# EFFECTIVE METHODS FOR MANIPULATING AND RENDERING SKINNED MESHES

BY

WEI-WEN FENG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Doctoral Committee:

Professor Yizhou Yu, Chair and Director of Research
Professor David Forsyth
Professor John C. Hart
Liang Peng, Ph.D

# Abstract

Mesh skinning has been a widely applied method in games for skeleton driven character animation. A gaming character can be easily animated and deformed by transforming every vertex using a weighted sum of proxy bone transformations in the skeleton. Arbitrary deformations such as facial and cloth animation can also be effectively represented with a relatively large number of proxy bones. However, the large number of necessary proxy bones make it difficult to intuitively control the animation of deformable surfaces. This dissertation investigates novel methods for data-driven deformation and level-of-detail rendering of skinned mesh.

The first part of this dissertation includes methods to effectively generate new animation for a skinned mesh, which may be a non-articulated model or highly deformable surface such as face or clothes. We develop a regression framework to learn deformation styles for a skinned mesh from example configurations based on kernel Canonical Correlation Analysis (CCA). Without expensive non-linear optimization at run-time stage, the regression method is very efficient and can achieve real-time performance in generating novel mesh deformations. The regression method is also investigated to generate deformation details for dynamic clothes. A hybrid approach for cloth animation is developed to find a mapping between coarse deformations to high-resolution spatial details in a cloth model. The quality of regression model is improved by making separate regressions at different detail scale and by identifying suitable rotation-invariant quantities for regression. The run-time components are implemented efficiently on GPU to achieve an overall real-time performance on high-resolution cloth models.

In the second part, we focus on real-time rendering methods for skinned mesh. We introduce feature preserving triangular geometry images for level-of-detail rendering of skinned mesh. Triangular charts pack efficiently, simplify the elimination of T-junctions, arise naturally from an edge-collapse simplification base mesh, and layout more flexibly to allow their edges to follow curvilinear mesh features. By incorporating skinning weights and skinned bounding boxes into the representation, a view-dependent LOD scheme can be applied for rendering skinned meshes stored and rendered entirely on the GPU to maximize throughput. We also develop a data management scheme for precomputed radiance transfer to render skinned models with global shading effects.

*To my family and my wife*

# Acknowledgments

When I began graduate school six years ago, I couldn not imagine the day when I would complete my dissertation and have the opportunity to write this acknowledgement.

That day is finally here and, while I am proud of all that I have accomplished, I could not have done it without the support of others. So it is with my deepest gratitude and respect that I recognize below my many outstanding colleagues and friends - all of whom, in their unique way, played an important role in helping me complete my dissertation.

First, I would like thank my PhD advisor, Yizhou Yu. While in the midst of defining my research path, Yizhou accepted me as his PhD student and has since guided me through various research projects that were both challenging and rewarding. His patience and thoroughness have prepared me with first-class knowledge and skills and, were it not for him, I would not have been able to accomplish as much as I did, nor finish this dissertation.

I would like to thank my thesis committee - John C. Hart, David Forsyth and Liang Peng - for their valuable time and efforts in refining my dissertation. Many thanks to Liang for mentoring me during my summer internship at Rambus and for teaching me so much about GPU architecture. I would also like to thank Michael Garland for inspiring me to explore computer graphics in-depth through his Digital Geometry Processing course.

Additionally, I would like to thank Byung-Uck Kim, Yuntao Jia, Lin Shi and Nathan Wesling for working with me on various research projects and Shuo-Heng Chung, Tian Xia, Qing Wu, Binbin Liao, and Victor Lu for sharing experiences and insights throughout our many discussions. You all provided important perspectives that helped shape my dissertation.

Finally, I would like to thank my family for their support and encouragement throughout my graduate studies. Most importantly, however, I would like to thank my wife, Li-Ying Chuang, to whom I dedicate this dissertation. Meeting Li-Ying was the most wonderful thing that ever happened to me; she is my strength, my confidant and my best friend.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Part I : Data-Driven Skinned Mesh Deformation and Cloth Animation

Mesh skinning has been widely used in games for skeleton driven skin deformation. Linear blend skinning (SSD) is a popular technique that transforms every vertex using a weighted sum of nearby bone transformations. Bone influence weights can be automatically estimated from examples. Meanwhile, fully automatic techniques have been developed to compute proxy bones and their influence weights from mesh animations [7]. In [9], the extracted proxy bones were used for segmenting markers into rigid segments. High quality skin deformations can be reconstructed via a second-order skinning scheme followed by RBF-based interpolation of the residual errors. The algorithm in [6] also relies on proxy bones in addition to rotational regression to overcome the limitations of SSD. Although the technique in [7] is primarily targeted at articulated models, it can be extended to highly deformable surfaces by fitting a suitable number of bones [3]. Nevertheless, the relatively large number of necessary proxy bones make it hard to intuitively control the animation of deformable surfaces.

The research in the first part of this thesis tries to provide an intuitive control method on the class of deformable objects generated by skinning [7, 3], which can effectively deform a non-articulated model or highly deformable surface such as face or clothes. We present a regression framework to learn deformation styles for a skinned mesh from example configurations based on kernel Canonical Correlation Analysis (CCA). Instead of solving for new bone transformations via a relatively expensive nonlinear optimization such as methods previously investigated in the framework of MeshIK [10, 11], we adopt a learning approach which has an offline training stage in addition to the run-time animation stage. Because of the precomputation in the training stage, impressive real-time performance is made possible during the run-time stage. In addition, our method can accommodate a larger number of training examples to faithfully learn a deformation style because its run-time stage has linear scalability in terms of training examples.

Cloth animation is a challenging task for real-time computer graphics. Although efforts have been made to achieve real-time performance for simulating relatively low-resolution cloth, there is still rooms for improvement to generate higher quality dynamic folds and wrinkles. We propose a framework for adding these details in high-resolution cloth on a low-resolution cloth simulation in real-time. It relies on data-driven models to capture the relationship between

cloth deformations at two resolutions. Such data-driven models are responsible for transforming low-quality simulated deformations at the low resolution into high-resolution cloth deformations with dynamically introduced fine details. The key components in this framework is two non-linear mappings for mid-scale and fine-scale deformations in the high-resolution cloth model. The method relies on the carefully chosen rotation invariant quantities that are well suited for training high-quality data-driven models. We have also developed a fast collision detection and handling scheme based on dynamically transformed bounding volumes. The run-time stage can thus be efficiently implemented on programmable graphics hardware to achieve an overall real-time performance on high-resolution cloth models.

## 1.2 Part II : Real-Time Rendering for Skinned Mesh

Realistic rendering and high frame rates are two most important aspects in games. However, the two goals are often in conflict since increasing rendering quality implies using higher resolution models, which in turns hurts the frame rates. Level-of-detail control provides a more efficient resource management scheme by rendering a higher resolution model only when it is closer to the viewer. Traditional view-dependent LOD representations based on mesh simplification [12, 13, 14] with poor cache coherence. Geometry images [15] support efficient LOD display [16, 17, 18] by storing the mesh as a MIP-mapped texture image with better GPU cache coherence, but flattening a mesh into a single geometry image can create severe parametric distortion.

Most of the existing LOD methods have been developed for static meshes only and and simplification for deforming meshes has been less explored. The quadric error metric [19] can be extended [20, 21] for simplification of a mesh with multiple deformed poses into a pose-independent simplified mesh. Hierarchical face clustering has been performed in [22] to achieve pose-dependent simplification with higher visual quality for precomputed mesh deformation sequences. To our knowledge, there is no previous work dealing with level-of-detail representation and control for skinned meshes.

The research in the second part of this thesis is to augment geometry images with new support for the dynamic meshes found in modern videogames animated by linear-blend skinning deformations. We propose triangular geometry images as a single solution that combines the cache coherence of geometry images, the lower distortion of multiple charts, the straightforward downsampling of a simple chart shape, and a feature preserving layout. To represent a skinned mesh as geometry images, we store vertex skinning weights in addition to the (rest-pose) vertex positions in the geometry image, and downsample both for dynamic mesh LOD. We also surround each skinned triangle chart with an oriented bounding box whose corners are themselves skinned to deform with its contents. The screen size of a projected bounding box selects the optimal LOD resolution for each triangular chart. The result is a view-dependent LOD representation for both static and skinned meshes stored and rendered entirely on the GPU to maximize throughput.

High frame rates requirements in games also restrict the quality of realistic rendering. To achieve photo-realism, complicated light transport needs to be simulated for the effects such as glossy reflection, interreflection, shadowing and subsurface scattering under environment lighting. However, these simulations are not feasible for real-time rendering. Precomputed radiance transfer (PRT) provides the opportunity to produce compelling realism with global shading effects in real time. Originally developed for static scenes [23, 24], PRT has been subsequently extended to fixed animation sequences as well as deformable objects with shading effects caused by detailed surface features but without cast shadows [25]. It has also inspired techniques that produce soft shadows for dynamic scenes [26, 27] as well as algorithms for real-time lighting design [28] and cinematic relighting [29]. However, the generalization of PRT to dynamic scenes with global shading effects, such as interreflection and subsurface scattering, has not been very successful.

The last part of this thesis is to extend real-time PRT rendering to skinned mesh, which is commonly used in computer games. We take the example-based approach that draws many samples in the pose subspaces for a particular object and precomputes radiance transfer for them. During run-time, the example PRT matrices are interpolated from nearest neighbors in the pose space to produce rendering results for the new poses. Since we need to precompute radiance transfer for every sampled pose, resulting datasets reach hundreds of gigabytes, and are orders of magnitude larger than those for a static object. Thus we present effective clustering and compression schemes for precomputed radiance transfer matrices so that the aforementioned runtime data communication, decompression and interpolation can be performed efficiently and accurately. As a result, high-quality real-time rendering with global shading effects is achieved.

# Chapter 2

# Real-Time Data-Driven Skinned Mesh Deformation Using Kernel Canonical Correlation Analysis

Achieving intuitive control of animated surface deformation while observing a specific style is an important but challenging task in computer graphics. Solutions to this task can find many applications in data-driven skin animation, computer puppetry, and computer games. We present an intuitive and powerful animation interface to simultaneously control the deformation of a large number of local regions on a deformable surface with a minimal number of control points. Our method learns suitable deformation subspaces from training examples, and generate new deformations on the fly according to the movements of the control points. Our contributions include a novel deformation regression method based on kernel Canonical Correlation Analysis (CCA) and a Poisson-based translation solving technique for easy and fast deformation control based on examples. Our run-time algorithm can be implemented on GPUs and can achieve a few hundred frames per second even for large datasets with hundreds of training examples.

## 2.1 Introduction

Achieving intuitive control of animated surface deformation while observing a specific style is an important but challenging task. Very often, the surface undergoing deformation does not have an intrinsic skeleton. For example, creating interesting and meaningful facial expressions requires surface deformation induced by an orchestrated coordination of a number of muscles. An intuitive and powerful animation interface should be able to simultaneously control the deformation of a large number of local regions on such a deformable surface with a minimal number of control points (animation parameters). Solutions to this task can find many applications in data-driven skin animation and computer puppetry. This is especially true with the recent trend in computer game design, which makes in-game avatars and virtual environments subject to user-level control and customization.

