{w} = \bar{\mathbf{d}} + \sum_{i=1}^{l-1} w_i \bar{\mathbf{d}}_i.$$

Note that the linear dependence introduced by mean centering implies using $l-1$ example features and weights instead of the l features and weights used in the non-centered linear combination.

Given only a few specified vertex positions, linear MESHIK computes the pose $\mathbf{x}^*$ whose features are most similar to the closest point $\mathbf{M}\mathbf{w}^*$ in the linear feature space:

$$\mathbf{x}^*, \mathbf{w}^* = \arg\min_{\mathbf{x},\mathbf{w}} \|\mathbf{G}\mathbf{x} - (\mathbf{M}\mathbf{w} + \mathbf{c})\|. \tag{4}$$

Recall that $\mathbf{G}$ and $\mathbf{c}$ are built such that the minimization will satisfy the positional constraints on the specified vertices. This equation replaces the feature vector in Eq. (3) with a linear combination of example features $\mathbf{M}\mathbf{w}$. Because the linear space extrapolates poorly, this metric can be further augmented to penalize solutions that are far from the example meshes:

$$\arg\min_{\mathbf{w},\mathbf{x}} \|\mathbf{G}\mathbf{x} - (\mathbf{M}\mathbf{w} + \mathbf{c})\| + k\|\mathbf{w}\|. \tag{5}$$

The second term $k\|\mathbf{w}\|$ favors examples close to the mean $\bar{\mathbf{d}}$ by penalizing large weights.

The value of k, which weights the penalty term, can be chosen in a principled fashion by considering the Bayesian interpretation of our linear model: It maximizes the likelihood of the parameter vector $\mathbf{w}$ with respect to the example poses. Accordingly, our method can be improved by compressing the matrix $\mathbf{M}$ using its principal components and selecting an appropriate value for the weighting parameter k as a function of the variance lost during PCA [Tipping and Bishop 1999].

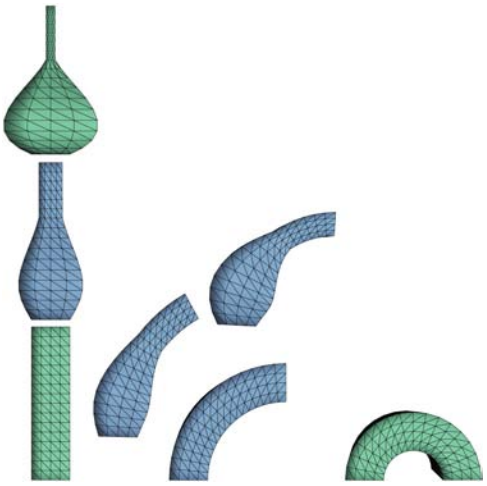

Figure 3: Our nonlinear feature space is used to perform a three-way blend between the green meshes, producing the blue ones.

An alternative to this linear Gaussian model is a nonlinear Gaussian-process–latent-variable model [Grochow et al. 2004] in which each component of the feature vector is an independent Gaussian process. This implies that one should carefully parameterize the feature space to match this independence assumption. For skeletons, exponential maps or Euler angles accomplish this task but introduce a nonlinear mapping between the independent parameters and the user-specified handles. Applying a similar strategy on meshes will also produce nonlinear constraints and make it difficult to solve for thousands of vertices interactively.

3.3 Nonlinear Feature Space

Since the MESHIK feature vectors are linear transformations of pose geometry vectors, linear blending of feature vectors amounts to naïve linear blending of poses, which is well known to result in unnatural effects if the blended poses have undergone rotation. In our setting, this means that linear blending will suffice only if the example set is dense enough that large rotations are not present. However, dense sampling is not the typical case and *generalizations* of the examples beyond small deformations are not possible. To avoid artifacts due to large rotations, which are typical in most non-trivial settings, we require a "span" of the example features which combines rotations in a more natural way. Our approach is based on polar decomposition [Shoemake and Duff 1992] and the matrix exponential map. Figure 3 demonstrates our nonlinear feature space used to interpolate between three different meshes. By setting the weights directly, rather than solving for them within the IK framework, the nonlinear feature space can create multi-way blends.

