Data-driven curvature for real-time line drawing of dynamic scenes

Evangelos Kalogerakis, Derek Nowrouzezahrai, Patricio Simari,
James McCrae, Aaron Hertzmann, Karan Singh
University of Toronto

This paper presents a method for real-time line drawing of deforming objects. Object-space line drawing algorithms for many types of curves, including suggestive contours, highlights, ridges and valleys, rely on surface curvature and curvature derivatives. Unfortunately, these curvatures and their derivatives cannot be computed in real-time for animated, deforming objects. In a preprocessing step, our method learns the mapping from a low-dimensional set of animation parameters (e.g., joint angles) to surface curvatures for a deforming 3D mesh. The learned model can then accurately and efficiently predict curvatures and their derivatives, enabling real-time object-space rendering of suggestive contours and other such curves. This represents an order-of-magnitude speed-up over the fastest existing algorithm capable of estimating curvatures and their derivatives accurately enough for many different types of line drawings. The learned model can generalize to novel animation sequences, and is also very compact, typically requiring a few megabytes of storage at run-time. We demonstrate our method for various types of animated objects, including skeleton-based characters, cloth simulation and blend-shape facial animation, using a variety of non-photorealistic rendering styles.

An important component of our system is the use of dimensionality reduction for differential mesh data. We show that Independent Component Analysis (ICA) yields localized basis functions, and gives superior generalization performance to that of Principal Component Analysis (PCA).

Categories and Subject Descriptors: I.3.3 [**Picture/Image Generation**]: Line and curve generation; I.3.5 [**Computational Geometry and Object Modeling**]: Geometric algorithms, languages, and systems

General Terms: Algorithms, Design

Additional Key Words and Phrases: real-time curvature, real-time line drawing, real-time non-photorealistic rendering for deforming objects, data-driven curvature, Independent Component Analysis (ICA), Neural Network Regression

Authors' emails: {kalo, derek, psimari, mccrae, hertzman, karan}@dgp.toronto.edu
Authors' address: Department of Computer Science, University of Toronto, 10 King's College Road, Room 3302 Toronto, Ontario, Canada M5S 3G4

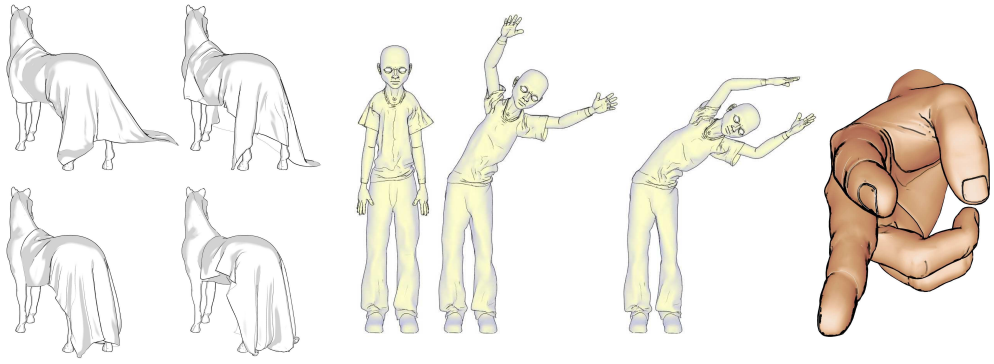

Fig. 1: *Line drawings of deforming 3D objects, generated in real-time (24 to 80 FPS) by our system.*

1. INTRODUCTION

Computer-generated 3D line drawing has been shown to produce clear and compelling imagery. These algorithms are based on a variety of curves defined on 3D surfaces, such as contours and suggestive contours [DeCarlo et al. 2003]. Some of these curves require surface curvature and curvature derivatives to be computed everywhere on the surface; important examples include suggestive contours, ridges, valleys, apparent ridges, principal highlights, and highlight lines. These curves are essential components of high-quality line drawing of smooth surfaces. For example, drawings using both contours and suggestive contours generally look dramatically better than with contours alone.

An important application for line drawing is interactive 3D rendering, such as in technical illustrations, games, and other virtual reality environments. Line drawings of static geometry can be rendered in real-time, because curvatures can be precomputed [DeCarlo et al. 2004]. For dynamic geometry, curvatures must be recomputed for each frame, and current methods for computing surface curvatures and their derivatives are too slow to be used in real-time (even for moderate-sized meshes). While some types of curvature-based drawings may be computed in image space [Lee et al. 2007], image-space algorithms often suffer from visual artifacts and lack the stylization options of their object-space counterparts.

This paper presents a fully automatic method for real-time line drawing of deforming objects. In a preprocessing step, a set of *curvature attributes*—namely, curvature tensors and derivatives—are computed for each vertex in each frame of a set of training meshes. The system then learns a mapping from the animation parameters to the curvature attributes. For a skeleton-based character, the animation parameters are simply the joint angles of the skeleton. For facial animation, the parameters are the blending weights. If no animation parameters are given explicitly, then they are determined automatically by dimensionality reduction on the animated surface geometry (e.g., for cloth). Then, during an interactive session, this mapping is applied in real-time to new animation parameters for estimating

Fig. 2: *Real-time renderings generated with our method (principal highlights and suggestive contours for the horse and apparent ridges and valleys for the other figures). For the hand, we apply textured chained-strokes for stylization.*

the curvatures of the new surface, which can then be used to generate line drawings. The system can produce animated 3D line drawings at real-time rates for meshes of 100K triangles in a single processor. With our system, it is now possible to generate accurate and stylizable curvature-based line drawings of 3D animated surfaces in real-time (Figures 1 and 2).

The results of our method are nearly indistinguishable from the per-triangle tensor fitting method of Rusinkiewicz [2004], with similar temporal coherence, but require an order-of-magnitude less computation during runtime. We apply our approach to three types of surfaces: skeleton-based characters, cloth simulation and blend-shape facial animation. We show the ability of our system to generalize to novel animation sequences that are not included in the training set. We demonstrate stroke stylization with real-time chaining (Figures 16 and 17). In addition, stroke thickness can be determined as a function of surface curvature.

A major component of our method is the use of dimensionality reduction to manage high-dimensional inputs and outputs. We use Independent Component Analysis (ICA) to reduce dimensionality, instead of the more commonly-used Principal Component Analysis (PCA). In our experiments, bases learned with ICA generalize better than PCA bases because they better capture local structures in the deformations.

2. RELATED WORK

Line drawing of 3D surfaces has been an active research area in non-photorealistic rendering. Line renderings can include occluding contours and hatching [Elber and Cohen 1990; Hertzmann and Zorin 2000; Markosian et al. 1997; Winkenbach and Salesin 1996], sharp creases [Markosian et al. 1997; Gooch et al. 1999], suggestive contours [DeCarlo et al. 2003; DeCarlo et al. 2004], ridges and valleys [In-

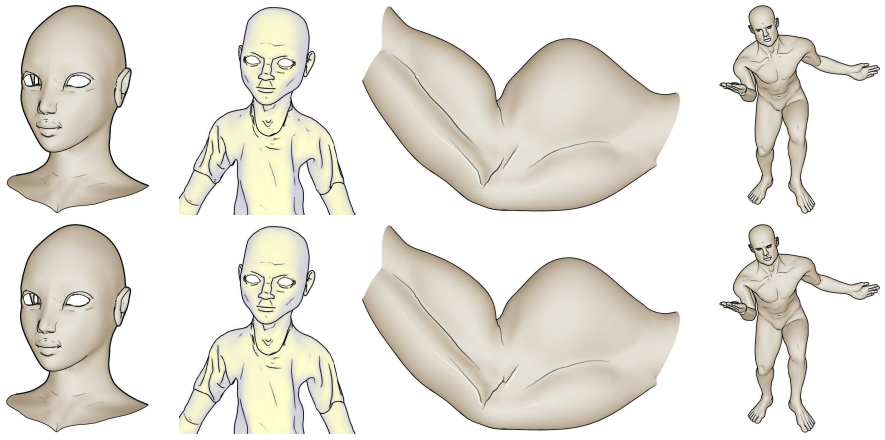

Fig. 3: *Results generated in real-time using our method (**top**) compared to those generated with explicit curvature re-calculation (**bottom**).*

terrante et al. 1995; Ohtake et al. 2004; Thirion and Gourdon 1996], apparent ridges [Judd et al. 2007], and principal and suggestive highlight lines [DeCarlo and Rusinkiewicz 2007; Lee et al. 2007]. Multi-scale representations of feature lines have also been proposed for meshes [Ni et al. 2006] and point-sampled surfaces [Pauly et al. 2003]. Many of these curves rely on the availability of surface curvature and curvature derivatives. However, even the fastest curvature estimation algorithms (e.g., [Taubin 1995; Meyer et al. 2002; Cohen-Steiner and Morvan 2003]) suffer from degenerate cases and noisy estimates, and do not compute third-order surface derivatives [Rusinkiewicz 2004]. Other fast methods based on focal surface approximations [Yoshizawa et al. 2007] are also affected by degeneracies and do not apply in parabolic regions (unless refined by slow non-linear optimization techniques [Yu et al. 2007]). In general, in order to maintain robustness to noise, irregular tessellation, and also to fully compute third-order derivatives, more expensive computations are necessary. Typically, multiple steps of curvature smoothing or feature-preserving optimization of the curvature tensors are required [Rusinkiewicz 2004; Kalogerakis et al. 2007].