With the rapid progress in data acquisition and physically based simulation techniques, one effective solution would be learning suitable deformation subspaces from acquired or simulated examples, and generate new deformations on the fly according to the translational movements of a sparse set of control points. Thus, one challenge we need to overcome in intuitive control of arbitrary deformations would be learning the mapping between sparse control point movements and the deformation of an entire surface with at least thousands of DOFs. Fortunately, deformations at

Figure 2.1: Novel deformations of various styles can be generated in real time with our data-driven method.

different regions of the surface might be highly correlated. For example, moving an eyebrow should not only deform the eye, but also affect the cheek and mouth in a meaningful expression. Moving one region of clothing might create wrinkles on other regions. Therefore, it is crucial to learn these potentially nonlinear relationships from the examples and let them guide novel deformations.

Another challenge in achieving our goal is the performance requirement in real-time applications such as gaming. An update rate of thirty frames per second is the basic requirement in gaming applications which have a large portion of their system resources devoted to AI and rendering instead of animation. Thus the run-time algorithm for deformations needs to be extremely fast and reach a frame rate much higher than 30fps. Preferably it should make use of modern GPU's parallel processing power. This consideration implies that the framework should not involve sophisticated run-time computation that is hard to parallelize effectively on GPUs.

In this paper, we present a statistically based framework to learn deformation styles for a skinned mesh from example configurations. Our contributions include a novel deformation regression method based on kernel Canonical Correlation Analysis (CCA) [30, 31] and a Poisson-based translation solving technique for easy and fast deformation control based on training examples. Our run-time algorithm can be implemented on GPUs and can achieve a few hundred frames per second even for large datasets with hundreds of training examples.

In our method, we first extract from example meshes a sparse set of control points as well as a skinned mesh with bones and bone influence weights. The goal of a subsequent learning process is to capture connections between control points and bone deformations in the example data and train a predictor generating novel bone deformations

from control point movements. To achieve this, we first perform nonlinear model reduction by applying kernel CCA to find a pair of nonlinear subspaces that maximize the correlation between the pairs of example configurations. The original data are then projected into the subspaces to obtain their reduced coordinates. Standard regression techniques can be performed on the reduced coordinates to train a desired predictor.

At run-time, the deformation predictor is used to generate novel bone deformations according to control point movements. The prediction process is reformulated into matrix-vector multiplications and kernel transformations, both of which can be efficiently implemented on GPUs for parallel processing. For deformations with large bone rotations, the predicted bone translations can be improved using a real-time Poisson-based linear solver which only requires a single matrix-vector multiplication if the pseudo inverse of its coefficient matrix has been precomputed. The resulting bone transformations from previous steps can be directly used in a GPU-based skinning algorithm. High performance has therefore been achieved in all of our experiments.

## 2.2 Related Work

Mesh skinning has been widely used in games for skeleton driven skin deformation. Linear blend skinning (SSD) is a popular technique that transforms every vertex using a weighted sum of nearby bone transformations. Bone influence weights can be automatically estimated from examples. Direct blending of rotation matrices by SSD suffers from collapsing joints and the "candy wrapper" artifact when joints are overly bent or twisted. Techniques have been proposed to compensate such inaccuracies. The method in [20] can well model example-based muscle deformations by adding extra joints around the area with large skinning errors. Their joint placement is compact and can resolve artifacts from SSD with little performance impact. Dual-quaternion skinning [32] have also demonstrated good results in terms of both deformation quality and performance. In addition to these fast skinning algorithms, there exist more expensive techniques [33, 34, 35] that effectively integrate recent mesh deformation techniques to generate high quality skin deformations.

Meanwhile, fully automatic techniques have been developed to compute proxy bones and their influence weights from mesh animations [7]. In [9], the extracted proxy bones were used for segmenting markers into rigid segments. High quality skin deformations can be reconstructed via a second-order skinning scheme followed by RBF-based interpolation of the residual errors. The algorithm in [6] also relies on proxy bones in addition to rotational regression to overcome the limitations of SSD. Although the technique in [7] is primarily targeted at articulated models, it can be extended to highly deformable surfaces by fitting a suitable number of bones [3]. Nevertheless, the relatively large number of necessary proxy bones make it hard to intuitively control the animation of deformable surfaces.

Example-based mesh deformation driven by a small number of translational handles instead of a skeleton have been previously investigated in the framework of MeshIK [10, 11], where deformation gradients are interpolated from

example poses and new vertex positions or new bone transformations are solved via a relatively expensive nonlinear optimization, which simultaneously involves all examples. In contrast, this paper adopts a learning approach which has an offline training stage in addition to the run-time animation stage. Because of the precomputation in the training stage, impressive real-time performance is made possible during the run-time stage. In addition, our method can accommodate a larger number of training examples to faithfully learn a deformation style because its run-time stage has linear scalability in terms of training examples while the time complexity of the algorithm in [11] is a cubic polynomial of the number of example poses. Thus, our method is better suited for real-time data-driven animation.

Facial expressions are difficult to animate due to correlated deformations in multiple regions, the variety of expressions and the existence of small features such as wrinkles. Blendshape face is a popular real-time technique for facial animation. By establishing relationships between motion capture data and blendshape weights, new facial animations can be generated from facial MoCAP data [36, 37]. However, unlike our method, these techniques require relatively dense marker movements. In FaceIK [2], spatially varying blending weights based on control point positions are computed to generate novel expressions from multiple example faces. In Face poser [38], a nonlinear optimization is formulated in a maximum a posteriori (MAP) framework to find optimal PCA coefficients. These face animation techniques are relatively expensive and are not well suited for real-time applications.

Cloth is another challenge for both deformation acquisition and creation. A pattern-based cloth capturing method has been proposed in [39] to reconstruct a deforming cloth mesh from multiple views, but a real-time interface is still needed to create potentially novel animations consistent with the style of the captured data. On the other hand, the main objective of the editing techniques in [40, 41, 42] was to modify an existing mesh animation but not to create a novel one according to control point movements.

This paper also shares the same motivation with previous work on skeletal animation driven by low-dimensional control signals [43, 44]. Specifically, the method in [43] performs global nonlinear model reduction using a particular form of Gaussian processes. At run-time, a relatively expensive nonlinear optimization is still necessary to reconstruct a complete skeletal configuration from low-dimensional signals. In comparison, our method performs local nonlinear model reduction for each proxy bone using kernel CCA and does not require nonlinear optimization at the run-time. Note that linear CCA has been applied in [45] for aligning two serial signals in skeletal animation.

## 2.3 Overview

Our goal is to achieve real-time example-based surface deformation and animation using sparse control points. The original surface can be an arbitrary deformable surface without a skeleton. Given a set of surface deformation examples, we first build two different abstractions for the surface. The first abstraction is designed as an animation interface. It is a very sparse set of control points whose locations on the surface are such that they can unambiguously convey

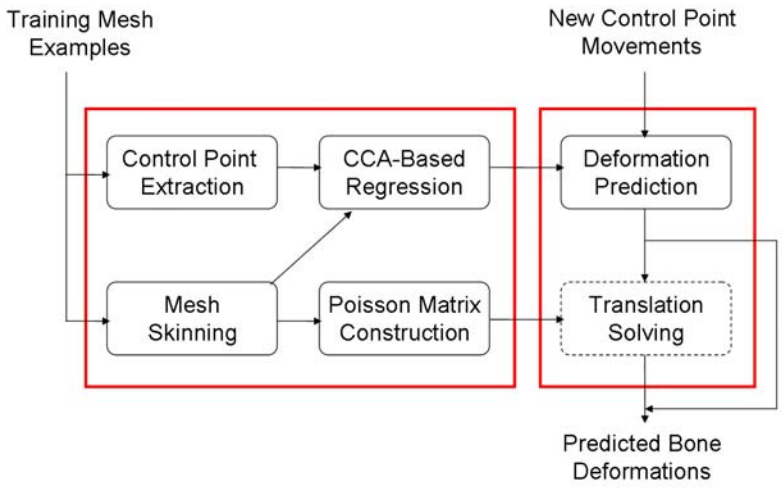



Figure 2.2: The work flow of our method.

the intended deformation. The second abstraction is a sparse set of abstract bones whose deformation parameters are collectively used for generating the deformation of every vertex on the surface in real time. To build connections between these two abstractions, a crucial preprocessing step is to train deformation predictors $\mathbf{d}(\mathbf{c})$ from pairs of configurations of control points and bone deformations, $(\mathbf{c}, \mathbf{d})$. At run-time, these predictors effectively generate new bone deformations, and in turn new surface deformations faithful to the style in the training examples, from novel control point movements. Since the prediction models are precomputed, generating new deformations at run-time is efficient and uses little computational resource. The work flow of our system is summarized in Figure 3.2.

**Training Stage.** Given meshes with $n_p$ different deformations, we identify several mesh vertices in highly deformable regions and use them as the control points. We adapt the method in [46] to guide our control point selection. The method works by performing principal component analysis (PCA) on vertex positions from mesh deformation examples to obtain a set of bases. Varimax rotation is used to rotate each PCA basis vector into a more localized version. Two representative points with largest differences are then selected automatically from each rotated basis vector. Finally, we interactively choose $n_c$ samples from these representative points that are placed near the semantic features of the input mesh such as eyes or the mouth. These points are then concatenated together to form the vector $\mathbf{c}$. Note that user interaction is only for choosing one of multiple points automatically identified near the same semantic feature and can be finished in a few minutes.

We also build a skinned mesh which includes a set of bone deformations and the bone influence weights for each vertex. The skinned mesh is generated by grouping original triangles with similar transformations into the same bone [6]. The error $E_{A \to B}$ of joining bone $A$ to bone $B$ is defined to include not only vertex prediction errors, induced by applying $B$'s transformation to $A$'s vertices, but also edge prediction errors in terms of both edge orientation and length. Each bone acts as an abstract representative for transformations, and its influence weights on a vertex are then

obtained by minimizing the fitting error of vertex positions in all examples. In terms of bone transformations, we choose to fit a rigid transformation in the form of a dual-quaternion using the method in [3].

Once the set of control point sequences and corresponding deformations have been collected for each bone from the examples, we perform nonlinear model reduction using kernel CCA [30, 31] and build a deformation predictor using linear regression. We first transform every data pair,$(\mathbf{c}, \mathbf{d})$, into reduced coordinates $(\mathbf{c_r}, \mathbf{d_r})$ by performing kernel CCA, which finds pairs of projection bases which maximize the correlation between $\mathbf{c_r}$ and $\mathbf{d_r}$. We then apply linear regression on the reduced coordinates $(\mathbf{c_r}, \mathbf{d_r})$ to compute the deformation predictor that maps $\mathbf{c_r}$ to $\mathbf{d_r}$. The mapping is in the form of a regression matrix, and can be computed by minimizing least-square errors. Both the CCA bases and regression matrix are computed for each bone in the skinned mesh.