First, we decompose the deformation gradient $\mathbf{T}_{ij}$ for the j-th triangle ($1 \le j \le m$) in the i-th pose ($1 \le i \le l$) into rotational and scale/shear components using polar factorization:

$$\mathbf{T}_{ij} = \mathbf{R}_{ij}\mathbf{S}_{ij}.$$

We then use the exponential map to combine the individual rotations of the different poses. The scale and shear part can be combined linearly without further treatment.

We implement the exponential map using the matrix exponential and logarithm functions [Murray et al. 1994]. These provide a mapping between the group of 3D rotations $\mathbf{SO}(3)$ and the Lie algebra

$so(3)$ of skew symmetric 3×3 matrices. A practical approach to interpolating rotations is to map them to $so(3)$ using the matrix logarithm, interpolate linearly in $so(3)$, and map back to $\mathbf{SO}(3)$ using the matrix exponential [Murray et al. 1994; Alexa 2002]. This leads to the following expression for the nonlinear span of the deformation gradient of the j-th triangle:

$$\mathbf{T}_j(\mathbf{w}) = \exp\left(\sum_{i=1}^{l} w_i \log\left(\mathbf{R}_{ij}\right)\right) \cdot \sum_{i=1}^{l} w_i \mathbf{S}_{ij}. \tag{6}$$

The matrix exponential and logarithm are evaluated efficiently using Rodrigues' formula [Murray et al. 1994].[1] We also experimented with exponential and logarithm functions for general matrices [Alexa 2002] which do not require factorization into rotations and scales. However, the singularities of this approach prevented a stable solution of our minimization problem.

We chose to use the matrix exponential and logarithm because we can easily take derivatives of the resulting nonlinear model with respect to $\mathbf{w}$. For later use in Section 4.1, we note that the partial derivatives of $\mathbf{T}_j(\mathbf{w})$ are given by

$$\begin{aligned} \mathrm{D}_{w_k}\mathbf{T}_j(\mathbf{w}) &= \exp\left(\sum_{i=1}^{l} w_i \log\left(\mathbf{R}_{ij}\right)\right) \log\left(\mathbf{R}_{kj}\right) \sum_{i=1}^{l} w_i \mathbf{S}_{ij} \\ &+ \exp\left(\sum_{i=1}^{l} w_i \log\left(\mathbf{R}_{ij}\right)\right) \mathbf{S}_{kj}. \end{aligned} \tag{7}$$

4 Numerics

In this section we show how to solve the following nonlinear analog of the linear inversion in Eq. (4):

$$\mathbf{x}^*, \mathbf{w}^* = \arg\min_{\mathbf{x},\mathbf{w}} \|\mathbf{G}\mathbf{x} - (\mathbf{M}(\mathbf{w}) + \mathbf{c})\|, \tag{8}$$

where $\mathbf{M}$ is now a function that combines the feature vectors nonlinearly according to Eq. (6). This is a nonlinear least-squares problem which can be solved using the iterative Gauss-Newton algorithm [Madsen et al. 2004]. At each iteration, a linear least-squares system is solved which involves solving the normal equations by Cholesky decomposition and back-substitution. We now elaborate on the key stages of this procedure.

4.1 Gauss-Newton Algorithm

In MESHIK, the Gauss-Newton algorithm linearizes the nonlinear function of the feature weights which defines the feature space:

$$\mathbf{M}(\mathbf{w} + \boldsymbol{\delta}) = \mathbf{M}(\mathbf{w}) + \mathrm{D}_{\mathbf{w}}\mathbf{M}(\mathbf{w})\boldsymbol{\delta}.$$