Image-processing algorithms can extract some types of feature lines and do not depend on object-space estimation of differential attributes [Lee et al. 2007; Saito and Takahashi 1990]. Lee *et al.* [2007] demonstrate near-interactive animations of line drawings using GPU-based image processing operations. Image-space methods are appealing in that they are generally simple and easy to implement. Furthermore, Lee *et al.* show how level-of-detail abstraction occurs automatically in image-space computation. However, there are a number of drawbacks as well: accuracy is limited by pixel resolution (often resulting in jagged or irregular lines), stylization options are limited (e.g., curves cannot be textured), speed is limited by hardware image processing performance, and careful setting of user-defined thresholds is re-

quired. As our approach operates in object space, it might be more complex to implement. However, it provides noticeable improvements in speed, visual quality, and stylization options.

Our work is inspired by methods for precomputing deformation and radiance transfer. Example-based skinning algorithms [Lewis et al. 2000; Mohr and Gleicher 2003; Wang et al. 2007] learn mappings from skeleton parameters to 3D shapes; our method for skeleton-based characters learns mappings to surface curvature. For cloth simulation, our method is in the same spirit with the photorealistic rendering algorithms of James and Fatahalian [2003] and Nowrouzezahrai *et al.* [2007; 2008; 2009], in which dimensionality reduction is applied to relate simulation and animation to rendering. To the best of our knowledge, this paper presents the first data-driven method for curvature estimation.

3. OVERVIEW

Our goal is to produce line drawings of a deforming smooth surface in real-time. For many of the curves we wish to draw—including suggestive contours, apparent ridges, ridges, valleys, principal highlights, and suggestive highlights—curvatures and/or their derivatives are required for each vertex. Computing these values is the bottleneck for line drawing; the main contribution of our work is computing them in real-time using a very compact model. Storing all the curvature and derivatives of curvature values per frame or storing key poses and then interpolating would require prohibitive amounts of storage (Section 8). Instead, we employ a series of learning techniques during precomputation so that only a few megabytes of storage are required per dataset, curvature synthesis is performed very efficiently and accurately during runtime and generalization capabilities are offered for novel, unseen animation sequences.

The main idea of our paper is to learn a mapping from a low-dimensional shape representation to the set of curvature attributes. The shape is represented by a low-dimensional state vector $\mathbf{x}_t$ at time t: for a skeleton-based character, cosines of joint angles of the underlying skeleton are used as $\mathbf{x}$; for blend-shape animation, the blending weights are used as $\mathbf{x}$. For cloth simulation, this parameterization is automatically determined by dimensionality reduction. The set of curvature attributes is represented by eight values capturing the curvature and derivatives-of-curvature at the surface vertices.

3.1 Stages

Our approach to computing line drawing has two stages: the preprocessing stage, which is performed offline, and the runtime synthesis stage, which is performed in real-time.

Preprocessing. In a preprocessing stage, we begin with an animation sequence, from which we can compute a set of M training pairs $\{(\mathbf{x}_i, \mathbf{y}_i)\}$, where $\mathbf{x}_i$ are the parameters for a mesh and $\mathbf{y}_i$ are a set of curvature attributes. The curvatures for these meshes are computed with the algorithm of Rusinkiewicz [2004]. Additional

curvature smoothing and optimization steps are required in order to obtain high-quality results [DeCarlo et al. 2003; Kalogerakis et al. 2007].

Then, we learn a mapping from the low-dimensional parameterization to surface curvatures:

$$\mathbf{y} = f(\mathbf{x}) \tag{1}$$

This mapping is very high-dimensional, and can also be highly nonlinear. Furthermore, it is crucial that the mapping can be evaluated fast enough to allow real-time rendering. In some cases, the locality of the mapping can be exploited, e.g., that surface curvatures will only be affected by a few nearby joints. We develop a method that combines dimensionality reduction and regression, while also taking advantage of locality as much as possible.

For a skeleton-based character, linear regression with a low-order polynomial model is used. A quadratic model is used for surface curvatures and principal directions and a cubic model is used for the derivatives of curvature. For cloth simulation, first, a low-dimensional representation of geometry is discovered and then linear regression is used. For facial animation, neural network regression maps from the blending parameters to the curvature space.

Run-time rendering. During an interactive session, the parameters $\mathbf{x}$ are determined for each frame. The curvatures $\mathbf{y}$ are computed by $\mathbf{y} = f(\mathbf{x})$. Then, various rendering options are supported. The mesh can be rendered with contours, suggestive contours, and any other lines that requires curvature. Real-time chaining can be performed to provide more stylization options, such as texturing strokes. Stroke thickness can also be determined as a function of surface curvature.

Our method for skeleton-based surfaces is described in Section 4. Simulated cloth surfaces are described in Section 5, followed by our method for blend-shape facial animation in Section 6.

3.2 Curvature attributes

The surface curvature data $\mathbf{y}$ can be represented in different ways. For our application, there are three primary considerations in choosing a representation. First, we want a spatially smooth representation that exploits the local correlations in the curvature field in order to reduce the size of the model through dimensionality reduction (Section 3.3). Second, we want the representation to smoothly vary as a function of animation parameters in order to achieve accurate regression and better temporal coherence. Lastly, we want a representation that stores as few values as possible for each vertex, in order to reduce storage costs. In order to fulfill these goals, we represent the curvature attributes as follows:

(1) **The principal curvatures** k_1 and k_2. We use the standard definition where $k_1 > k_2$, rather than $|k_1| > |k_2|$ [Rusinkiewicz 2007], since the latter definition introduces temporal discontinuities in the curvature field (swapping of principal directions), which adversely affects the learning procedure.

(2) **The principal direction of maximum curvature** $\vec{e}_1$. This direction is represented by using its first two components in a local coordinate system. This particular representation of the curvature attributes is chosen for invariance to rigid transformations of parts of the surface. While a 1D angular representation (i.e., the angle in the tangent plane of each vertex) would be more compact, this parametrization would have singularities at 2π.

The local coordinate systems are determined by first segmenting the surface into rigid segments and then performing PCA on the vertices of each segment. For skeleton-based characters, the segmentation is computed by applying mean-shift clustering [Comaniciu and Meer 2002] to the skinning weights. For cloth simulation, rigid components are found using the method of James and Twigg [2005]. At run-time, the third component of $\vec{e}_1$ and the other principal direction $\vec{e}_2$ can be computed from this representation and the per-vertex normals. Per-vertex normals are computed in a standard manner per-frame, i.e., as the weighted average of incident face normals. In order to improve spatial smoothness, we also adjust the principal directions to match the segment's coordinate system orientation. The local rigid coordinate frame is aligned to a reference mesh edge and normal vector (which are selected to match closely the PCA directions) for each segment. Then, in subsequent frames, we orient the principal directions to match their previous orientation in order to achieve temporal coherence.

(3) **The derivatives of curvatures.** These derivatives form a $2 \times 2 \times 2$ tensor, which, due to symmetry, can be represented by four values.

We will learn a separate mapping ($\mathbf{y} = f(\mathbf{x})$) from the animation parameters $\mathbf{x}$ to each of the eight curvature attributes listed above.

3.3 Dimensionality reduction

The attribute vector $\mathbf{y}$ for a mesh is very high-dimensional. However, as there are significant spatial correlations in the curvatures, dimensionality reduction can be employed to significantly compress these vectors. Dimensionality reduction also helps to denoise unstable attributes (such as principal directions near umbillic points), since noisy data are not captured by the first few principal components, since they correspond to larger variance in the data. Thus, noisy data are not represented in the low-dimensional subspace.