**Run-time Stage.** At the run-time, the CCA bases are used to transform new control point coordinates into reduced coordinates. The regression matrix is then applied to the reduced coordinates as the predictor to obtain new dual quaternion transformations for each bone. For highly deformable meshes, instead of using the predicted bone translations, we recompute them by solving the Poisson equation [47] in real time to distribute errors more uniformly over the entire mesh as well as to satisfy positional constraints for better user control. Note that we still use the rotational part of the prediction as usual. Most of the run-time process in our method can be formulated as simple matrix-vector multiplications plus kernel function evaluations, which can be implemented on GPUs very efficiently. The run-time performance of our method can achieve hundreds of frames per second in our experiments.

## 2.4 CCA Based Regression

We apply a statistically based method called *canonical correlation analysis* [30] to perform model reduction and obtain optimal basis pairs that reveal the functional dependency between control points and bone deformations. While principal component analysis (PCA) performs feature extraction for a single set of variables, CCA extracts pairs of features that yield maximum correlation between two sets of variables and, thus, is better suited as a preprocessing step for regression. We choose to use CCA because of its ability to capture data dependency while avoiding overfitting. If we directly apply linear regression or kernel-based regression methods such as RBFs on the input data, the resulting predictor can fit the example data very well, but there is the risk of overfitting if the input examples are sparse. The obtained predictor might not be able to learn the essence of the actual deformation model, especially in terms of nonlinear facial deformations, and produce unnatural novel deformation sequences (Figure 2.3). Moreover, for example pairs that are nonlinearly correlated, the kernel trick can be applied in the CCA formulation to establish nonlinear dependency between variables, which enrich the classes of deformation our system can handle.

In the following subsections, we review the mathematical background of CCA, and describe the details of our

regression method.

### 2.4.1 Canonical Correlation Analysis

Given two sets of variables $\{\mathbf{c}, \mathbf{d}\}$ with $\mathbf{c} \in R^n$ and $\mathbf{d} \in R^m$, CCA finds pairs of bases $\{\mathbf{u_c}, \mathbf{u_d}\}$ such that the projections $c_r = \mathbf{u_c}^T\mathbf{c}$ and $d_r = \mathbf{u_d}^T\mathbf{d}$ have their correlation $\rho$ maximized. Here we follow the derivation from [31]. Specifically,

$$\rho = \frac{E[c_r d_r]}{\sqrt{E[c_r^2]E[d_r^2]}} = \frac{E[\mathbf{u_c}^T\mathbf{cd^T u_d}]}{\sqrt{E[\mathbf{u_c}^T\mathbf{cc^T u_c}]E[\mathbf{u_d}^T\mathbf{dd^T u_d}]}}. \tag{2.1}$$

The maximization can be formulated as :

$$\max_{\mathbf{u_c},\mathbf{u_d}} \rho = \max_{\mathbf{u_c},\mathbf{u_d}} \frac{\mathbf{u_c}^T\Sigma_{cd}\mathbf{u_d}}{\sqrt{\mathbf{u_c}^T\Sigma_{cc}\mathbf{u_c}\mathbf{u_d}^T\Sigma_{dd}\mathbf{u_d}}} \tag{2.2}$$

where $\Sigma_{cd} = cov(\mathbf{c}, \mathbf{d})$ is the cross-covariance matrix of $\mathbf{c}, \mathbf{d}$, and $\Sigma_{cc}, \Sigma_{dd}$ are defined similarly. The solution for $\{\mathbf{u_c}, \mathbf{u_d}\}$ can be obtained via singular value decomposition (SVD) of the matrix :

$$\Sigma_{cc}^{-\frac{1}{2}}\Sigma_{cd}\Sigma_{dd}^{-\frac{1}{2}} = \mathcal{U}\mathcal{D}\mathcal{V}^{\mathcal{T}} \tag{2.3}$$

where $\mathcal{U}, \mathcal{D}, \mathcal{V}$ are the resulting decomposition of SVD. The $i$-th basis pair can be obtained by computing

$$\mathbf{u_c}^i = \Sigma_{cc}^{-\frac{1}{2}}\mathcal{U}^i \tag{2.4}$$

and

$$\mathbf{u_d}^i = \Sigma_{dd}^{-\frac{1}{2}}\mathcal{V}^i \tag{2.5}$$

where $\mathcal{U}^i$ and $\mathcal{V}^i$ are the $i$-th column of matrices $\mathcal{U}$ and $\mathcal{V}$. Note that there are a maximum number of $\min(m, n)$ basis pairs, where $\min(m, n)$ is equal to the smaller dimension of $\mathbf{c}$ and $\mathbf{d}$.

### 2.4.2 The Kernel Trick

Instead of representing CCA bases in a linear subspace, we can also construct a nonlinear version of the algorithm via kernel functions. The kernel method was originally used to extend a support vector machine (SVM) to its non-linear version. It works by mapping the original data into a higher dimensional feature space and solving a corresponding nonlinear version of the problem in that feature space. Suppose $\phi : R^s \rightarrow R^t, t > s$ is a mapping that transforms $\mathbf{x}$ into the feature space. A kernel function, $k(\mathbf{x}, \mathbf{y})$, can be used to define the dot product in the feature space. That is, $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T\phi(\mathbf{y})$. In many cases, the kernel function has a simple closed-form expression even when the mapping

itself is hard to formulate explicitly. The kernel trick means if the original problem can be reformulated to depend only on the dot product of the original data, the nonlinear version of the problem can be formulated to depend only on the kernel function. As a simple example, let $\phi(\mathbf{x})$ be a mapping which transforms a vector $\mathbf{x} = (x_1, x_2)$ into the vector of all second degree monomials $(x_1^2, x_2^2, x_1x_2, x_2x_1)$. We can clearly see that $\phi(\mathbf{x})^T\phi(\mathbf{y}) = k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2$. Here $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2$ is the second degree polynomial kernel.

The nonlinear version of CCA in our current context can be derived via the kernel trick. Given $n_p$ pairs of example data $\{\mathbf{c}^i, \mathbf{d}^i\}_{i=1}^{n_p}$ with $\mathbf{c}^i \in R^n$ and $\mathbf{d}^i \in R^m$, we define $\mathbf{C} = (\mathbf{c^1} \dots \mathbf{c}^{n_p})$ and $\mathbf{D} = (\mathbf{d^1} \dots \mathbf{d}^{n_p})$ as data matrices with their $i$-th columns being $\mathbf{c}^i$ and $\mathbf{d}^i$, respectively. The covariance matrices for nonlinearly mapped data can be written as :

$$\Sigma_{cd} = \phi(\mathbf{C})\mathbf{D}^T, \Sigma_{cc} = \phi(\mathbf{C})\phi(\mathbf{C})^T, \Sigma_{dd} = \mathbf{D}\mathbf{D}^T,$$

where $\phi(\mathbf{C})$ is a nonlinear version of matrix $\mathbf{C}$ by applying the mapping $\phi$ on each column of $\mathbf{C}$. We choose to transform only the input matrix $\mathbf{C}$ instead of both $(\mathbf{C}, \mathbf{D})$ because a nonlinear mapping of $\mathbf{D}$ would result in a nonlinear reconstruction at run time from reduced coordinates to original ones, which would be expensive considering the performance requirement of our method. Because the basis pair $\{\mathbf{u_c}, \mathbf{u_d}\}$ always lies in the span of the mapped data $\{\phi(\mathbf{C}), \mathbf{D}\}$, we can further express a pair of bases $\{\mathbf{u_c}, \mathbf{u_d}\}$ as $\mathbf{u_c} = \phi(\mathbf{C})\mathbf{f_c}$ and $\mathbf{u_d} = \mathbf{D}\mathbf{f_d}$, where $\mathbf{f_c}, \mathbf{f_d} \in R^{n_p}$ are coefficient vectors with their dimensions equal to the number of example pairs. Therefore we can write the nonlinear version of (2.2) using the kernel as follows:

$$\max_{\mathbf{f_c}, \mathbf{f_c}} \rho = \max_{\mathbf{f_c}, \mathbf{f_c}} \frac{\mathbf{f_c}^T K_c K_d \mathbf{f_d}}{\sqrt{\mathbf{f_c}^T (K_c)^2 \mathbf{f_c} \mathbf{f_d}^T (K_d)^2 \mathbf{f_d}}} \tag{2.6}$$

where $K_c = \phi(\mathbf{C})^T\phi(\mathbf{C})$ and $K_d = \mathbf{D}^T\mathbf{D}$. Note that the entries of $K_c$ can be computed by the kernel function $k_c$ instead of by $\phi(\mathbf{c})$ explicitly. This is the dual form of (2.2), and can be solved in a similar manner via SVD as (2.4) and (2.5).

Our Results PCA+RBF Regression RBF Regression

Figure 2.3: A comparison among our method, regression based on both PCA and RBFs, and direct RBF-based regression. The top row shows the fitting quality of a training example, and the bottom row shows a predicted deformation once the user pulls a control point. The colored spheres are the control points, with the red one indicating the point being edited by the user. All three methods can fit the training data very well without noticeable differences. However, RBF and joint PCA-RBF regression fail to generalize beyond original examples and produce distorted results.

Suppose we retain $n_f$ pairs of coefficient vectors from SVD to form two matrices, $\mathbf{F_c} = (\mathbf{f_c}^1 \dots \mathbf{f_c}^{n_f})$ and $\mathbf{F_d} = (\mathbf{f_d}^1 \dots \mathbf{f_d}^{n_f})$. The projection of an input $\mathbf{c}$ onto the basis $\mathbf{u_c}^i$ can be computed as

$$\phi(\mathbf{c})^T \mathbf{u_c}^i = \sum_{j=1}^{n_p} f_{cj}^i \phi(\mathbf{c})^T \phi(\mathbf{c}^j) = \sum_{j=1}^{n_p} f_{cj}^i k_c(\mathbf{c}, \mathbf{c}^j).$$

Thus, the vector of reduced coordinates, $\mathbf{c_r} \in R^{n_f}$, can be expressed in a matrix-vector form as follows.

$$\mathbf{c_r} = \mathbf{F_c}^T \xi_{\mathbf{c}}, \tag{2.7}$$

where $\xi_{\mathbf{c}} \in R^{n_p}$ is the kernelized vector whose $j$-th entry is $k_c(\mathbf{c}, \mathbf{c^j})$.

### 2.4.3 Regression

We use regression to build up connections between control point coordinates and bone deformations. We use dual quaternions for representing bone deformations [32]. A dual quaternion $\mathbf{q_0} + \epsilon \mathbf{q}_\epsilon$ represents a rigid transformation. It involves two classic quaternions, $\mathbf{q_0}$ and $\mathbf{q}_\epsilon$, therefore, a bone deformation involves eight parameters, $\mathbf{d} \in R^8$. Linear blending of dual quaternions can be applied to blend multiple corresponding rigid transformations, and the resulting dual quaternion still represents a rigid transformation. We choose to represent a bone deformation using a dual quaternion instead of an affine transformation matrix because it has better interpolation property than the transformation matrix and can avoid the "candy wrapper" artifact [3] when used for skinning. It also has fewer parameters than an affine transformation matrix, and is therefore a more suitable choice for regression.