Then, each Gauss-Newton iteration solves a linearized problem to improve $\mathbf{x}_k$ and $\mathbf{w}_k$—the estimates of the vertex positions and the weight vector at the k-th iteration:

$$\begin{aligned} \boldsymbol{\delta}_k, \mathbf{x}_{k+1} &= \arg\min_{\boldsymbol{\delta},\mathbf{x}} \|\mathbf{G}\mathbf{x} - \mathrm{D}_{\mathbf{w}}\mathbf{M}(\mathbf{w}_k)\boldsymbol{\delta} - (\mathbf{M}(\mathbf{w}_k) + \mathbf{c})\| \\ \mathbf{w}_{k+1} &= \mathbf{w}_k + \boldsymbol{\delta}_k. \end{aligned} \tag{9}$$

The process repeats until convergence, which we detect by monitoring the change in the objective function $f_k = f(\mathbf{w}_k)$, the gradient of the objective function, and the magnitude of the update vector $\boldsymbol{\delta}_k$ [Gill et al. 1989]:

$$\begin{aligned} \|f_k - f_{k-1}\|_\infty &< \varepsilon(1 + f_k) \\ \|\mathrm{D}_{\mathbf{w}} f(\mathbf{w})\|_\infty &< \sqrt[3]{\varepsilon}(1 + f_k) \\ \|\boldsymbol{\delta}_k\|_\infty &< \sqrt[2]{\varepsilon}(1 + \|\mathbf{w}_k\|_\infty). \end{aligned}$$

In our experiments, the iteration converges after about six iterations with $\varepsilon = 1.0 \times 10^{-6}$.

Solving the linear least-squares problem in Eq. (9) leads to a system of normal equations:

$$\mathbf{A}^\top \mathbf{A} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\delta} \end{bmatrix} = \mathbf{A}^\top \left(\mathbf{M}(\mathbf{w}_k) + \mathbf{c}\right), \tag{10}$$

where $\mathbf{A}$ is a sparse matrix of size $9m \times (3n + l)$ of the form

$$\mathbf{A} = \left[\begin{array}{ccc|c} G & & & -J_1 \\ & G & & -J_2 \\ & & G & -J_3 \end{array}\right].$$

Recall that G is also a very sparse matrix, having only four entries per row. As we will see in Section 4.2, this permits efficient numerical solution of the system despite its size. The three blocks J_i are the blocks of the Jacobian matrix $\mathrm{D}_{\mathbf{w}}\mathbf{M}(\mathbf{w})$ partitioned according to the three vertex coordinates.

4.2 Cholesky Factorization

Without a special purpose solver, the normal equations in Eq. (10) in each Gauss-Newton iteration can take a minute or longer to solve. This is much too slow for an interactive system, which, in our experience, requires at least two solutions for every second of interaction. The key to accelerating the solver is to reuse computations between iterations. A direct solution with a general purpose method (e.g., Cholesky or QR factorization [Golub and Loan 1996]) will not be able to reuse the factorization from the previous iteration because $\mathbf{A}$ continually changes. And, despite the very sparse matrix $\mathbf{A}^\top\mathbf{A}$, conjugate gradient converges too slowly even with a variety of preconditioners.

Our solution uses a direct method with specialized Cholesky factorization. We exploit the block structure of the system matrix:

$$\mathbf{A}^\top\mathbf{A} = \begin{bmatrix} G^\top G & & & -G^\top J_1 \\ & G^\top G & & -G^\top J_2 \\ & & G^\top G & -G^\top J_3 \\ -J_1^\top G & -J_2^\top G & -J_3^\top G & \sum_{i=1}^{3} J_i^\top J_i \end{bmatrix}. \tag{11}$$

The three $G^\top G$ blocks, each sparse $n \times n$ matrices, are constant throughout the iterations. If these blocks are pre-factored, the remaining portion of the Cholesky factorization may be computed efficiently.

First, symbolic Cholesky factorization $\mathbf{U}^\top\mathbf{U} = \mathbf{A}^\top\mathbf{A}$ reveals the block structure of the upper-triangular Cholesky factor:

$$\mathbf{U} = \begin{bmatrix} R & & & -R_1 \\ & R & & -R_2 \\ & & R & -R_3 \\ & & & R_s \end{bmatrix}$$

where

$$R^\top R = G^\top G.$$

We precompute R by sparse Cholesky factorization [Toledo 2003] after re-ordering the columns to reduce the number of additional non-zero entries [Karypis and Kumar 1999].