We use ICA [Bell and Sejnowski 1997; Comon 1994; Cao et al. 2003], a linear dimensionality reduction technique. Like PCA, ICA computes a linear low-dimensional projection. While PCA has the property that it is least-squares optimal for compressing the training data, this does not guarantee that it will generalize to new shapes not included in the training data. In fact, we find that ICA does generalize better because it prefers sparse bases, yielding localized basis functions corresponding to structure in the data, such as folds, wrinkles, and other similar structures (Figure 10). Similarly, it has been often noted in the literature that ICA applied to image data yields localized features, e.g., [Bartlett et al. 2002; Bell and Sejnowski

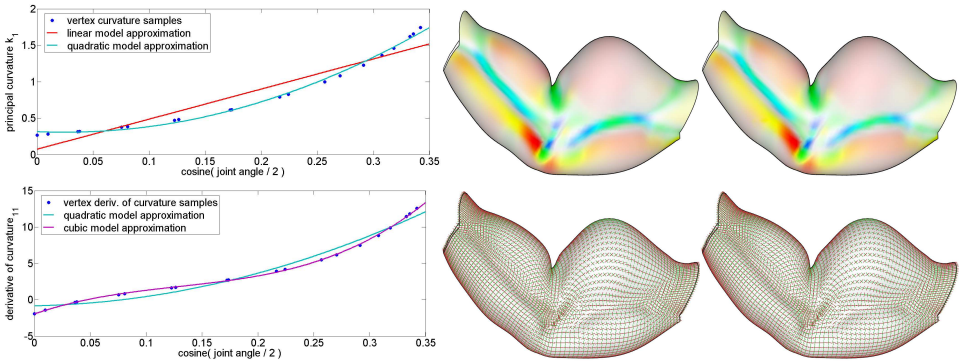

Fig. 4: Left: *Typical plots of curvature and derivatives of curvature at a vertex as a function of one joint angle, for the muscle mesh. The vertex shown is the one with highest variance in principal curvature k_1 during training. The quadratic model is more suitable than a simple linear model, while a cubic model is more appropriate for the derivatives of curvature components.* Top right: *Comparison of principal curvatures produced by the method of Rusinkiewicz [2004], as compared to those produced by our method.* Bottom right: *Comparison of principal directions. The most significant differences in principal direction estimates occur at umbilic points where the directions are unstable. We also report the running times for Rusinkiewicz's and our method.*

1997]. In contrast, the PCA bases are global: the first components contain a mixture of many distinct folds and wrinkles that are less likely to co-occur for novel poses.

4. SKELETON-BASED DEFORMATIONS

Our method for skeleton-based curvature prediction exploits the special structure of skinned geometry. Specifically, we note that the skeleton's joint angle values provide a natural parameterization, and so we will use them as the inputs $\mathbf{x}$ to the regression. Furthermore, the curvature attributes we wish to predict depend only locally on joint values. For example, the angle of an elbow affects the skin only within its nearby support area, and not the rest of the body. This is similar to the locality of weights used in example-based skinning algorithms (e.g., [Mohr and Gleicher 2003; Wang et al. 2007]).

Our method for skeleton-based characters works as follows. First, we gather the training data, and represent it as described in the next section. We predict curvature as a function of joint angles, using a polynomial regression model described in Section 4.2. For each vertex, we determine which joints have a significant influence on the curvature at the vertex by applying a statistical test (Section 4.3). To simplify the regression, we perform dimensionality reduction on the curvature attributes of the influenced vertices per joint (Section 4.4). Finally, we apply regression to build the mapping from animation parameters to curvature (Section 4.5).

4.1 Training

We begin with a set of training poses. These poses may, for example, correspond to a typical animated sequence for this character. Following Wang et al.'s scale/shear regression [2007], we represent a pose $\mathbf{x}$ as a vector of *bones*, where each bone is parameterized by the associated joint and its parent joint angles. Each joint is represented as three Euler rotation angles with respect to the corresponding axis of rotation in its local coordinate frame. Therefore, each bone has 6 degrees of freedom. We represent each element of $\mathbf{x}$ as $\cos(\theta/2)$, where θ is a joint angle. This representation is motivated by the fact that the discrete mean curvature at an edge depends on the cosine of half of the dihedral angle [Polthier 2002], and thus these values were found to be better for predicting curvature.

For each training pose i, we compute the corresponding surface attributes $\mathbf{y}_i$. Note that some vertices can be treated as rigid, such as vertices with neighborhoods influenced only by one bone. We detect vertices with curvature variation less than 0.5% of the maximum curvature variation in the data. These vertices are treated as having constant curvature and removed from the learning process. In the Mr. Fit model (Figure 7), about 25% of the vertices are treated as rigid.

4.2 Regression model

In order to select an appropriate regression model, we first consider the case of a character with only a single joint. As shown in Figure 4, we find that the curvature at the vertices around a joint can be approximated very well by a quadratic function of the joint angle, while a cubic is sufficient for derivative of curvature. We found that higher-order models (such as B-splines) are more powerful than necessary for articulated data, thus requiring more storage and running time for the same-quality results while also exhibiting poorer generalization.

Hence, we will perform regression with the model

$$\mathbf{y} = \mathbf{V}\boldsymbol{\phi}(\mathbf{x}) \tag{2}$$

where $\mathbf{V}$ is a matrix of regression weights. For surface curvatures, we use quadratic features:

$$\boldsymbol{\phi}(\mathbf{x}) = [1, x_1, ..., x_K, x_1^2, ..., x_K^2]^T \tag{3}$$

while, for derivatives of curvature, we use cubic features:

$$\boldsymbol{\phi}(\mathbf{x}) = [1, x_1, ..., x_K, x_1^2, ..., x_K^2, x_1^3, ..., x_K^3]^T \tag{4}$$

where K is the total number of joint angles. We omit the bilinear terms $x_i x_j$ for $i \neq j$ and other higher-order terms, as we have found that these lead to worse generalization, due to overfitting.

4.3 Determining which joints influence curvature at each vertex

In general, the curvature attributes at a vertex can be affected by more than one joint, namely, all joints with nonzero skinning weight at that vertex. Joints with

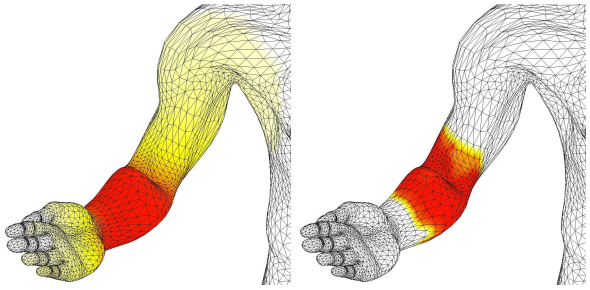

Fig. 5: Left: *Typical smooth skinning weights for an elbow joint.* Right: *Curvature attribute weights ($w_{j,v}$) from the elbow joint as determined by our method. Note that fewer curvatures require input from this joint; additionally, the distribution of weights is noticeably different from the skinning weights. We observed that blending the curvatures with skinning weights between different joints resulted in significant errors and lower runtime speed. With our weighting scheme, the weights of the joints on the curvature of vertices were distributed more appropriately.*

nonzero weights at neighboring vertices can also affect curvature. However, the curvature can often be predicted using only a subset of these joints. In order to reduce the model size, we determine a subset of joints to be used for regression at each vertex. That is, we find the joints which have a significant effect on the curvature at vertex v. For this purpose, we use a statistical test applied at each vertex. This statistical test is performed based on prediction of mean curvature $\kappa = (k_1 + k_2)/2$. The joints selected to influence each vertex based on mean curvature will be used for all other curvature attributes.

Specifically, for each vertex, we fit the mean curvature values for each training pose i by least-squares regression, minimizing:

$$E_{\text{FULL}} = \sum_i ||\kappa_i - \mathbf{a}^T \boldsymbol{\phi}(\mathbf{x}_i^{[v]})||^2 \tag{5}$$

where $\mathbf{x}_i^{[v]}$ are the K elements (joint angles) of $\mathbf{x}_i$ that influence vertex v (as determined by the skinning weights), $\boldsymbol{\phi}$ is a quadratic feature vector (Equation 3), and $\mathbf{a}$ are the regression weights. Then this regression is repeated using only individual joints as inputs (as in Section 4.2). Regression on joint j (i.e., using the six elements of its angles and its parent joint angles as the inputs $\mathbf{x}_i^{[v]}$) gives another residual E_j. An F-test [Weisberg 2003] is then applied to determine whether to keep the joint's influence: this test simply determines whether including a joint makes a significant improvement to the residual. Specifically, the F statistic is:

$$F = \frac{(E_j - E_{\text{FULL}})/(9J - 6)}{E_{\text{FULL}}/(N - 9J)} \tag{6}$$

where J is the number of joints for this vertex with non-zero skinning weights and N is the number of training poses. The corresponding joint will then be kept for the regression for v if F is greater than the critical value for the F distribution for $p > 0.05$.