Regression is performed once for each bone. The outcome of regression is a predictor, $\mathbf{d}(\mathbf{c})$, that is able to predict a bone deformation, $\mathbf{d}$, given the concatenation of all control point coordinates, $\mathbf{c} \in R^{n_c \times n_{DOF}}$ where $n_c$ is the number of control points and $n_{DOF}$ is the degree of freedom of each control point. Note that a control point can have at most 3 DOFs in its 3D position. Given $n_p$ surface deformation examples, we first extract from each example a pair of data, $(\mathbf{c}^i, \mathbf{d}^i)$ for the bone. Kernel CCA is then performed on all extracted data pairs to obtain the set of $n_f$ CCA bases and the collection of reduced coordinates, $\mathbf{C_r} \in R^{n_f \times n_p}$ and $\mathbf{D_r} \in R^{n_f \times n_p}$, of all data pairs after projection onto the bases. While PCA can only have a common set of bases for all bones, there is a distinct set of CCA bases specifically tailored for each bone. Given those pairs of reduced coordinates, we compute an optimal predictor $\mathbf{B_c} \in R^{n_f \times n_f}$ by performing the following linear regression:

$$\min_{\mathbf{B_c}} \sum_{i=1}^{n_p} \|\mathbf{B_c} \mathbf{c_r}^i - \mathbf{d_r}^i\|^2 \tag{2.8}$$

where $\mathbf{c_r}^i$ and $\mathbf{d_r}^i$ are the $i$-th column of $\mathbf{C_r}, \mathbf{D_r}$, respectively. We choose a simple linear predictor because the

Rest Pose Linear Kernel Non-linear Kernel

Figure 2.4: A comparison of kernel functions in our CCA-based model reduction. Predictors based on nonlinear polynomial kernels produce higher quality deformations without artifacts when compared with results based on the linear kernel.

example data pairs have already been nonlinearly transformed using kernel CCA to maximize their dependency and linear regression can already produce accurate predictions in all our experiments.

Since the output deformation from our predictor is in the form of reduced coordinates as well, they need to be used to further reconstruct the eight parameters of a dual quaternion. Different from PCA, the bases generated by CCA are not necessarily orthogonal to each other. Therefore we cannot directly reconstruct these parameters by considering the reduced coordinates as weights over the CCA bases. In theory, dual bases of the CCA bases need to be computed to achieve accurate reconstruction. Since we do not kernelize the example deformations in our CCA formulation, the matrix of dual bases can be represented as a linear mapping $\mathbf{H_d} \in R^{8\times n_f}$ which transforms the reduced coordinates $\mathbf{D_r}$ to optimally approximate the example deformations $\mathbf{D}$. Thus, it is the solution of the following linear least-squares problem:

$$\min_{\mathbf{H_d}} \sum_i \|\mathbf{H_d}\mathbf{d_r}^i - \mathbf{d}^i\|^2. \tag{2.9}$$

Note that since the matrices $\mathbf{F_c}, \mathbf{B_c}, \mathbf{H_d}$ are consecutive linear mappings, we can concatenate them into a single linear operator $\mathbf{M_b} = \mathbf{H_d}\mathbf{B_c}\mathbf{F_c}$ that predicts the resulting deformation $\mathbf{d}$ given the kernelized vector $\xi_\mathbf{c}$ from the input $\mathbf{c}$. In our run-time implementation, this combined predictor $\mathbf{M_b}$ is used in place of $\mathbf{F_c}, \mathbf{B_c}$ and $\mathbf{H_d}$ for efficient execution.

We have verified the merit of CCA-based nonlinear model reduction by comparing our results with regression based on radial basis functions (RBF). We tested both direct RBF regression and RBF-based regression from reduced coordinates obtained by applying PCA to control point configurations. In both schemes, a regression model is computed by minimizing least-squares errors. As we can see from Figure 2.3, although the fitting error for the training

Without Poisson With Poisson

Figure 2.5: Deformation prediction without an additional translation solver may yield distorted results. However, once our Poisson-based translation solver has been applied, the resulting deformation becomes natural without artifacts.

examples is small in all of the methods, the direct regression scheme tends to overfit the training data and produce unnatural predicted results. On the other hand, while PCA-based model reduction can prevent overfitting, there are still distortions in the predicted results. This is because the same set of PCA bases are used in the regression for all bone transformations, and it cannot necessarily give rise to a good functional dependency for every bone.

We have also compared the quality of predicted deformations from linear and polynomial kernels in CCA-based model reduction. As shown in Figure 2.4, for certain deformation styles such as facial animation, nonlinear kernels yield more natural results.

## 2.5 Poisson Translation Solver

At run-time, the transformation for each bone is generated independently using the predictor learned in the previous section. Usually we can directly use this resulting transformation for dual quaternion skinning. However, when nearby bones undergo very different 3D rotations, small prediction errors in translation may be amplified in the visual results and create artifacts in regions jointly controlled by multiple bones. We solve this problem by computing a new set of translations for the bones in a Poisson formulation. The motivation is that while bone rotations define deformed local shapes, it is the bone translation that integrates these local shapes together. By solving for new translations, we can ensure that the resulting transformations will be consistent among nearby bones while preserving the deformed local shapes from our prediction. In addition, control point positions can be enforced as soft constraints in the Poisson formulation.

As a preprocessing step for the Poisson solver, we perform linear blend skinning for both vertices and edges on the deformable mesh surface. Let $v_i^0$ be the rest position of the $i$-th vertex $v_i$, $\{w_i^b\}_{b=1}^{n_b}$ be its skinning weights for all bones, and $\{\mathbf{R_b}, \mathbf{t_b}\}_{b=1}^{n_b}$ be the rotation matrices and translation vectors of all bones, the skinned vertex position $v_i^s = \sum_b \mathbf{w}_i^b(\mathbf{R_b}v_i^0 + \mathbf{t_b})$. We also fit a set of skinning weights $\{\hat{w}_k^b\}_{b=1}^{n_b}$ for each edge $e_k = v_{k0} - v_{k1}$ in the mesh. We treat $\hat{w}_k^b$ as a skinning weight for edge $e_k$ analogous to $w_i^b$ for $v_i$. Specifically, we compute $\{\hat{w}_k^b\}_{b=1}^{n_b}$ by minimizing the following fitting error of the edge:

$$\min_{\hat{\mathbf{w}}_\mathbf{k}} \sum_{j=1}^{n_p} \|e_k^j - \sum_{b=1}^{n_b} \hat{w}_k^b(\mathbf{R_b}{}^j e_k^0)\|^2 \tag{2.10}$$

Note that we use the same set of bone transformations when fitting weights for edges. Since the edge orientation is a vector, we have removed the translation component in the error formulation.

Our Poisson-based translation solver is formulated as an optimization problem which minimizes the differences of two edge predictions. The objective function of this minimization is formulated as follows.

$$\min_{\mathbf{t}} \sum_{k=1}^{n_e} \|(v_{k0}^s - v_{k1}^s) - e_k^s\|^2 + \beta \sum_{l=1}^{n_c} \|v_{i_l}^s - c_l\|^2 \tag{2.11}$$

where $\beta$ is a weighting factor for positional constraints, $e_k^s = \sum_b \hat{w}_k^b(\mathbf{R_b}e_k^0)$ is the skinned edge representation, $c_l$ represents the current position of the $l$-th control point, $v_{i_l}$ is the vertex corresponding to the $l$-th control point, and $n_e$ is the number of edges. Here the rotation matrix $\mathbf{R_b}$ is directly from prediction, and new translation vectors $\mathbf{t} = \{t_0 \dots t_{n_b}\}$ are sought as the solution of this minimization. It can be easily verified that (2.11) is actually a linear least-squares problem, $\min_\mathbf{t} \|\mathbf{At} - \mathbf{T}\|^2$, where $\mathbf{T}$ contains all bone rotation matrices and control point positions, and matrix $\mathbf{A}$ can be computed from rest-pose vertex positions and the skinning weights for both vertices and edges. Since entries in matrix $\mathbf{A}$ do not change at run-time, we precompute its pseudo inverse $\mathbf{P} = (\mathbf{A}^\mathbf{T}\mathbf{A})^{-1}\mathbf{A}^T$ in a preprocessing stage. Therefore the solution of (2.11) can be directly found by a matrix-vector multiplication $\mathbf{t} = \mathbf{PT}$. This simple solution technique makes it straightforward to implement on GPUs.

The insightful reader might notice that dual-quaternion skinning has been replaced by linear blend skinning in our Poisson formulation. The reason for this approximation is that the evaluation of dual quaternions makes the solution nonlinearly depend on the predicted rotations. Thus the problem becomes a nonlinear optimization, which is much more expensive to solve at run-time. In practice, we have found the resulting translations from this approximation works very well in all our experiments.

To validate our translation solving process, we have compared the predicted deformations from novel control point movements with and without the translation solver. As we can see in Figure 2.5, the predicted deformation is much more distorted without the Poisson-based translation solver because of the inconsistency of the translations among

nearby bones. On the other hand, new translations obtained from our translation solver can better position nearby bones and are much more natural visually.

**Remark** A solution in the form of matrix-vector multiplication for the Poisson problem has previously been used for solving for transformations of a reduced deformable model [6]. However, their method was derived in the context of deformation gradients and both the new matrix transformations and translations were solved. In our method, we propose a novel view of the problem by treating edges as additional elements for skinning, and therefore do not require any specific treatment for deformation gradients or recomputing the rotation matrices.

## 2.6 GPU Implementation

Since all the run-time components can be reduced to linear operations, we can implement most of them efficiently on GPUs. During each frame, the data sent to the GPU from CPU only includes the current control point coordinates, which have less than 150 bytes in all of our examples. Both the deformation prediction and translation solving processes are performed entirely on GPUs.

The overall GPU implementation can be separated into three stages: two matrix-vector multiplications for CCA-based prediction, one matrix-vector multiplication for solving translations, and a dual-quaternion skinning step in a vertex shader. In both prediction and translation solving steps, there are additional kernel transformations and quaternion conversion for each bone. But overall the computation is dominated by straightforward matrix-vector multiplications. In our implementation, we use CUDA [48], a general GPU programming framework by nVidia, for the first two GPGPU stages.

### 2.6.1 Deformation Prediction

Given uploaded control point coordinates $\mathbf{c}$, we first compute the kernelized vector $\xi_{\mathbf{c}}$ from data matrix $\mathbf{C}$, which is defined at the end of Section 2.4.2. The predicted deformation $\mathbf{d_b}$ for bone $b$ can then be obtained via a matrix-vector multiplication $\mathbf{d_b} = \mathbf{M_b}\xi_{\mathbf{c}}$, where $\mathbf{M_b}$ is defined in Section 2.4.3 as the product of multiple matrices. For computational efficiency, we concatenate $\mathbf{M_b}$ from all bones into one large matrix $\mathbf{M}$, and perform the multiplication only once per pass. Note that both $\mathbf{M}$ and $\mathbf{C}$ can be precomputed and preloaded to GPU before the run-time process.