[1] Note that the matrix logarithm is a multi-valued function: each rotation in $\mathbf{SO}(3)$ has infinitely many representations in $so(3)$. In some cases, interpolation may require equivalent rotations in a different range which can be computed by adding multiples of 2π. However, our implementation of the matrix logarithm always returns rotation angles between $-\pi$ and π.

The only equations that remain to be solved in every iteration (to compute the remaining blocks of $\mathbf{U}$) are:

$$R^\top R_i = G^\top J_i, \qquad 1 \le i \le 3 \tag{12}$$

$$R_s^\top R_s = \sum_{i=1}^{3} J_i^\top J_i - R_i^\top R_i. \tag{13}$$

In Eq. (12), backsubstitution with the precomputed R computes the blocks R_1, R_2, R_3 by solving three linear systems. These blocks are in turn used on the right-hand side of Eq. (13) to compute the $l \times l$ matrix whose dense Cholesky factorization yields the last block R_s. For a large number of examples, this factorization step will eventually become the bottleneck. In our experiments, however, with l=20 or fewer examples, the solution of Eq. (12) for three dense $n \times l$ blocks and their use in the computation of $R_i^\top R_i$ dominates the cost.

5 Experimental Results

We have implemented MESHIK both as an interactive mesh manipulation system as well as in an offline application that uses key-framed constraints to solve for mesh poses over time. In our interactive system, the user can select groups of vertices that become "handles" which can be interactively positioned. As the handles are moved the rest of the mesh is automatically deformed.

Figure 5 demonstrates the power of MESHIK. Given a cylindrical bar in two poses (5 A), one straight, and one smoothly bent, the user constrains the left cap to stay in place and manipulates one vertex on the right cap. Using the nonlinear feature space, our system is able to generalize to any other bend of the bar in the same plane (5 B). In contrast, the linear feature space (5 C) interpolates the two examples poorly (the tip of the bar collapses in between the examples) and extrapolates even more poorly. If the end of the bar is dragged perpendicular to the example bend (5 D), it deforms differently since no example has demonstrated how to deform in this direction. Given an additional example, the bar can bend in that plane (5 E) as well as the space in between (5 F). In Figure 1, we show that by supplying a different example, the bar bends differently. Thus, MESHIK does not prescribe one type of deformation but instead derives the appropriate class of deformations from the examples.

In Figure 6 we demonstrate how MESHIK can be used to pose a character. Ten example poses, shown in green in the top row, were used for this demonstration. Two handle vertices are selected as constraints on the front and back foot of the reference pose (6 A). By dragging the front foot forward, the lion bends its front legs at the hip and stretches its body forward. The position of the lion's paw can be precisely controlled by the user. In (6 B) the paw has been pulled farther forward than its position in any example. The body of the lion deforms realistically to meet the constraints so that there is no discernible distortion. In order to pose only the front right leg and keep the rest of the body fixed (6 C), we select the unwanted region (shown in red) and remove it from the objective function by building a feature space that ignores the deformation gradients of the selected triangles. This region remains fixed in place, but does not contribute to the error as the optimal weights are computed. This allows the user to pose the front leg independent of the rest of the body. After performing the same operation for the tail (6 D), the user has achieved a novel pose different from all those shown in the example set.

Figure 7 demonstrates how MESHIK can pose a mesh whose deformations have no obvious skeletal representation: a flag blowing in the wind. The input for this demonstration is fourteen flag examples from a dynamic simulation, shown in the top row. Starting with an undeformed flag (7 A), the user arbitrarily positions the four corners of the flag (7 B–D). The interior deforms in a cloth-like fashion. By key-framing the position of the constraints over time, we can even create an animation of a walking flag (7 E–F).