This test is repeated for all joints with nonzero skinning weights at this vertex; those that pass the test are deemed as influencing this vertex. If all the joints fail

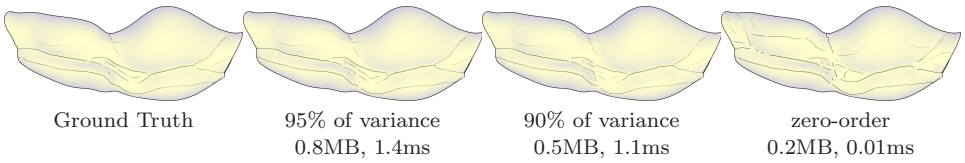

Fig. 6: *Ridges and Valleys for muscle dataset with respect to decreasing variance captured by the basis. Ridges and Valleys based on ground truth curvature data is on the left. A zero-order prediction based only on the mean of the curvature data is also depicted on the right for comparison. A reasonable choice that balances the trade-off between speed and accuracy is selecting the number of components based on 95% of the variance. The size of the model and running times per frame are also shown.*

the F-test, then the one with smallest residual E_j is kept. In practice, we observe that two joints are sufficient for most vertices in most cases. Although it is possible that a joint of a bone will affect the curvature at a vertex for which it has zero skinning weight, smoothness of the skinning weights implies that the effect of the bone is negligible. In our experiments, this statistical test typically halves the size of the learned model and speeds up run-time curvature prediction by 150-200%.

4.4 Dimensionality reduction

Due to the large number of vertices, directly learning the mapping to all the curvature attributes per vertex would require estimating and storing an impractical number of weights. Instead, we exploit the spatial coherence of the curvature attributes and perform regression on a reduced-dimensional model, as described below. The following process is performed eight times, once for each curvature attribute.

For each joint j, we define a vector $\mathbf{y}^{(j)}$ consisting of the values of the curvature attribute to be predicted from this joint. One such vector $\mathbf{y}_i^{(j)}$ is computed for each pose i in the training set and contains the attribute to be predicted (e.g., k_1). Because this vector $\mathbf{y}^{(j)}$ is high-dimensional (its dimensionality is equal to the number of vertices influenced by the joint), we apply ICA to the training data to obtain a reduced representation:

$$\mathbf{y}^{(j)} = \mathbf{W}_j \mathbf{z} + \bar{\mathbf{y}} \tag{7}$$

All terms on the right-hand side are determined by the FastICA algorithm [Hyvärinen 1999]. We keep the first D independent bases, where D is set to the number of the eigenvalues required to capture 95% of the variance of W. This threshold is selected empirically to balance the trade-off between speed and accuracy (Figure 6).

4.5 Regression

We use least-squares regression to map from the animation parameters $\mathbf{x}$ to their corresponding values $\mathbf{z}$ in the low-dimensional space of curvature attributes. Specif-

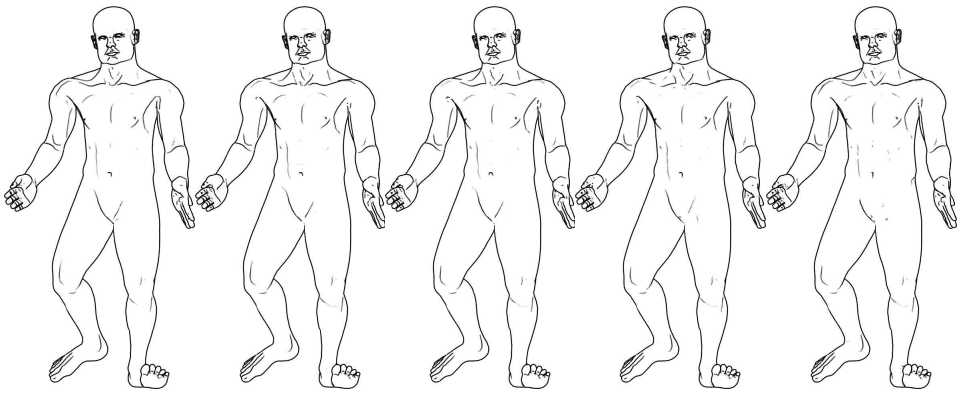

Fig. 7: *Suggestive contours for Mr. Fit dataset with respect to the number of training examples. Suggestive contours based on ground truth curvature data are on the left. Our system can accurately synthesize surface curvatures using a few training examples.*

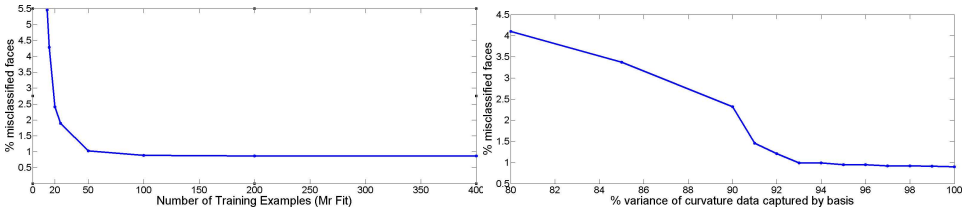

Fig. 8: Left: *Plot of % misclassified faces for the suggestive contour drawings for the Mr. Fit test sequences versus number of training examples (the number of ICA components is chosen to correspond with the 95% of the variance of the curvature data). More precisely, we compute the percentage of mesh faces that are not identified as having or not having a suggestive contour. Note that the test error is smoothly decreasing and is relatively small even for a small number of training examples. The minimal amount of training data is 19 training poses for character animation sequences since there are at most 6 DOFs and the feature vector is cubic for derivatives of curvature.* Right: *Plot of % misclassified faces for ridge and valley drawings for the muscle dataset versus the variance of the curvature data captured by our basis. The zero-order prediction had an error of 6.25%.*

ically, we solve for the weights $\mathbf{V}$ that minimize

$$\sum_{i=1}^{M} ||\mathbf{z}_i - \mathbf{V}_j \boldsymbol{\phi}(\mathbf{x}_i^{(j)})||^2 \tag{8}$$

where $\mathbf{x}^{(j)}$ are the six elements of $\mathbf{x}$ that depend on joint j. Each joint now provides a separate predictor of the curvature at a particular vertex v, i.e.,

$$\tilde{y}_{v,j}(\mathbf{x}) = \mathbf{W}_j \mathbf{V}_j \boldsymbol{\phi}(\mathbf{x}^{(j)}) + \bar{\mathbf{y}}_v \tag{9}$$

where the subscript v indexes rows specific to that vertex. The predictor $\tilde{y}_{v,j}(\mathbf{x})$ can be viewed as an estimate of y_v. (Note that each joint j will have its own $\mathbf{W}$ and $\mathbf{V}$ matrices).

We create the final predictor of y_v by linearly combining these predictors in a manner similar to boosting [Bishop 2006].

$$y_v^* = f_v(\mathbf{x}) = \sum_{j=1}^{\hat{J}} w_{j,v} \tilde{y}_{v,j}(\mathbf{x}) \tag{10}$$

where $\hat{J}$ is the number of joints with nonzero influence on this vertex (as determined in Section 4.3). One option for determining the weights w is by least-squares fitting. However, we have obtained better results by weighting the predictors according to their fit to the training data. Specifically, let $r_j = \sum_i (y_{i,v} - \tilde{y}_{v,j}(\mathbf{x}))^2$ be the residual of the j-th predictor. Then, we set the weight for predictor j proportional to the sum of the residuals for all other predictors, normalized to sum to 1:

$$w_{j,v} = \frac{\sum_{k \neq j} r_k}{(\hat{J} - 1) \sum_{k=1}^{\hat{J}} r_k} \tag{11}$$

where $\hat{J}$ is the number of predictors. This can be thought of as similar to the linear blend skinning process, but averaging target curvatures rather than target poses. We visualize our resulting weights in Figure 5.

4.6 Run-time evaluation

During run-time, given a new pose $\mathbf{x}$, the curvature attributes for each vertex are computed by applying Equation 10. Curvature prediction is visualized in Figure 4. We also provide error analysis with respect to the number of ICA components and number of training examples used in Figure 7. Example skeleton-based renderings are shown in Figures 1, 2, 3, 6, 7, 17 and in the accompanying video. We also show examples of generalization of our method to novel animation sequences in the accompanying video.

5. CLOTH SIMULATION

To learn curvatures for cloth simulation, we begin with an animated cloth sequence $(\mathbf{s}_1, .., \mathbf{s}_M)$ as training data. Our goal is to be able to compute curvatures $\mathbf{y}$ for a new cloth shape $\mathbf{s}$. Because no low-dimensional state vector is provided for the cloth, we apply dimensionaily reduction to the animation state to obtain one. We will learn a mapping from this low-dimensional space derived from the current cloth shape $\mathbf{s}$ (Section 5.1) to the low-dimensional space of surface curvatures (Section 5.2).