### 2.6.2 Translation Solving

Once we have obtained the predicted dual quaternions, we convert their non-dual parts into rotation matrices and concatenate them into a single vector $\mathbf{T}$. Similar to the previous stage, we can precompute and preload the pseudo inverse matrix $\mathbf{P}$ to the GPU memory. The translations $\mathbf{t}$ can then be computed directly by $\mathbf{t} = \mathbf{PT}$. Once we have obtained $\mathbf{t}$, we use it to generate a new set of dual quaternions $\mathbf{d}'$ by updating the translation part of $\mathbf{d}$ with $\mathbf{t}$. We map

the resulting dual quaternions $\mathbf{d}'$ into an OpenGL texture buffer to make it available in the skinning stage.

### 2.6.3 DQ-Palette Skinning

The last step of our GPU implementation performs dual-quaternion skinning [32] to actually deform the mesh. Dual-quaternion skinning for vertex $v_i$ is formulated as

$$\hat{v_i^s} = (\sum_{b=1}^{n_b} w_i^b \mathbf{d_b}) \hat{v_i^0} (\overline{\sum_{b=1}^{n_b} w_i^b \mathbf{d_b}})^{-1}$$

where $w_i^b$ is a vertex skinning weight, $\bar{\mathbf{d}} = \mathbf{q_0} - \epsilon \mathbf{q}_\epsilon$ is the dual quaternion conjugation, $\mathbf{d}^{-1} = \mathbf{d}^* = \mathbf{q}_\mathbf{0}^* + \epsilon \mathbf{q}_\epsilon^*$ is the inverse of unit dual-quaternion $\mathbf{d}$ with $\|\mathbf{d}\| = 1$, and $\hat{v} = 1 + \epsilon(v_x i + v_y j + v_z k)$ is the dual quaternion representation for vertex $v = (v_x, v_y, v_z)$.

We perform the above skinning in a vertex shader in an intermediate pass before normal computation and shading. The rest-pose vertex positions and their skinning weights are preloaded to GPU as a Vertex Buffer Object (VBO). In the vertex shader, we compute at each vertex a weighted sum of revised dual quaternions from the previous pass. The interpolated dual quaternion is then applied to the vertex to obtain its deformed position. In the final rendering pass, new vertex normal vectors are computed on the fly based on deformed positions from previos stages. This results in more accurate normal and improve the shading results.

## 2.7 Experimental Results

We chose three different types of example deformations, including facial animation, articulated mesh animation, and secondary deformation of clothing driven by underlying articulated motion, for our experiments. Among them, the facial animation and clothing deformation examples were real-world data acquired using computer vision techniques [2, 39]. In the CCA-based regression stage, we chose the fifth degree inhomogeneous polynomial kernel, $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + c)^5$, for facial animation, and the Gaussian kernel for other deformations. The differences between these two kernel functions are not significant in predicted results. In our experiments, the Gaussian kernel worked well for all of our examples, while the polynomial kernel produced slightly better result for face demo.

The parameters of the kernel functions are estimated automatically. For the Gaussian kernel, its standard deviation is set to the standard deviation of control point positions from the training examples. In the chosen polynomial kernel, $c$ is set to the average norm of control points scaled by a constant factor (0.1 in our case).

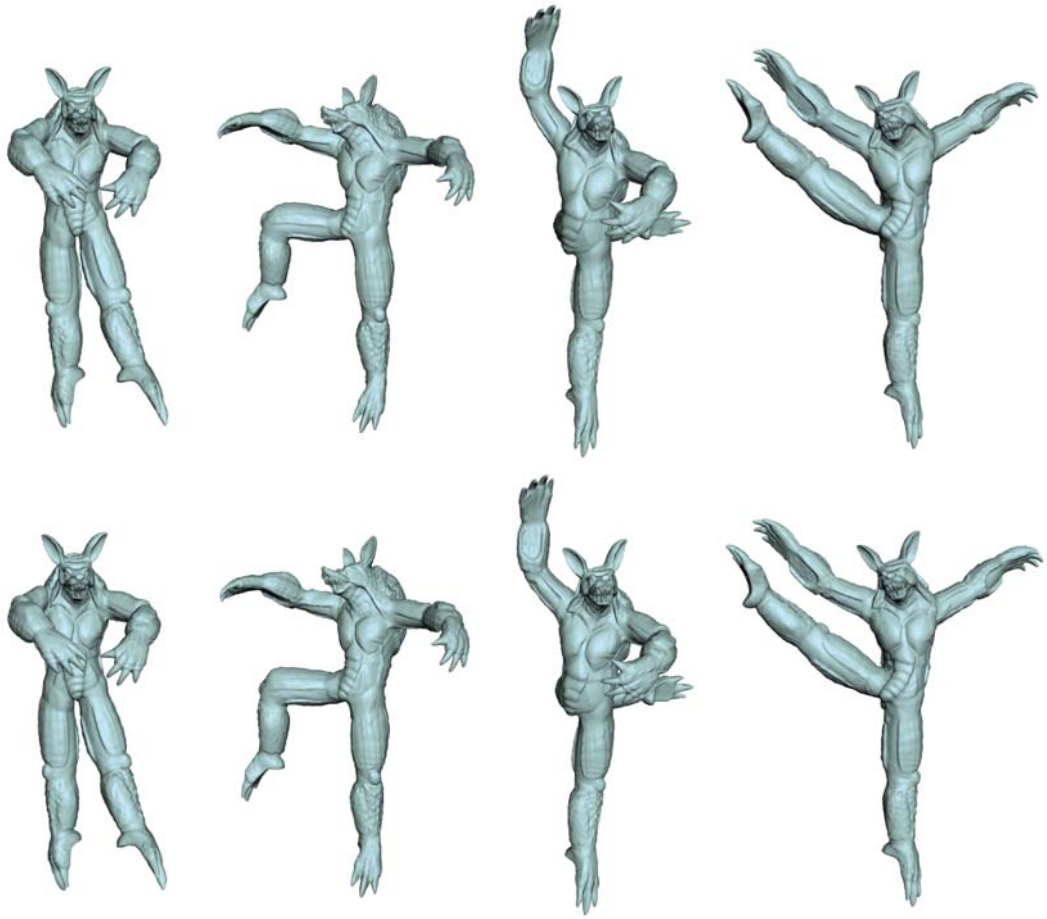


Figure 2.6: Comparisons with ground truth on articulated mesh deformation. The top row shows the ground truth which was not part of the training data, and the bottom row shows our synthetically generated results. (Kernel function : Gaussian)

Figure 2.7: Frames from predicted facial deformations generated using our method. The frames in the top row belong to a testing facial animation sequence. The bottom row shows novel expressions generated from interactive editing. (Kernel function : Polynomial)

Figure 2.8: Comparisons with ground truth on cloth deformation. The top row shows the ground truth which was not part of the training data, and the bottom row shows our synthetically generated results. (Kernel function : Gaussian)

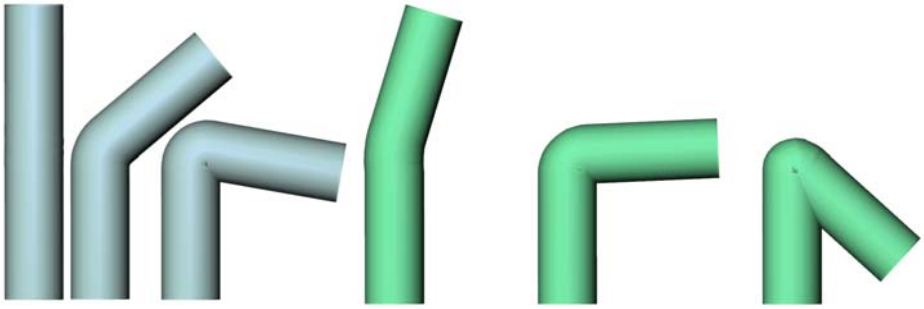


Figure 2.9: A bending style is trained from given examples to generate novel deformations with the same style. The blue models on the left are training examples. The green ones on the right are novel deformations. (Kernel function : Gaussian)

Figure 2.10: A simultaneous galloping and collapsing sequence generated from a deformation predictor trained using two separate galloping and collapsing sequences. (Kernel function : Gaussian)

At run-time, novel bone deformations can be predicted in real time in our system (Table 5.1) with novel control point movements. We demonstrate this by generating novel animated surface deformation with the motion trajectories of a sparse ($< 10$) set of control points. Here we obtain the control point movements directly from testing mesh sequences, though they can also be obtained from MOCAP data. Although surface deformation is underdetermined with such a sparse set of constraints in the original deformation space, our regression model is able to predict the deformations from the learned subspaces based on the training examples. The resulting deformations are shown in Figures 2.6, 2.7, and 2.8. The top rows of Figures 2.6 and 2.8 show ground truth that was not part of the training data. We have also designed an interface to let the user drag any of the control points either in a 3D space or on a 2D projection plane. The interactively defined control point positions can also be used for generating novel surface deformations.

We show the capability of our method in learning different deformation styles in Figure 2.9 and 2.15. Given sparse training examples of a bending style, our method can produce novel deformations with the same style. We can also combine examples of different deformation styles to produce a hybrid deformation predictor. In Figure 5.3, a simultaneous galloping and collapsing sequence is generated from our predictor using training examples from separate horse galloping and collapsing sequences.

We have compared our results with FaceIK [2] and PCA-based blendshape on facial animation. We used the

Ground Truth FaceIK PCA Blendshape Our Method

Figure 2.11: A comparison of predicted deformation among our method, FaceIK [2] and PCA-based blendshape. Our method generalizes well within the deformation subspace learned from training examples, and produces results closer to the ground truth than the other two methods.

| Examples | Vertices | Bones | #CCA | #Train | #Test | fps | Prep |
|---|---|---|---|---|---|---|---|
| Face | 23,728 | 260 | 8 | 38 | 384 | 502 | 20 min |
| Pants | 1,453 | 150 | 8 | 80 | 1691 | 466 | 15 min |
| Armadillo | 33,000 | 80 | 8 | 29 | 298 | 408 | 18 min |
| Horse | 8,431 | 150 | 8 | 27 | n/a | 506 | 13 min |
| Cylinder | 2,000 | 40 | 2 | 3 | 40 | 798 | 2 min |
| Bar | 8,000 | 90 | 2 | 3 | 40 | 632 | 6 min |

Table 2.1: Statistics and Timings. All performance measurements were taken from a 3.0GHz Pentium D processor with nVidia Geforce 8800GTS 640MB VRAM. '#CCA' means the number of CCA bases used for each bone, '#Train' means the number of training examples, '#Test' means the number of testing examples, and 'Prep' means the total time for all preprocessing steps. Our Poisson-based translation solver were not used for Face and Cylinder. Instead of the total number of vertices, the number of bones, training examples and the translation solver are more influential factors affecting the final frame rate.

same set of control points and their moving sequences for all three methods and recorded the prediction errors for each method. The RMS errors for FaceIK, PCA-based blendshape and our method are $0.0242$, $0.0143$ and $0.0108$, respectively, when the largest dimension of the bounding box of the face is scaled to have a unit length. Visual comparison results are also shown in Figure 2.11. While the numerical errors are not large for all methods, the visual result from our method is more natural and closer to the ground truth than others. FaceIK is not a real-time technique and sometimes produces results with obvious distortion around facial features such as the mouth. For PCA-based blendshape, we built a set of blendshape bases using PCA and solved for optimal blendshape weights from new control point positions using least-squares. Because the sparse set of control points are not sufficient to robustly solve for the blendshape weights, there are visible artifacts in the resulting deformations. We have also compared the performance of our method with Face poser [38]. In their method, a nonlinear optimization is performed on CPU at run-time and requires significantly more time than our method. It can only solve for less than six frames per second while our method is around two orders of magnitude faster.