Mesh	Verts	Tris	Ex	Factor	Solve	Total
Bar	132	260	2	0.000	0.000	0.015
Flag	516	932	14	0.016	0.015	0.020
Lion	5,000	9,996	10	0.475	0.150	0.210
Horse	8,425	16,846	4	0.610	0.105	0.160
Elephant	42,321	84,638	4	13.249	0.620	0.906

Table 1: Number of vertices, triangles, and example meshes as well as timing data for the demonstrated results.

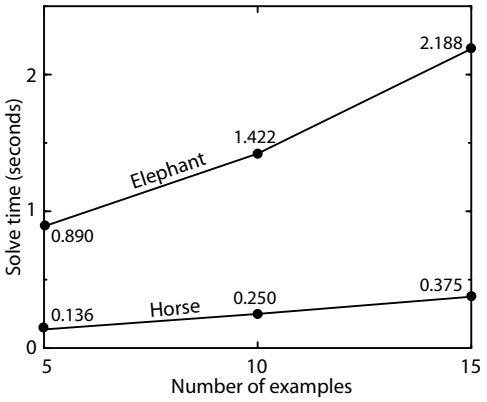

Figure 4: Solve time as a function of the number of examples for the horse and elephant meshes.

Figure 8 shows our system used to produce a galloping animation. Four example poses of a horse were used as input, and one vertex on each foot of the horse was key-framed to follow a gallop gait. The positions of the remaining vertices of the horse were chosen by our system for each frame, resulting in a galloping animation. If we replace the four horse poses with those of an elephant and use the same key-framed foot positions, we compute a galloping elephant.

When generating animations with our offline application, temporal coherence is important. Since our deformation system is nonlinear, a small change in the constraints may result in a large change in the resulting deformation. In order to achieve temporal coherence, we add the additional term $p||\mathbf{w} - \mathbf{w}_0||$ to the objective function in Eq. (8). This encourages the new blending weights $\mathbf{w}$ to be similar to the ones from the previous frame of animation $\mathbf{w}_0$. We used a value of 100 for the factor p in all animations.

The conference DVD-ROM contains live recordings of interactive editing sessions with the bar example from Figure 5 and the lion from Figure 6, as well as the flag animation from Figure 7 and the horse and elephant animations from Figure 8.

Table 1 gives statistics about the meshes used in our results including the number of vertices, the number of triangles, the number of examples, and the running times. The timing was measured on a 3.4 GHz Pentium 4 PC with 2GB of RAM. The "factor" column indicates the time required to compute the Cholesky factorization of $G^\top G$. This computation is a preprocess as the factorization does not change for a particular choice of handle vertices. The "solve" column indicates the time required to perform one iteration of the Gauss-Newton algorithm described in Section 4. After each iteration, the user-interface is updated and new positions for the constrained handle vertices are queried by the solver. This allows our system to remain interactive during the nonlinear optimization. The "total" column includes the solve time plus additional unoptimized bookkeeping that is performed during each iteration. Figure 4 graphs the solve time as a function of the number of examples for the horse and elephant meshes.