5.1 Dimensionality reduction for cloth state

We apply ICA to the 3D cloth shapes $\{\mathbf{s}_i\}$ to obtain animation parameters $\{\mathbf{x}_i\}$ such that $\mathbf{s} = \mathbf{A}\mathbf{x} + \bar{\mathbf{s}}$ [Bishop 2006; James and Fatahalian 2003]. For this step, we represent the cloth state $\mathbf{s}$ in terms of dihedral angles. For example, we typically find that 50 basis vectors are sufficient to represent 95% of the variation for the

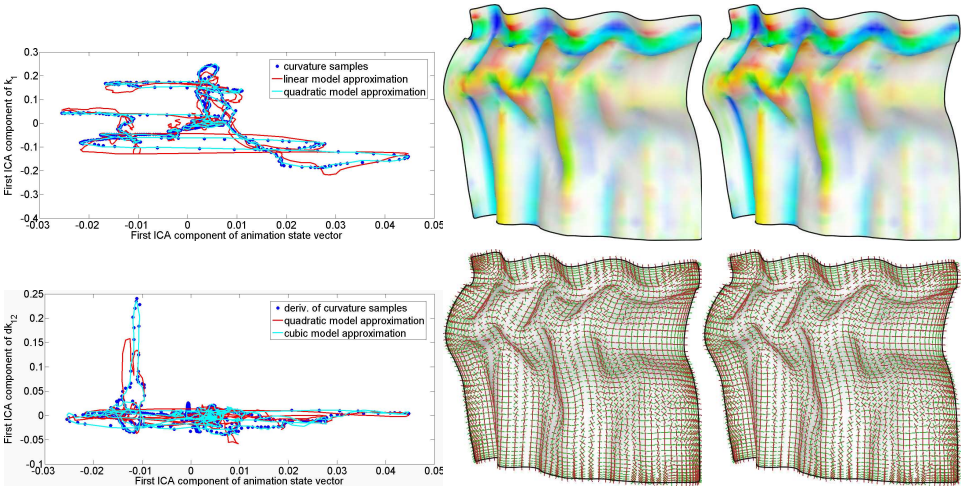

Fig. 9: Left: *Typical plots of the first ICA component of curvature and derivatives of curvature data for a cloth simulation with respect to the first ICA component of the animation state vector. A quadratic and a cubic model are more appropriate for fitting curvatures and derivatives of curvatures respectively.* Top right: *Comparison of principal curvatures produced by the method of Rusinkiewicz [2004] and smoothed, as compared to those produced.* Bottom right: *Comparison of principal directions. We also report running times for both methods.*

horse cloth with 10K vertices (and thus 20K dihedral angles) providing a good tradeoff between speed and prediction accuracy.

In addition, ICA is applied to curvature data to obtain a reduced representation as well:

$$\mathbf{y} = \mathbf{W}\mathbf{z} + \bar{\mathbf{y}} \tag{12}$$

5.2 Regression

As for articulated characters, we use least-squares regression with quadratic features to map from the low-dimensional animation state $\mathbf{x}$ to the corresponding low-dimensional surface curvatures $\mathbf{z}$. More specifically, we estimate weights $\mathbf{V}$ to minimize:

$$\sum_{i=1}^{M} ||\mathbf{z}_i - \mathbf{V}\boldsymbol{\phi}(\mathbf{x}_i)||^2 \tag{13}$$

5.3 Run-time evaluation

Given a new cloth shape $\mathbf{s}$, generating curvatures requires the following steps. First, the dihedral angles are projected to the ICA subspace to obtain the low-dimensional state. Then, the new curvatures $\mathbf{y}^*$ are predicted for the vertices of the cloth as:

$$\mathbf{y}^* = f(\mathbf{x}) = \mathbf{W}\mathbf{V}\boldsymbol{\phi}(\mathbf{x}) + \bar{\mathbf{y}} \tag{14}$$

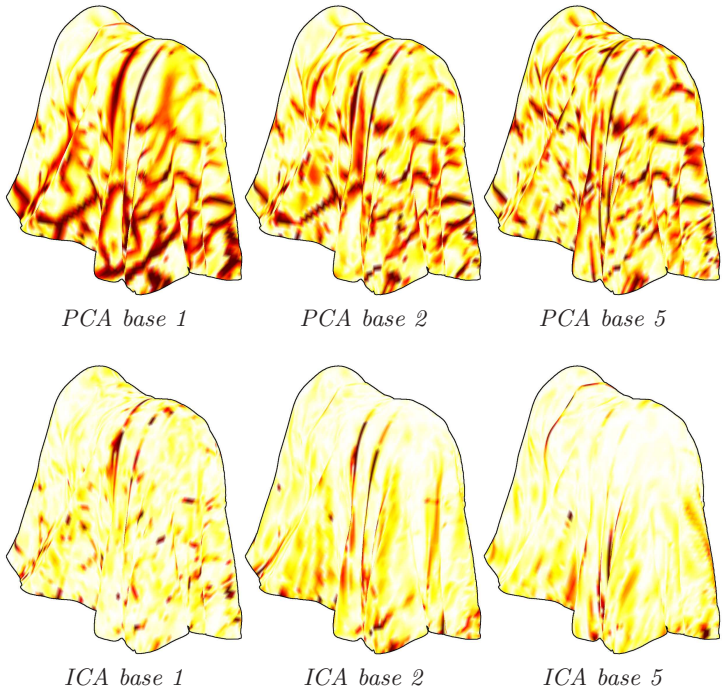

Fig. 10: *Cloth curvature-bases found by PCA (**top**) and ICA (**bottom**). The ICA bases exhibit much greater sparsity and locality, capturing fold and wrinkle structures. Colors correspond to magnitude, with white for zero and red for the largest magnitude.*

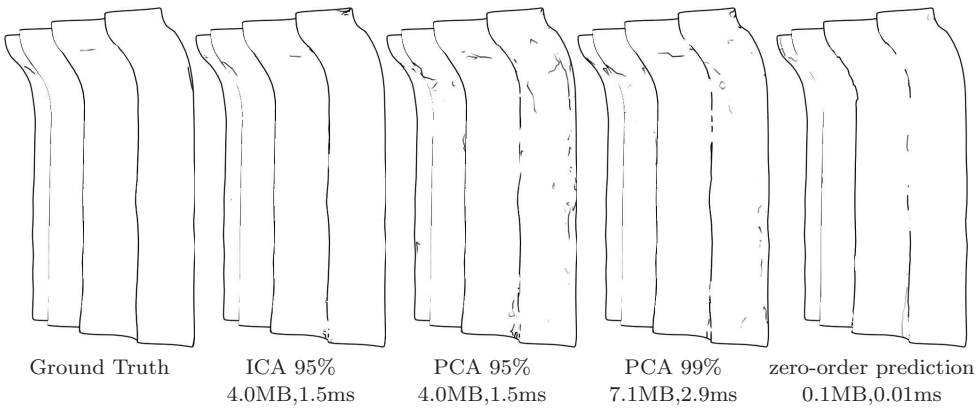

Fig. 11: *Suggestive contours for a novel frame of cloth with respect to the basis used corresponding to the given variance.* From left to right: *We show results for ground truth, ICA with number of base vectors corresponding to 95% of the variance of the curvature data, PCA capturing 95% of the variance and zero-order prediction. The sparsity and locality of ICA, as depicted in Figure 10, offers better line drawing results. Even if the number of basis is increased for PCA (99% correspond to three times more coefficients), the result does not improve much.*

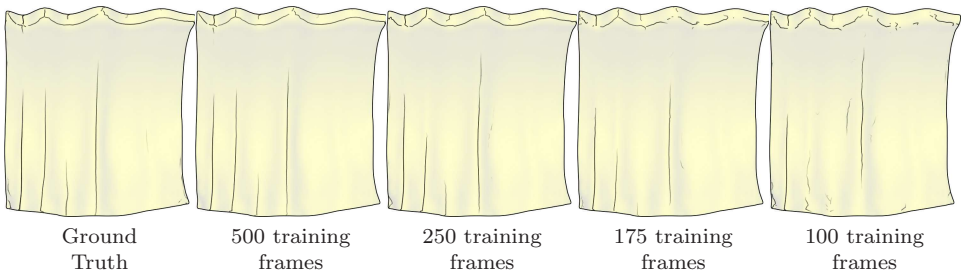

Fig. 12: *Apparent ridges for a novel frame of cloth with respect to the number of training examples.*

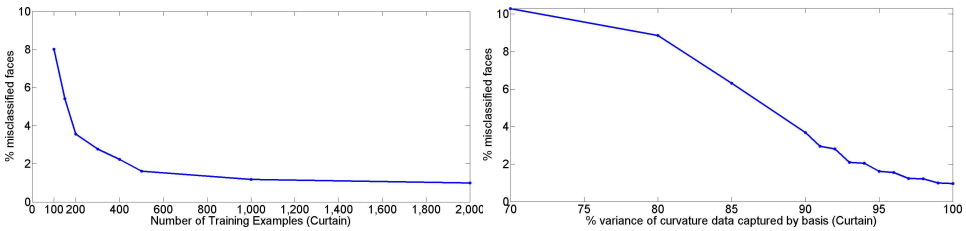

Fig. 13: Left: *Plot of % misclassified faces for apparent ridges drawing for the curtain test sequence versus number of training examples (the number of ICA components is chosen to correspond the 95% of the variance of the curvature data). The minimal amount of training data is 97 training poses for character animation sequences since the dimensionality of the animation state vector is 32 and the feature vector is cubic for derivatives of curvature (the minimal amount of training data depends on the dimensionality of the reduced animation state vector deduced in the first step. Typically, for keeping 95% of the animated geometry, this varies from 30 to 100 in our examples).* Right: *Plot of % misclassified faces for apparent ridges drawing for the same dataset versus variance of curvature data captured by the basis for curvature. The zero-order prediction had error 20.58%.*

Example cloth renderings using our method are shown in Figures 1, 2, 3 and in the accompanying video. In Figure 11 and 12, we also provide error analysis as a function of the number of independent bases and the number of training examples used respectively. Given training data covering a range of motions, our model can still predict the curvature when the parameters of the dynamics (e.g., an air field or a turbulence field) controlling the cloth animation change. We show the generalization of our method in the accompanying video.