We have compared the accuracy of our regression with SAD [3] using the same number of proxy bones. Since SAD cannot be directly used for generating novel deformations, we only perform the comparison on the training examples. Our results were obtained by predicting the deformations from control point configurations in the training examples, and SAD results were from direct skinning of training examples. The comparison is shown in Figure 2.12, where our method can faithfully reproduce the deformations from original training examples after regression while SAD fails to accurately fit the training examples with the same number of bones. Since SAD uniformly places proxy bones on the mesh, its skinning results are less optimal.

To demonstrate the scalability of our method, we have tested the performance of our system with an increasing number of proxy bones and training examples. As shown in Figure 2.13, our system can still achieve more than 100 fps even for the extreme case of 650 bones with the Poisson-based translation solver, which has a quadratic dependence on the number of bones. In Figure 2.14, we demonstrate the performance of our system by generating the deformations for a group of 15 pairs of pants on the fly at 35 fps.

## 2.8 Conclusions and Future Work

We have presented an intuitive and powerful user interface to simultaneously control the deformation of an entire deformable surface with a minimal number of control points. Our contributions include a novel deformation regression method based on kernel CCA, a Poisson-based translation solving technique, and an efficient GPU implementation. Our run-time algorithm can achieve a few hundred frames per second even for large datasets with hundreds of examples. Comparisons show our method can achieve better results than existing ones on challenging tasks such as handle-based facial animation.

Ground Truth SAD Our Method

Figure 2.12: A comparison of fitting quality between our method and SAD [3]. Our method generates more accurate and natural results.

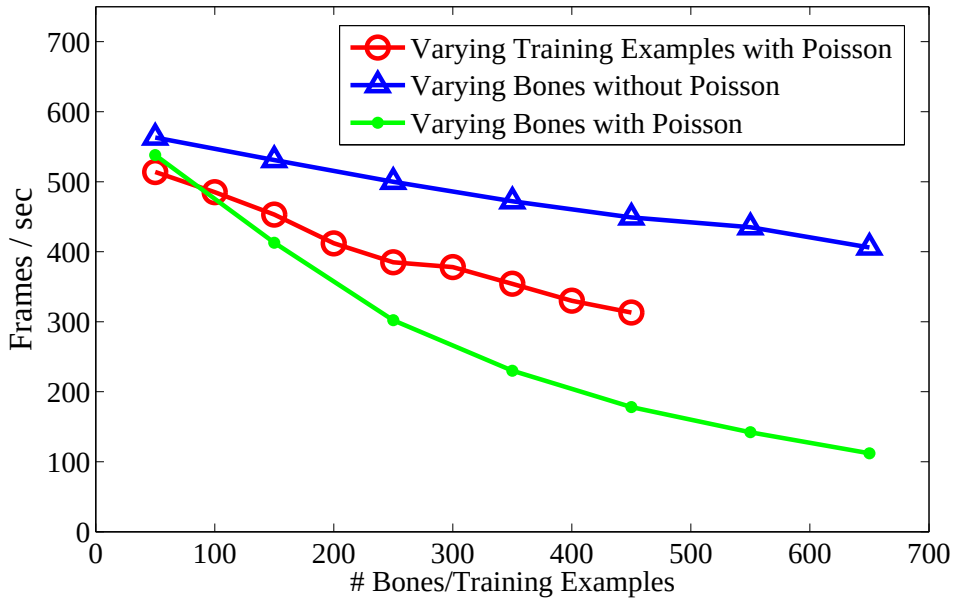



Figure 2.13: Performance plots of our method with or without the Poisson-based translation solver, using an increasing number of bones and training examples. For plots with a varying number of bones, we use 38 training examples for the regression model. For the plot with a varying number of training examples, we use 100 proxy bones.

There exist a few limitations in our method that need further research. First, the control points need to be specified before regression and fixed in subsequent animations. However, our current regression implementation only took less than 30 seconds in all of our experiments. With further optimization, it is possible to change control points and rebuild the regression model at run-time. Second, the predicted deformations are not always localized when moving control points. This is undesirable when precise control is necessary. However, in typical data-driven animations, the movements of control points are not independent but highly correlated. Therefore, such a limitation would not create serious problems in practice. Finally, since our method is data-driven, the quality of predicted deformations depend on training examples. As shown in Figure 2.15, when examples are completely missing in certain directions, our system can only generate simple shearing deformation. This can be alleviated by adding extra examples in those directions. Since our system is highly scalable with respect to the number of training examples, adding a few extra examples would not hurt performance in practice.

Figure 2.14: A group of 15 pairs of pants are animated simultaneously at 35 FPS. Deformations for individual pairs are generated independently in real time.

(A) (B) (C) (D)

Figure 2.15: Our data-driven method depends on training examples in (A) to produce an extreme new deformation in (B), which demonstrates a strong extrapolation capability. However, if examples are completely missing in the perpendicular direction, the predicted deformation becomes a simple shear as shown in (C), which can be improved by inserting an extra training example in that direction as shown in (D).

# Chapter 3

# A Deformation Transformer for Real-Time Cloth Animation

Achieving interactive performance in cloth animation has significant implications in computer games and other interactive graphics applications. Although much progress has been made, it is still much desired to have real-time high-quality results that well preserve dynamic folds and wrinkles. We introduce a hybrid method for real-time cloth animation. It relies on data-driven models to capture the relationship between cloth deformations at two resolutions. Such data-driven models are responsible for transforming low-quality simulated deformations at the low resolution into high-resolution cloth deformations with dynamically introduced fine details. Our data-driven transformation is trained using rotation invariant quantities extracted from the cloth models, and is independent of the simulation technique chosen for the lower resolution model. We have also developed a fast collision detection and handling scheme based on dynamically transformed bounding volumes. All the components in our algorithm can be efficiently implemented on programmable graphics hardware to achieve an overall real-time performance on high-resolution cloth models.

## 3.1 Introduction

Achieving interactive performance in cloth animation has significant implications in computer games, online fashion shows and other interactive graphics applications. Indeed, much progress has been made to achieve a reasonable quality in real time. Nevertheless, it is still much desired to have real-time high-quality results that better preserve dynamic folds and wrinkles.

It is challenging to achieve this goal for the following reasons. First, physically based cloth simulation involves two expensive steps, PDE integration and collision handling. Even the latest GPUs have a hard time performing both tasks in real time on high resolution cloth models. Second, data-driven model reduction can be potentially applied to improve cloth simulation performance. However, unlike elastic materials and fluids, cloth primarily has secondary motion driven by collisions against an animated body. It is not obvious how model reduction could accelerate collision detection and handling for high resolution cloth models.

In this paper, we explore a different approach that tries to decouple the spatial dimensions from the temporal

Figure 3.1: Cloth animations generated by our method.

dimension. The temporal dimension is in charge of dynamics and the two spatial dimensions provide a domain to define spatially varying details, such as folds, over the cloth surface. Spatial details across a cloth surface are highly correlated. For example, at least dozens of vertices over a high-resolution cloth need to move together in a coherent way to create a single fold. This means it is possible to generate all the spatial details from a lower dimensional space.

Based on this observation, we introduce a hybrid method for real-time cloth animation. The dynamics of a cloth model is generated by a low-resolution physically based simulation, and the high-resolution spatial details over the cloth surface are generated by a data-driven model from a low-dimensional space. We integrate the data-driven model with the dynamics model to produce complete high-quality cloth animations. This integration is achieved by training additional data-driven models that accurately capture the relationship between simulated coarse deformations and data-driven high-resolution spatial details. Such data-driven models transform simulated deformations at the lower resolution into high-resolution cloth deformations with dynamically introduced fine details. One of the major contributions of this paper is the identification and extraction of rotation invariant quantities that are well suited for training high-quality data-driven models.

We have developed a new animation pipeline to make the aforementioned hybrid cloth animation possible. Each time step starts with a one-step simulation of a low-resolution cloth, followed by deformation transformation and collision handling, and finally ends with high-resolution cloth surface reconstruction and rendering. The most important part is deformation transformation, which includes two nonlinear mappings that are respectively responsible for mid-scale and fine-scale deformations in the high-resolution model. To maintain a low collision detection and handling cost, we have developed a fast and effective scheme based on dynamically transformed bounding volumes. Every step of our run-time stage has been carefully designed to fully utilize the parallel processing power of modern GPUs. As a result, we achieve hundreds of frames per second when generating a high quality cloth animation from a synchronous

coarse simulation.

## 3.2 Related Work

**Data Driven Deformation.** Skeleton subspace deformations (SSD) attach a skin to bones and each vertex of the skin is deformed according to a weighted sum of nearby bone transformations. It has been generalized in a number of techniques [49, 20, 32, 6] to prevent potential artifacts. Meanwhile, fully automatic techniques have been developed to compute proxy bones and their influence weights from existing mesh animations [7].

Data-driven approaches have been able to produce realistic and detailed results. Allen *et al.* [50] interpolate scanned models with various poses to generate animations of skin deformation. Anguelov *et al.* [51] built a data-driven model of both the shape and deformation of full body surfaces to generate novel subjects animated with novel motion. Park and Hodgins [9] performed motion capture of skin deformation with a large number of markers. High quality skin deformations are reconstructed via a second-order skinning scheme followed by interpolation of the residual errors. Marker-based techniques for facial motion capture and synthesis have been reported in [52, 53]. Detailed facial geometry, such as wrinkles, can be successfully synthesized from local deformation. Feng *et al.* [54] introduced an example-based regression model for static surface deformation driven by sparse control points. This paper uses the same regression method as in [54] but in a different context for medium to fine-scale cloth motion transformation. More importantly, unlike [54], this paper relies on carefully designed rotation invariant quantities to achieve superior regression results rather than performing regression on global transformations. Moreover, the work in [Feng et al. 2008] only deals with mid-scale bone transformations. Thus it cannot produce detailed wrinkles without an excessive number of bones.

Producing high-quality dynamic data either via simulation or from a motion capture setup is a costly and time-consuming process. Much research has been performed to obtain dynamical models from existing data. This is typically achieved using dimension reduction and model fitting techniques [55, 56]. Barbič and James [55] derive a reduced dynamical model from an original one by applying dimension reduction to the state vectors while Park and Hodgins [56] adopt an empirical dynamical model for reduced dimensions and solve its parameters by fitting the model to motion capture data.