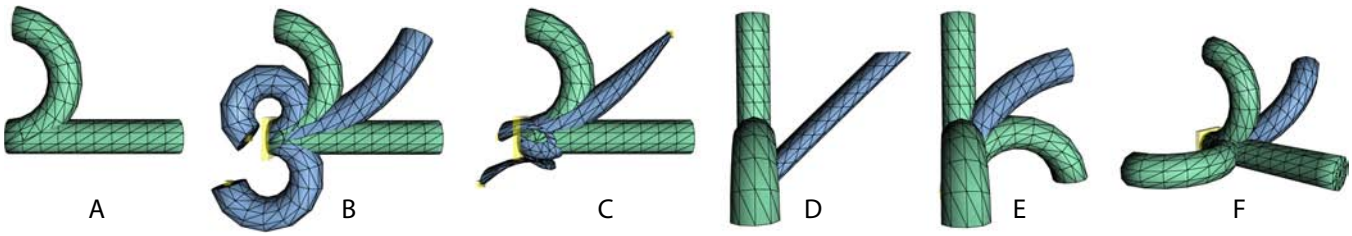

Figure 5: Using MESHIK to pose a bar: (A) Two example poses superimposed on top of each other. (B) The left cap of the unbent bar is constrained to stay in place while a single vertex on the right side is manipulated. Three edits using our nonlinear feature space are shown. Note that MESHIK generalizes beyond the two examples and can create arbitrary bends in the plane. (C) In contrast, the linear feature space interpolates and generalizes poorly. (D) In this top down view, moving the constrained vertex perpendicular to the bend causes a shear since no examples were provided in this direction. (E)–(F) Providing one additional example in the perpendicular direction allows MESHIK to generalize to bends in that direction as well as in the space in between.

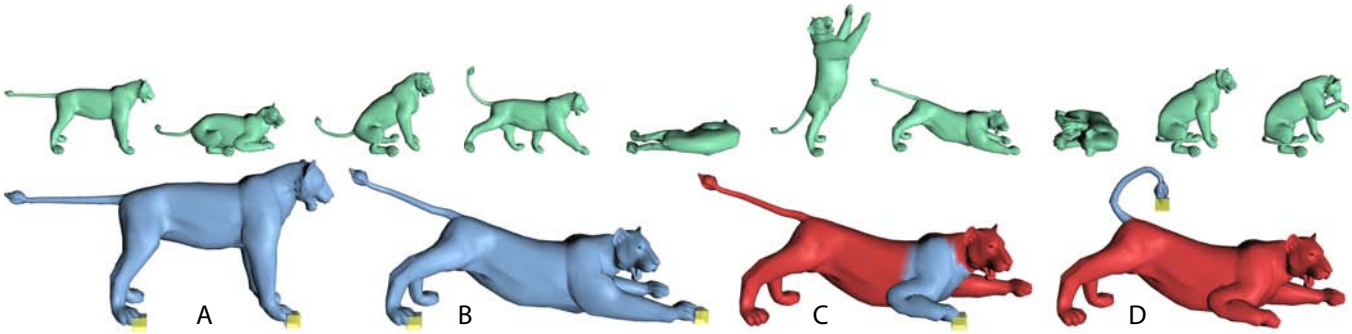

Figure 6: Top row: Ten lion example poses. Bottom row: A sequence of posing operations. (A) Two handle vertices are chosen. (B) The front leg is pulled forward and the lion continuously deforms as the constraint is moved. (C) The red region is selected and frozen so that the front leg can be edited in isolation. (D) A similar operation is performed to adjust the tail. The final pose is different from any individual example.

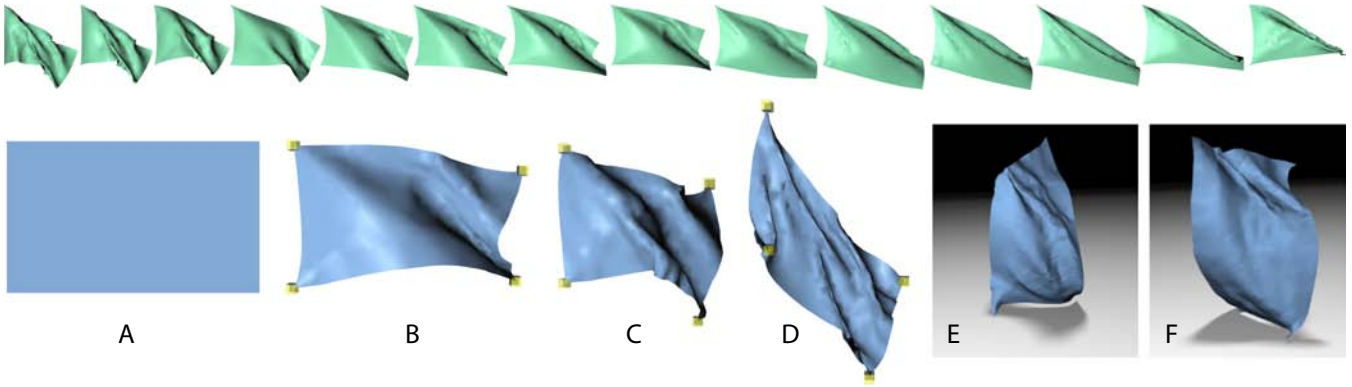

Figure 7: Posing simulated flag. Top row: Fourteen examples of a flag blowing in the wind created with a cloth simulation. (A) An undeformed flag is used as the reference pose. (B)–(D) By positioning only the corners of the flag, we create realistic cloth deformations without requiring any dynamic simulation. (E)–(F) Two frames from an animation in which the constraints on the corners were key-framed to produce a walking motion.