6. BLEND-SHAPE FACIAL ANIMATION

In the case of blend-shape facial animation, we assume we are given M low-dimensional weight vectors $\mathbf{x}$, each of which can be used to generate a 3D face shape $\mathbf{s}$ by blending. For each training pose, we compute the surface curvature attributes $\mathbf{y}$. Unlike with skeleton-based characters and cloth, in the case of facial animation, we did not find a simple linear relationship between the blending parameters and the curvature attributes (Figure 14). We employ Artificial Neural Network (ANN) regression to fit this nonlinear map.

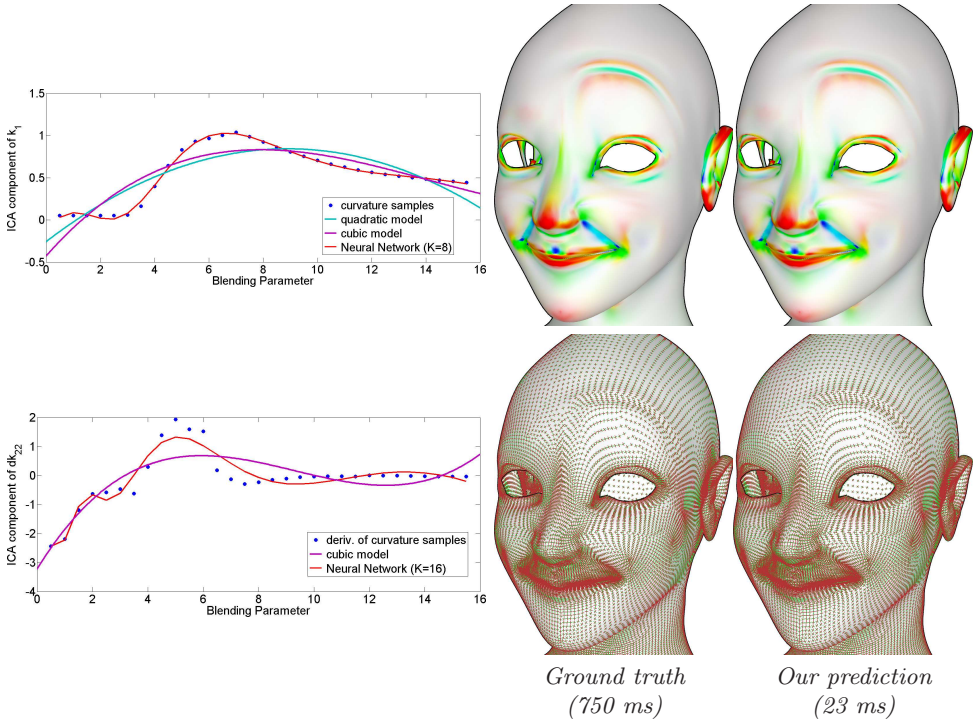

Fig. 14: Left: *Typical plots of the first ICA component of curvature and derivatives of curvature for a face animation with respect to one of the blending parameters. In this case, a quadratic or a cubic model cannot approximate the data well. On the other hand, non-linear regression with ANNs is more appropriate in this case. The number of neurons is selected with cross-validation.* Middle: *Comparison of principal curvatures produced by the method of Rusinkiewicz [2004] and smoothed, as compared to those produced by our ANN.* Right: *Comparison of principal directions.*

6.1 Neural Network Regression

As before, the learning process starts by reducing the curvature data with ICA, $\mathbf{y} = \mathbf{W}\mathbf{z}+\bar{\mathbf{y}}$, once for each of the eight curvature attributes. We then perform ANN regression [Bishop 2006] to learn a nonlinear mapping from the dimensionality-reduced shape $\mathbf{x}$ to the dimensionality-reduced curvature $\mathbf{z}$; one such regression is performed for each of the ICA coefficients of all the 8 curvature attributes. The ANN for each attribute has the form:

$$g(\mathbf{x}) = \sum_{\ell=1}^{L} \mathbf{w}_\ell \tanh\left(\mathbf{b}_\ell^T \mathbf{x} + b_0\right) + w_0 \tag{15}$$

where L is the number of neurons, $\mathbf{w}_\ell$ and $\mathbf{b}_\ell$ are L pairs of weight vectors, and w_0 and b_0 are bias terms. The weights are obtained by optimizing the following regularized least-squares objective:

$$E(w,b) = \sum_i ||\mathbf{z}_i - g(\mathbf{x}_i)||^2 + \lambda \sum_{\ell=1}^{L} \left(||\mathbf{w}_\ell||^2 + ||\mathbf{b}_\ell||^2\right) \tag{16}$$

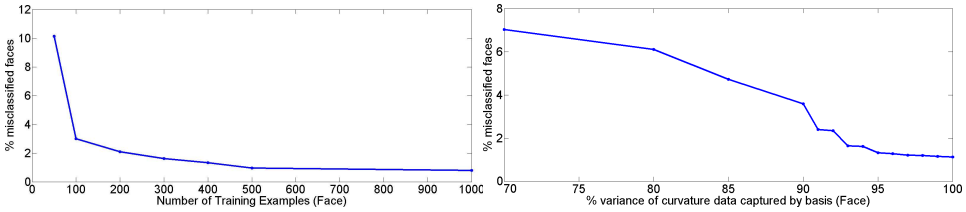

Fig. 15: Left: *Plot of % misclassified faces for suggestive contours for the face test sequence versus the number of training examples (the number of ICA components is chosen to correspond the 95% of the variance of the curvature data).* Right: *Plot of % misclassified faces for suggestive contour drawings for the same dataset versus the variance of curvature data captured by the basis. The zero-order prediction had an error of 14.85%.*

where λ is a smoothing parameter and L is the number of neurons. Optimization is performed by 5000 iterations of the BFGS algorithm with cubic line search [Nocedal and Wright 1999]. The weights $\mathbf{w}$ and $\mathbf{b}$ are initialized by sampling from a uniform distribution over $-1/K$ to $1/K$ for the elements of $\mathbf{w}_\ell$ (where K is the number of blending parameters) and over $-1/L$ to $1/L$ for $\mathbf{b}_\ell$. The smoothing parameter λ and the number of neurons L is chosen by cross-validation [Bishop 2006] in a preprocessing step.

6.2 Run-time evaluation

Given a new face with blending parameters $\mathbf{x}$, we compute the surface curvatures as follows:

$$\mathbf{y}^* = f(\mathbf{x}) = \mathbf{W}g(\mathbf{x}) + \bar{\mathbf{y}} \tag{17}$$

We show our curvature synthesis results in Figure 3 and in the accompanying video.

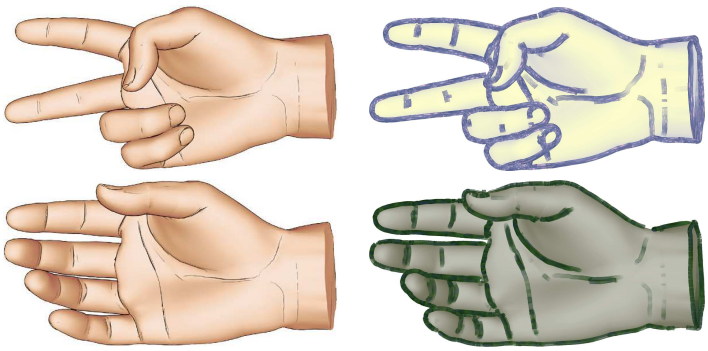

Fig. 16: *Regular curvature-modulated stylization (**left**) and textured chained-strokes (**right**), using apparent ridges and valleys.*

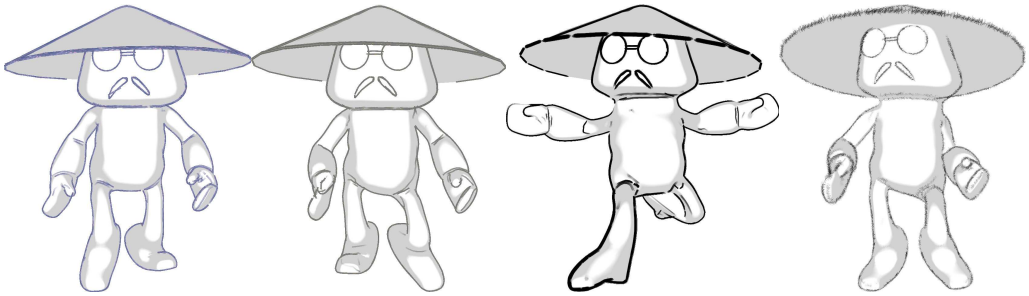

Fig. 17: *More textured chained-strokes for Master Pai dataset, using apparent ridges and suggestive contours.*

7. STYLIZATION

Our default rendering style entails detecting surface curves (such as contours and suggestive contours) defined as zero-sets [DeCarlo et al. 2003; Hertzmann and Zorin 2000]. Each mesh face yields a line segment, which is rendered in OpenGL. The curvatures generated by our method can also be used for stroke stylization: following Goodwin *et al.* [2007], we make line thickness T a function of depth z and radial curvature κ_r: $T = \text{clamp}(c/(z(\kappa_r + \epsilon)))$, where c and ϵ are user-defined constants, and clamp$(\cdot)$ clamps the thickness to a user-defined range.