**Interactive Cloth Simulation.** Extensive research has been performed on cloth simulation [57, 58, 59, 60]. Much effort has also been devoted to making cloth simulation reach interactive performance. A common approach to faster cloth simulation replaces large in-plane forces with constraints [61, 4, 60] and iteratively solve the system of governing equations. Physics-based geoemtric subdivision methods are applied in [62] and [63] to produce additional folds in a coarse cloth mesh. These methods can generate more interesting high resolution cloth than applying traditional

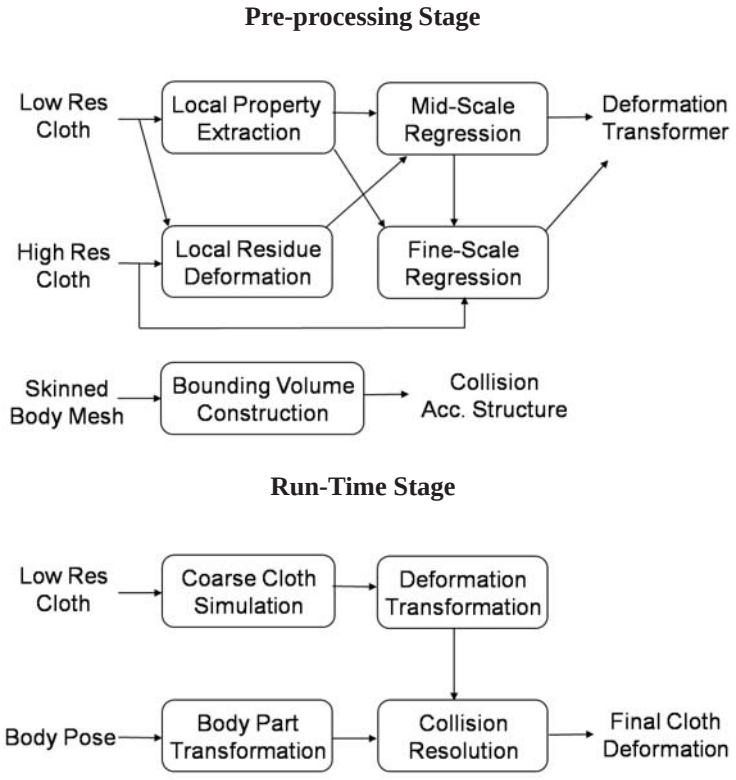



Figure 3.2: The workflow of our method.

geometric subdivision. However, it is hard to produce many wrinkles within one coarse triangle as the subdivision is driven by minimizing membrane energy or enforcing the edge length constraint.

Cordier *et al.* [64, 65] developed real-time techniques for animating clothes dressed on virtual characters. Their data-driven technique uses pre-simulated cloth animations as examples to correct the simulation run in real time on a much coarsened cloth mesh. Vertex positions or local cloth details are interpolated from the examples with the closest neighborhood configurations. In comparison, our technique does not need to look up closest neighborhoods. It is based on an advanced kernel regression method that takes a global configuration of the coarse cloth as the input. Results from our method have better visual quality. Stumpp *et al.* [66] proposed a shape-matching cloth model which can plausibly reproduce folds and wrinkles from a physically based simulation. A GPU-based implementation of a finite element method [67] has also been reported in [68]. An interactive performance up to 30fps on a cloth with 10K vertices has been reported using these techniques.

## 3.3 Overview

We introduce a data-driven framework for transforming low-resolution cloth simulations into higher resolution cloth animations in real time. Unlike deformation transfer [69] and motion retargetting [70] where the source animation has

a sufficient quality and the goal is to integrate the source animation with a new hosting model, the source animation in our framework may have a low quality and our goal is to transform it to a higher-quality one. This is made possible by including high-quality high-resolution cloth deformations as part of the training data. The workflow in our method, including both a preprocessing stage and a run-time stage, is summarized in Figure 3.2.

**Preprocessing Stage** The input to the preprocessing stage is a set of $n$ triplets, $\{(A^i, H^i, D^i)\}_{i=1}^n$, where $A^i$ is a skinned character model, $H^i$ is a deformed high-resolution cloth model for the character, and $D^i$ is a corresponding deformed low-resolution cloth model. In this paper, these three types of models are all represented as triangle meshes, and meshes with the same type but a differing superscript share the same topology. Our goal is to train nonlinear mappings that can approximately transform every $D^i$ to its corresponding $H^i$. Note that the underlying mechanisms for generating $D^i$ and $H^i$ may be different. For example, one could choose a fast low-quality simulator to generate $D^i$, and a slower, more expensive simulator to generate $H^i$.

We represent the deformation in the high-resolution model $H^i$ at two different scales, a mid scale and a fine scale. Mid-scale deformations treat the cloth surface as a set of local patches smoothly joined together. Internal variations of the patches, such as small folds and wrinkles, are modeled as fine-scale deformations. Mid-scale deformations are represented using a skinning model while fine scale deformations are represented as residual vectors at individual vertices. We perform regressions to obtain two separate mappings for these two types of deformations. These two mapppings are collectively called a *deformation transformer*. Our training procedure for the mappings is independent of the simulation technique chosen for the low-resolution model.

To accelerate collision handling at the run time, we also preprocess the cloth and character model pairs $(H^i, A^i)$ and build a two-level bounding volume structure for collision detection.

**Run-Time Stage** During the run-time stage, a coarse cloth model is simulated in real time. At every frame, local surface properties of the deformed coarse model are used as the input to the trained nonlinear mappings. Once mid-scale and fine-scale deformations have been generated from the mappings, we perform collision detection against the character model and further update the cloth deformation to resolve the collisions. The final deformation is used to reconstruct a high-resolution cloth surface for the current frame. Every step of our run-time process has been designed to fully utilize the parallel processing power of modern GPUs.

## 3.4 Cloth Deformation Transformation

Since a cloth is a highly deformable surface, the deformation transformer needs to be carefully designed to capture its intricate movements and folds. Moreover, a cloth can easily have multiple collisions with the animated character.

How to resolve these collisions efficiently also affects our choice of representing cloth deformations.

### 3.4.1 Mid-Scale Deformations

A physically based cloth simulation usually needs to model a cloth using thousands of vertices to generate interesting animations. However, once fine-scale details have been removed, the remaining cloth deformation becomes more spatially coherent within local regions. Therefore, we can choose to represent mid-scale cloth deformations using linear blend skinning with a set of proxy bones [7] each of which represents a local region. Note that these proxy bones are solely used for cloth deformation and are independent of the character's skeleton. Because cloth is mostly inextensible and its deformation is mostly local bending, the deformation of proxy bones is restricted to rigid transformations with no scaling.

We need to decide next which quantity regression should be performed on. Although it is tempting to perform regression directly on global bone transformations, there exist several drawbacks. First, a global transformation is not rotation invariant. The trained mapping needs to produce distinct results for rotated versions of the same deformation, which makes the training stage much harder. Second, the dynamic range of absolute rotation and translation of a proxy bone could be very large, which makes it hard for our mapping to accurately predict them. Finally, any inaccuracies in the mapped global transformations directly affect the resulting cloth geometry, leading to obvious visual artifacts. To resolve these issues, we observe that although the low-resolution cloth simulation does not produce high-quality deformations, it does provide us with a simple overall shape of the deformed cloth. Therefore, we choose to use the cloth geometry in the low-resolution simulation to obtain a first-order approximation of every bone transformation in the high-resolution cloth, and then perform nonlinear regression to estimate the residual transformation of this first-order approximation.

Given $n$ example deformations of the high-resolution cloth mesh, $\{H^i\}_{i=1}^n$, we partition the mesh model into a set of $m$ local patches $P_k$, $k = 1..m$, using a face clustering algorithm similar to hierarchical clustering in [6] (Figure

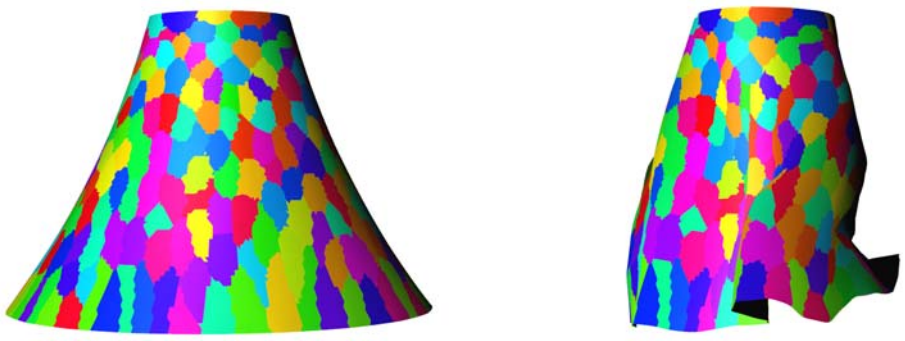
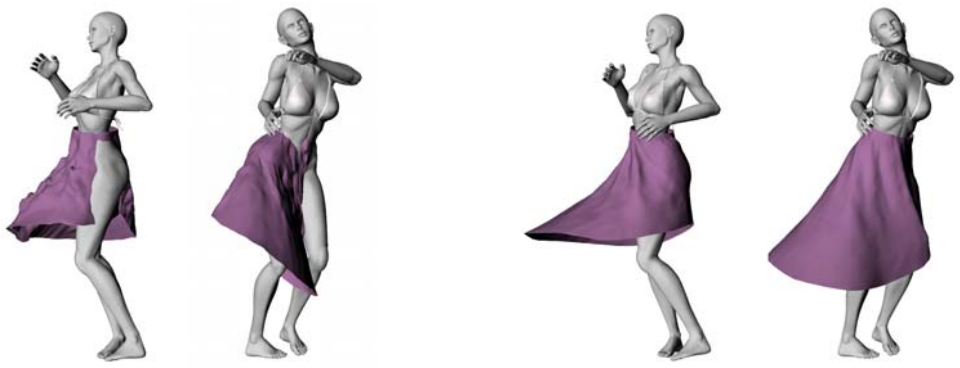


Figure 3.3: Proxy bones are used to model mid-scale cloth deformations. (Left) Bone clusters visualized on a rest-pose cloth model. (Right) Same bone clusters shown on a deformed cloth model. Note that cluster boundaries roughly follow cloth folds.

3.3). We set each patch $P_k$ as a proxy bone and extract its corresponding rotation, $R_k^i$, from the deformed model $H^i$. This rotation is defined with respect to the corresponding patch in the rest-pose high-resolution mesh. We also extract a rotation $G_t^i$ for each triangle $f_t$ in the deformed coarse mesh $D^i$. $G_t^i$ is defined with respect to the triangle corresponding to $f_t$ in the rest-pose coarse mesh.

The first-order approximation of $R_k^i$ is formulated as $\bar{T}_k^i = \sum_j w_t^k G_t^i$, where $w_t^k$'s are linear blending coefficients for patch $P_k$ and can be obtained by minimizing the following least-squares objective function, $\sum_{i=1}^{n} \| \sum_t w_t^k G_t^i - R_k^i \|^2$. Since $\bar{T}_k^i$ is not necessarily a rotation, we apply the polar decomposition to extract the rotation matrix $\bar{R}_k^i$ from $\bar{T}_k^i$. The residual transformation $\tilde{R}_k^i$ is the difference between $\bar{R}_k^i$ and $R_k^i$, defined as $\tilde{R}_k^i = (\bar{R}_k^i)^{-1} R_k^i$. It is advantageous to perform a regression on $\tilde{R}_k^i$ than $R_k^i$ because it is local, rotation invariant, and encodes intrinsic differences between coarse and high resolution cloth deformations. In practice, we use the exponential form of a rotation and perform regression on the logarithm of $\tilde{R}_k^i$, which becomes the axis-angle representation of a rotation. This is because the axis-angle representation is less ambiguous than a quaternion and also achieves more accurate regression results. We demonstrate the benefit of performing regression on residual transformations in Figure 3.4. A mapping directly trained on global transformations tends to produce highly distorted deformations because of the ambiguity and a large fitting error. Note that the regression performed in [Feng et al. 2008] also predicts global transformations. Therefore it would suffer from the same type of difficulties.