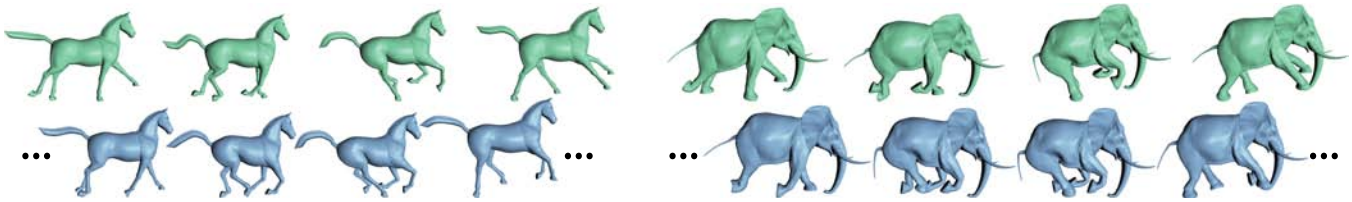

Figure 8: Galloping horse and elephant animations were created using only four examples of each along with the same key-framed motion of one vertex on each foot.

6 Conclusion

Intuitive manipulation of meshes is a fundamental technique in modeling and animation: modelers use it to edit shapes and animators use it to pose them. MESHIK is an easy-to-use manipulation tool that adapts to each task by learning from example meshes. It provides a direct interface for specifying the shape by allowing the user to select and adjust any subset of vertices. It relieves the user from having to adjust every vertex by extrapolating from examples to position the remaining vertices automatically.

Current limitations of our method direct us to areas of future work. The time required to solve the nonlinear optimization limits its interactive manipulation to meshes with around 10,000 vertices and 10 examples. Different mesh representations, such as subdivision surfaces or multiresolution hierarchies, may allow a more efficient formulation of MESHIK for complex objects. Experimentation with other feature vectors and numerical methods for their inversion may also yield improvement. Different feature vectors may capture the essential shape properties more compactly or yield a different inversion process with a more efficient numerical solution.

MESHIK describes a feature space with a nonlinear blend of all example shapes. This choice is effective for a small number of examples, but describing complex mesh configurations may require using many example shapes. Although this decreases the interactivity of our present system, new representations of the feature space could be designed when examples are plentiful. The linear feature space we describe is a possible starting point for such examination.

Perhaps the most exciting extension of our system would capture dynamic effects such as inertia and follow-through. Our only experiment in this arena was the simple extension of the objective function to encourage small weight changes between successive animation frames. A comprehensive treatment would introduce dynamic features and a mechanism for matching both static and dynamic feature vectors. Such a system might ultimately provide a practical compromise between the automation offered by physical simulations and the control provided by key-framing techniques.

7 Acknowledgments

We are grateful to Sivan Toledo for showing us the Cholesky factorization technique described in Section 4.2. The work of Craig Gotsman was partially supported by Israel Ministry of Science grant 01-01-01509 and European FP6 NoE grant 506766 (AIM@SHAPE).

References

ALEXA, M., AND MÜLLER, W. 2000. Representing animations by principal components. *Computer Graphics Forum 19*, 3, 411–418.

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 157–164.

ALEXA, M. 2002. Linear combination of transformations. *ACM Transactions on Graphics 21*, 3 (July), 380–387.

ALEXA, M. 2003. Differential coordinates for mesh morphing and deformation. *The Visual Computer 19*, 2, 105–114.