Additional stylization effects are possible by chaining curves on the surface; we modify the method of randomized contour detection of Markosian *et al.* [1997] for zero-set contours and suggestive contours. For each frame, the algorithm iterates over every face in the mesh. When a face is detected that contains a contour or suggestive contour (represented as a line segment), the algorithm "walks" along the mesh, following the contour or suggestive contour until it ends or loops. This walking is performed in two directions from the starting face. This produces a chain of line segments (one for each face). Visibility for each point on the chain is computed using a reference ID image, and visible portions of chains are rendered with textured triangle strips [Northrup and Markosian 2000].

8. RESULTS

We test our method on ten datasets, including skeleton-based characters, cloth and facial animation (Figures 1, 2 and 3 and the accompanying video). Curvatures computed with our method have very low error (Figure 3, 8, 13, 15). Visual differences between our curvatures and ground truth are negligible (Figures 4, 9 and 14); differences in final line drawings are also negligible. As ground truth, we used Rusinkiewicz's method plus curvature smoothing when necessary [2004] and Kalogerakis *et al.*'s method [2007].

As a baseline comparison, we compare with the performance of Rusinkiewicz's method that is efficient and can fully compute both curvatures and derivatives-of-curvature for line drawings. Our curvature calculation at runtime is about 10 times faster than this method. However, this comparison is somewhat misleading:

in order to generate smooth and more temporally coherent line drawings for many datasets, a few rounds of curvature and derivatives of curvature smoothing are required based on vector field diffusion [Diewald et al. 2000] (also implemented in the trimesh2 library [Rusinkiewicz 2007]) or maximum likelihood estimates [Kalogerakis et al. 2007]. These operations add significantly to run-time computation. Simple mesh smoothing can be done in advance but eliminates surface detail and alters the mesh.

Thus, our method is approximately 10 times faster than Rusinkiewicz's method (e.g., for smooth and regularly sampled meshes), but in most cases, it is about 20-50 times faster than performing all the necessary smoothing or optimization steps for high-quality smooth and temporally coherent line drawings. More specifically, in our experiments, we smoothed the derivatives of curvature for Mr. Fit, Master Pai and face using vector field diffusion. We smoothed the curvatures for the muscle, draping, curtain, and flag datasets. We used Kalogerakis *et al.*'s method to robustly compute the curvatures and their derivatives for the Angela, hand and horse cloth datasets that seemed to be more noisy. We present running times for our method versus Rusinkiewicz's method and the total curvature re-estimation time including the necessary curvature smoothing in Table I.

An alternative is to precompute curvatures for all frames and store them, for cases where generalization to new frames is not necessary. However, this would be prohibitively expensive; e.g., storing all curvatures for the Mr. Fit dataset (50K faces and 2000 frames) would require about 1 Gb of storage, whereas our method requires 10.7 Mb at run-time. Nearest-neighbor interpolation of curvature values based e.g., on a regularly-sampled grid of examples would also need orders-of-magnitude larger storage (at least 300 Mb) than our technique and with no generalization capability to novel poses. Note that such interpolation requires an exponential amount

Dataset name	Number of Vertices	Rusinkiwicz's method (ms)	plus smoothing /optimization	Our Method (ms)	Model size (MB)
Mr. Fit	20536	81	240	7.9	10.72
Master Pai	11850	29	87	3.2	5.21
Muscle	5256	20	105	1.4	0.8
Hand	9284	25	227	2.6	4.05
Angela	25462	119	930	14	26.07
Curtain	2401	16	91	1.7	4.2
Flag	3285	19	101	2.5	5.0
Horse cloth	7921	41	529	5.0	11.9
Draping cloth	3969	26	124	2.8	4.8
Face	40767	207	750	23	32.66

Table I: *Running times (in sec) for curvature estimation with our method (fifth column) compared to an explicit re-estimation with Rusinkiewicz's method (third column) and explicit re-estimation with Rusinkiewicz's method plus the necessary curvature smoothing or Kalogerakis* et al.*'s optimization technique (fourth column). Note that smooth and plausible line drawings require curvature smoothing in many cases that cannot be performed in advance. In both cases, we exclude the vertices whose curvatures do not change significantly (less than 1% of maximum variance). Timings are captured on a 2GHz Intel Core Duo Processor (no parallelization is used for any of the above methods). We also report the size of our learned model (last column).*

of storage with respect to the number of DOFs and would quickly result in huge model representations when many DOFs are present.

For the case of cloth, approximately 30% of the time is spent on the projection to the ICA basis for the cloth shape. Then, 65% of the time is spent on the ICA re-projection of curvatures. The remainder is used for the model regression and the re-projection of principal directions to the global coordinate system. For face and skeleton-based characters, about 90% of the time is spent on the ICA re-projection of curvatures and the remainder is used by the rest of the operations.

9. SUMMARY, LIMITATIONS AND FUTURE WORK

We have presented a data-driven method for real-time surface curvature computation with applications to NPR. Our method can be used to compute suggestive contours, ridges and valleys, apparent ridges, and highlight lines for real-time NPR. The results are nearly indistinguishable from ground truth. The method does not introduce temporal coherence artifacts.

The major limitation of our approach is the need for training data and a preprocessing step, along with storage space for the learned mappings. This is typical with many real-time rendering applications that are based on offline precomputation steps [Sloan et al. 2002; James and Fatahalian 2003; Nowrouzezahrai et al. 2009]. The most crucial goals in such approaches are efficiency during runtime and compactness of the model, which are fully achieved by our method. The generalization capabilities of our method to novel animation sequences also rely on the training data; i.e., the training data should be sufficient to cover a range of motions based on the analysis and examples we provided in the paper. If the testing data cover completely different ranges of motion, then our method will not generalize. For example, if an elbow joint is not active during the training sequence, our method will not predict the curvatures around this joint for animation sequences where this joint is active; our method will not generalize from a cloth falling onto a table to a flag animation. This dependence on the training data is typical of data-driven methods [James and Fatahalian 2003; Wang et al. 2007].

Curvature is a fundamental component of digital geometry processing. Hence, we believe many previously off-line techniques—such as real-time hatching with smoothed directions [Hertzmann and Zorin 2000], exaggerated shading [Rusinkiewicz et al. 2006], apparent relief [Vergne et al. 2008], curvature-domain shape processing [Eigensatz et al. 2008], and dynamic model simplification [Heckbert and Garland 1999]—can be made real-time for dynamic geometry. Our method computes all curvature attributes independently, so, if only a subset is needed, then the computation time will decrease proportionally; e.g., if only principal directions are needed, then the timing decreases by a factor of 4. As our technique uses primarily matrix-vector operations, we expect that a much faster GPU implementation should be possible. Another interesting area of research is the development of more localized and compact bases for reducing the dimensionality of mesh curvature data.

Acknowledgements

We thank Szymon Rusinkiewicz for providing his rtsc and trimesh2 code online. We thank Robert Wang and Joel Anderson for providing us with animation test sequences. We thank Michael Comet for the muscle arm model. We thank Chris Landreth for the Angela dataset and Alexis Angelidis for the Master Pai mesh. We thank Eitan Grinspun and Rony Goldenthal for the horse and draping cloth. We also thank the reviewers for their insightful and helpful comments which greatly contributed to the clarity of the paper. The motion capture data used on the Fit and Pai datasets was obtained from the CMU Motion Capture database.

This work was funded by the Alfred P. Sloan Foundation, the Canada Foundation for Innovation (CFI), the Canadian Institute for Advanced Research (CIFAR), Microsoft Research, the National Sciences and Engineering Research Council of Canada (NSERC), the Ontario Ministry of Research and Innovation (MRI), the Ontario Ministry of Education and Training and the Canadian Research Network for Mathematics of Information Technology and Complex Systems (MITACS).