BARR, A. H. 1984. Global and local deformations of solid primitives. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, vol. 18, 21–30.

BLANZ, V., AND VETTER, T. 1999. A morphable model for the synthesis of 3d faces. In *Proceedings of ACM SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 187–194.

BREGLER, C., LOEB, L., CHUANG, E., AND DESHPANDE, H. 2002. Turning to the masters: Motion capturing cartoons. *ACM Transactions on Graphics 21*, 3 (July), 399–407.

GILL, P. E., MURRAY, W., AND WRIGHT, M. H. 1989. *Practical Optimization*. Academic Press, London.

GOLUB, G. H., AND LOAN, C. F. V. 1996. *Matrix Computations*, third ed. Johns Hopkins University Press, Baltimore, Maryland.

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

GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution signal processing for meshes. In *Proceedings of ACM SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 325–334.

HAUSER, K. K., SHEN, C., AND O'BRIEN, J. F. 2003. Interactive deformation using modal analysis with constraints. In *Proceedings of Graphics Interface 2003*, 247–256.

JAMES, D. L., AND FATAHALIAN, K. 2003. Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics 22*, 3 (July), 879–887.

KARYPIS, G., AND KUMAR, V. 1999. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing 20*, 1. http://www.cs.umn.edu/∼metis.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of ACM SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, 105–114.

KOBBELT, L. P., BAREUTHER, T., AND SEIDEL, H.-P. 2000. Multiresolution shape deformations for meshes with dynamic vertex connectivity. *Computer Graphics Forum 19*, 3 (Aug.), 249–260.

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

LIPMAN, Y., SORKINE, O., COHEN-OR, D., LEVIN, D., RÖSSL, C., AND SEIDEL, H.-P. 2004. Differential coordinates for interactive mesh editing. In *Proceedings of Shape Modeling International*, 181–190.

MADSEN, K., NIELSEN, H., AND TINGLEFF, O. 2004. Methods for nonlinear least squares problems. Tech. rep., Informatics and Mathematical Modelling, Technical University of Denmark.

MURRAY, R. M., LI, Z., AND SASTRY, S. S. 1994. *A mathematical introduction to robotic manipulation*. CRC Press.

NGO, T., CUTRELL, D., DANA, J., DONALD, B., LOEB, L., AND ZHU, S. 2000. Accessible animation and customizable graphics via simplicial configuration modeling. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 403–410.

PENTLAND, A., AND WILLIAMS, J. 1989. Good vibrations: Modal dynamics for graphics and animation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 89)*, vol. 23, 215–222.

ROSE, C. F., SLOAN, P.-P. J., AND COHEN, M. F. 2001. Artist-directed inverse-kinematics using radial basis function interpolation. *Computer Graphics Forum 20*, 3, 239–250.

SHEFFER, A., AND KRAEVOY, V. 2004. Pyramid coordinates for morphing and deformation. In *Proceedings of the 2nd Symposium on 3D Processing, Visualization and Transmission*, 68–75.

SHOEMAKE, K., AND DUFF, T. 1992. Matrix animation and polar decomposition. In *Proceedings of Graphics Interface 92*, 259–264.

SLOAN, P.-P. J., III, C. F. R., AND COHEN, M. F. 2001. Shape by example. In *2001 ACM Symposium on Interactive 3D Graphics*, 135–144.

SORKINE, O., LIPMAN, Y., COHEN-OR, D., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 179–188.

SUMNER, R. W., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Transactions on Graphics 23*, 3 (Aug.), 399–405.

TIPPING, M. E., AND BISHOP, C. M. 1999. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B 61*, 3, 611–622.

TOLEDO, S., 2003. TAUCS: A library of sparse linear solvers, version 2.2. http://www.tau.ac.il/∼stoledo/taucs.

XU, D., ZHANG, H., WANG, Q., AND BAO, H. 2005. Poisson shape interpolation. In *Proceedings of ACM Symposium on Solid and Physical Modeling*.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics 23*, 3 (Aug.), 644–651.

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

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *Proceedings of ACM SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 259–268.