REFERENCES

Bartlett, M., Movellan, J., and Sejnowski, T. 2002. Face recognition by independent component analysis. *IEEE Transations on Neural Networks 13,* 6, 1450–1464.

Bell, A. J. and Sejnowski, T. J. 1997. The independent components of natural scenes are edge filters. *Vision Research 37*, 3327–3338.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Cao, Y., Faloutsos, P., and Pighin, F. 2003. Unsupervised Learning for Speech Motion Editing. In *Proceedings of the Symposium on Computer Animation 2003*. 225–231.

Cohen-Steiner, D. and Morvan, J.-M. 2003. Restricted delaunay triangulations and normal cycle. In *Proceedings of the Symposium on Computational Geometry 2003*. 312–321.

Comaniciu, D. and Meer, P. 2002. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24,* 5, 603–619.

Comon, P. 1994. Independent component analysis, a new concept? *Signal Processing 36,* 3, 287–314.

DeCarlo, D., Finkelstein, A., and Rusinkiewicz, S. 2004. Interactive rendering of suggestive contours with temporal coherence. In *Proceedings of the International symposium on Non-photorealistic animation and rendering 2004*. 15–24.

DeCarlo, D., Finkelstein, A., Rusinkiewicz, S., and Santella, A. 2003. Suggestive Contours for Conveying Shape. *ACM Transactions on Graphics 22,* 3, 848–855.

DeCarlo, D. and Rusinkiewicz, S. 2007. Highlight Lines for Conveying Shape. In *Proceedings of the International symposium on Non-photorealistic animation and rendering 2007*. 63–70.

Diewald, U., Preusser, T., and Rumpf, M. 2000. Anisotropic diffusion in vector field visualization on euclidean domains and surfaces. *IEEE Transactions on Visualization and Computer Graphics 6,* 2, 139–149.

Eigensatz, M., Sumner, R. W., and Pauly, M. 2008. Curvature-domain shape processing. *Computer Graphics Forum (Eurographics Proceedings) 27,* 2, 241–250.

Elber, G. and Cohen, E. 1990. Hidden Curve Removal for Free Form Surfaces. In *SIGGRAPH 1990 Proceedings*. Vol. 24. 95–104.

Gooch, B., Sloan, P.-P. J., Gooch, A., Shirley, P., and Riesenfeld, R. 1999. Interactive Techincal Illustration. In *Proceedings of the Symposium on Interactive 3D Graphics and Games 1999*.

Goodwin, T., Vollick, I., and Hertzmann, A. 2007. Isophote Distance: A Shading Approach to Artistic Stroke Thickness. In *Proceedings of the International symposium on Non-photorealistic animation and rendering 2007*. 53–62.

Heckbert, P. S. and Garland, M. 1999. Optimal triangulation and quadric-based surface simplification. *Computational Geometry Theory and Applications 14*, 49–65.

Hertzmann, A. and Zorin, D. 2000. Illustrating Smooth Surfaces. In *SIGGRAPH 2000 Proceedings*. 517–526.

Hyvärinen, A. 1999. Fast and Robust Fixed-Point Algorithms for Independent Component Analysis. *IEEE Transations on Neural Network 10,* 3, 626–634.

Interrante, V., Fuchs, H., and Pizer, S. 1995. Enhancing Transparent Skin Surfaces with Ridge and Valley Lines. In *Proceedings of the 6th conference on Visualization 1995*. 52–59.

James, D. L. and Fatahalian, K. 2003. Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics 22,* 3, 879–887.

James, D. L. and Twigg, C. D. 2005. Skinning mesh animations. *ACM Transactions on Graphics 24,* 3, 399–407.

Judd, T., Durand, F., and Adelson, E. 2007. Apparent Ridges for Line Drawing. *ACM Transactions on Graphics 26,* 3, 19.

Kalogerakis, E., Simari, P., Nowrouzezahrai, D., and Singh, K. 2007. Robust statistical estimation of curvature on discretized surfaces. In *Proceedings of the Symposium on Geometry Processing 2007*. 13–22.

Lee, Y., Markosian, L., Lee, S., and Hughes, J. F. 2007. Line drawings via abstracted shading. *ACM Transactions on Graphics 26,* 3, 18.

Lewis, J. P., Cordner, M., and Fong, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *SIGGRAPH 2000 Proceedings*. 165–172.

Markosian, L., Kowalski, M. A., Trychin, S. J., Bourdev, L. D., Goldstein, D., and Hughes, J. F. 1997. Real-Time Nonphotorealistic Rendering. In *SIGGRAPH 1997 Proceedings*. 415–420.

Meyer, M., Desbrun, M., Schröder, P., and Barr, A. H. 2002. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*. 35–57.

Mohr, A. and Gleicher, M. 2003. Building efficient, accurate character skins from examples. *ACM Transactions on Graphics 22,* 3, 562–568.

Ni, A., Jeong, K., Lee, S., and Markosian, L. 2006. Multi-scale line drawings from 3D meshes. In *Proceedings of the International Symposium on 3D Data Processing, Visualization and Transmission 2006*. 133–137.

Nocedal, J. and Wright, S. J. 1999. *Numerical Optimization*. Springer-Verlag.

Northrup, J. D. and Markosian, L. 2000. Artistic Silhouettes: A Hybrid Approach. In *Proceedings of the International symposium on Non-photorealistic animation and rendering 2000*. 31–38.

Nowrouzezahrai, D., Kalogerakis, E., and Fiume, E. 2009. Shadowing dynamic scenes with arbitrary BRDFs. In *Eurographics 2009 (To Appear)*.

Nowrouzezahrai, D., Kalogerakis, E., Simari, P., and Fiume, E. 2008. Shadowed relighting of dynamic geometry with 1d BRDFs. In *Eurographics 2008 Proceedings*.

Nowrouzezahrai, D., Simari, P., Kalogerakis, E., Singh, K., and Fiume, E. 2007. Compact and efficient generation of radiance transfer for dynamically articulated characters. In *Proceedings of the GRAPHITE 2007*. 147–154.

Ohtake, Y., Belyaev, A., and Seidel, H.-P. 2004. Ridge-valley lines on meshes via implicit surface fitting. *ACM Transactions on Graphics 23,* 3, 609–612.

Pauly, M., Keiser, R., and Gross, M. 2003. Multi-scale feature extraction on point-sampled surfaces. In *Eurographics 2003 Proceedings*. 281–289.

Polthier, K. 2002. Polyhedral surfaces of constant mean curvature. Ph.D. thesis, TU-Berlin.

Rusinkiewicz, S. 2004. Estimating Curvatures and Their Derivatives on Triangle Meshes. In *Proceedings of the International Symposium on 3D Data Processing, Visualization and Transmission 2004*. 486–493.

Rusinkiewicz, S. 2007. Trimesh2 library. http://www.cs.princeton.edu/gfx/proj/trimesh2/.

Rusinkiewicz, S., Burns, M., and DeCarlo, D. 2006. Exaggerated shading for depicting shape and detail. *Proc. SIGGRAPH 25,* 3, 1199–1205.

Saito, T. and Takahashi, T. 1990. Comprehensible Rendering of 3-D Shapes. In *SIGGRAPH 1990 Proceedings*. Vol. 24. 197–206.

Sloan, P.-P., Kautz, J., and Snyder, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH 2002 Proceedings*. 527–536.

Taubin, G. 1995. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proceedings of the Fifth International Conference on Computer Vision 1995*.

Thirion, J.-P. and Gourdon, A. 1996. The 3D marching lines algorithm. *Graphical Models and Image Processing 58,* 6, 503–509.

Vergne, R., Barla, P., Granier, X., and Schlick, C. 2008. Apparent relief: a shape descriptor for stylized shading. In *Proceedings of the International symposium on Non-photorealistic animation and rendering 2008*.

Wang, R. Y., Pulli, K., and Popović, J. 2007. Real-time enveloping with rotational regression. *ACM Transactions on Graphics 26,* 3, 73.

Weisberg, S. 2003. *Applied Linear Regression*, 3rd edition ed. Wiley/Interscience.

Winkenbach, G. and Salesin, D. H. 1996. Rendering Parametric Surfaces in Pen and Ink. In *SIGGRAPH 1996 Proceedings*. 469–476.

Yoshizawa, S., Belyaev, A., Yokota, H., and Seidel, H.-P. 2007. Fast and faithful geometric algorithm for detecting crest lines on meshes. In *Pacific Graphics 2007 Proceedings*. 231–237.

Yu, J., Yin, X., Gu, X., McMillan, L., and Gortler, S. 2007. Focal surfaces of discrete geometry. In *Proceedings of the Symposium on Geometry Processing 2007*. 23–32.

Received June 2008;