Since we have chosen to use rotation-invariant quantities from the high-resolution model, we need to use rotation-invariant properties of the coarse cloth as well during regression. There are a few possible choices available, including linear rotation invariant coordinates (LRI) [71] and local connection maps [72]. In our system, we choose dihedral angles between adjacent triangle pairs because it is both compact and rotation invariant. Using all dihedral angles from each example can capture static deformations, but may still miss the dynamic aspect of a cloth simulation. Therefore we use the complete set of dihedral angles, $\boldsymbol{\Theta}$, and their velocity, $\dot{\boldsymbol{\Theta}}$, during regression.



Global Rotation Invariant

Figure 3.4: A comparison of cloth deformations using global and rotation invariant bone transformations as regression targets. Global bone transformations are more difficult to generalize in the training stage and produce obvious artifacts in the resulting cloth deformations.

(A) (B) (C)

Figure 3.5: Steps of our mid-scale deformation transformation. Start with a coarse cloth deformation in (A), we extract a rotation from each coarse triangle. An intermediate first-order approximation of the rotations in the high-resolution cloth using these rotations of the coarse triangles is shown in (B). The residual transformations are then predicted from our regression model, and are combined with the rotations from the first-order approximation to produce mid-scale surface deformations in (C).

We seek a mapping $g_k(\boldsymbol{\Theta}, \dot{\boldsymbol{\Theta}}) \rightarrow \omega_k$ for each $P_k$, where $\omega_k$ is the three-dimensional axis-angle form of the residual rotation of $P_k$. The regression method we use to train the mapping is similar to the one in [54]. Since the mapping is likely to be nonlinear, to adequately account for the nonlinearity while avoiding overfitting, we perform nonlinear dimension reduction on $(\boldsymbol{\Theta}, \dot{\boldsymbol{\Theta}})$ by applying kernel canonical correlation analysis [31] between $(\boldsymbol{\Theta}, \dot{\boldsymbol{\Theta}})$ and $\omega_k$. Since our regression target is the residual rotation with an axis-angle representation, which is a 3D vector, the reduced dimension is set to three. This dimension reduction is followed by a linear regression between $\omega_k$ and the reduced coordinates for $(\boldsymbol{\Theta}, \dot{\boldsymbol{\Theta}})$. At run time, given a novel coarse cloth mesh, our mapping predicts the residual rotation $\tilde{R}_k$ and combines that with $\bar{R}_k$ to reconstruct $R_k$ (Figure 3.5). Since $\tilde{R}_k$ is a local rotation, it does not have a bone translation necessary for high-resolution cloth surface reconstruction. We compute bone translations by solving a coarse-grain Poisson equation in real time as in [54].

### 3.4.2 Fine-Scale Deformations

Detailed folds and wrinkles are difficult to model with proxy bone transformations since they are high-frequency features that may change rapidly even within a local patch. Increasing the number of proxy bones may alleviate this problem, but using too many proxy bones would affect the overall performance and defeat our original purpose for real-time applications. Thus we revise the eigenskin method [73] to represent these high-frequency details, and then train a second mapping for predicting such details. The original eigenskin technique was proposed to model details in articulated skin deformation that cannot be captured by proxy bones. It partitions a mesh into a set of influence regions associated with joints and performs dimension reduction on their residue offsets. However, in our problem,

these influence regions are not well-defined since a cloth model has its own dynamics and is not tightly coupled to joints in the articulated character.

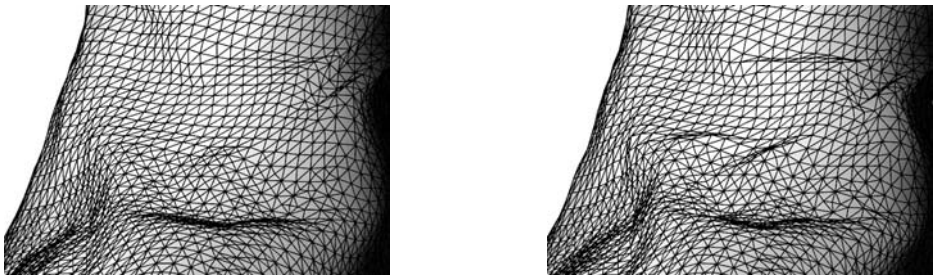
To adapt the eigenskin technique to solve our problem, we need to define suitable criteria for partitioning the cloth surface into local regions. We notice that although cloth wrinkles tend to be quite complicated, wrinkle offsets at individual vertices are usually correlated with each other. That is, offset patterns at one vertex may give rise to similar offset patterns at some other vertices. Therefore our goal is to identify regions that have coherent vertex offsets throughout the training examples. We propose a clustering scheme based on both hierarchical clustering and clustered principal component analysis (CPCA) [24] to find the coherent regions.

Given $n$ deformed high-resolution cloth meshes, $\{H^i\}_{i=1}^n$, as training examples, we represent the fine-scale deformation at vertex $v_j$ on mesh $H^i$ as a 3D displacement vector $u_j^i$ on the rest-pose mesh. Let the original and skinned position of $v_j$ on mesh $H^i$ be $v_j^i$ and $\bar{v}_j^i$, respectively. According to linear blend skinning, $\bar{v}_j^i = \sum_k w_k^j M_k^i v_j^0$ where $v_j^0$ is the rest-pose position of $v_j$, $M_k^i$ is the transformation of bone $P_k$ at mesh $H^i$, and $w_k^j$ represents the influence weight of bone $P_k$ on vertex $v_j$. Thus, $u_j^i = (\sum_k w_k^j M_k^i)^{-1} v_j^i - v_j^0$. The set of displacement vectors at the same vertex but across all $n$ training examples, $\{u_j^i\}_{i=1}^n$, are concatenated to form a $3n$-dimensional *per-vertex residual vector*. This residual vector represents per-vertex displacement vectors across all training examples instead of across different vertices.

At the beginning, our clustering algorithm sets up a face cluster $\Gamma_s$ for each triangle $f_s$. It performs hierarchical merging on these clusters afterwards. A PCA basis is computed for all the per-vertex residual vectors within a cluster. The error induced by merging cluster $\Gamma_{l_a}$ to $\Gamma_{l_b}$ is defined as

$$\sum_i \sum_{j \in \Gamma_{l_a}} \|u_j^i - \tilde{u}_j^i\|,$$

where $\tilde{u}_j^i$ is the approximation of per-vertex displacement $u_j^i$ by the PCA basis of cluster $\Gamma_{l_b}$. This error metric tends to merge clusters with similar per-vertex displacements across all training examples. We greedily merge the cluster pair with the lowest merging error. Whenever two clusters are merged, the PCA basis is recomputed from all the



Without Clustering With Clustering

Figure 3.6: A comparison of cloth deformations with and without face clustering in the modified eigenskin technique.

Mid-Scale Only Mid+Fine Scale

Figure 3.7: A comparison of deformation transformation results with and without fine-scale deformations. Mid-scale results produce a relatively smooth cloth surface while more interesting folds are generated by fine-scale deformation transformation.

per-vertex residual vectors in the merged cluster. This process is repeated until a desired number of clusters has been reached. Each of the final clusters represents a region with similar per-vertex residual vectors.

Now let us define a *per-example residual vector*, $\Upsilon_l^i$, for a final cluster $\Gamma_l$ on mesh $H^i$ by concatenating all 3D per-vertex displacement vectors in $\{u_j^i\}_{j\in\Gamma_l}$. The per-example residual vector is $3m$-dimensional if cluster $\Gamma_l$ has $m$ vertices. A PCA basis is also computed for all the per-example residual vectors corresponding to the same final cluster. For example, there is a PCA basis for the set of residual vectors, $\{\Upsilon_l^i\}_{i=1}^n$. This new PCA basis plays a similar role as the eigenskin basis in [73] and will be used for reconstructing wrinkle patterns within a cluster from given PCA coefficients.

With this new PCA basis for each final cluster $\Gamma_l$, we train a second mapping, $h_l(\boldsymbol{\Theta}, \dot{\boldsymbol{\Theta}}) \rightarrow c_l$, for fine-scale deformations in $\Gamma_l$. Here $c_l$ represents the PCA coefficients of a per-example residual vector from $\Gamma_l$. As in the previous section, the mapping is also trained using both kernel canonical correlation analysis and linear regression. The advantage of face clustering before regression is that we can obtain local mappings for the fine-scale details within each region, instead of a global mapping for details over the entire surface. As shown in Figure 3.6, a global mapping may miss certain fine-scale details, while a set of local mappings produce deeper folds and wrinkles. Note that the total size of PCA basis vectors are identical between a global PCA and a clustered PCA.

At run time, the residual vector within every cluster is reconstructed from predicted PCA coefficients to produce detailed folds over the high-resolution cloth surface. We show the benefits of adding fine-scale deformations in Figure 3.7. The deformation results with our eigenskin-based mapping have more detailed folds that cannot be captured by bone transformations alone.

## 3.5 Run-Time Implementation

We carefully choose operations involved in our run-time algorithms to make sure all of them can be implemented efficiently on programmable graphics hardware to achieve real-time performance on high-resolution cloth models. The overall run-time stage has four steps: low-resolution cloth simulation, deformation transformation, collision detection and resolution, and high-resolution cloth surface reconstruction. In our implementation, we use CUDA for the first three steps, and GLSL for final surface reconstruction and rendering.

**Coarse Cloth Simulation** We implemented the mass-spring model using Verlet integration [74] in the low-quality coarse cloth simulation. Since an explicit integration scheme is used, the spring lengths need to be constrained to ensure stable integration. A few iterations of Jacobi relaxation is applied in parallel to every cloth spring to enforce such length constraints [4]. Cloth-body collision detection is performed by testing every cloth triangle against the body mesh via two levels of bounding volumes as discussed in the previous section.

**Deformation Transformation** Once we have obtained the coarse cloth deformation, dihedral angles are extracted from the current vertex positions. $f_k(\mathbf{\Theta}, \dot{\mathbf{\Theta}})$ and $g_l(\theta, \dot{\mathbf{\Theta}})$ are used to predict residual rotations for mid-scale deformations and PCA coefficients for fine-scale offsets, respectively, in the high-resolution cloth model. This process involves matrix-vector multiplications and kernel function evaluations, which can be implemented efficiently on the GPU. Once we have obtained bone rotations, we need to solve a linear system (a Poisson equation) to obtain bone translations. The solution matrix for the linear system can be precomputed, and thus only a matrix-vector multiplication is needed at the run-time stage.

**Collision Detection and Resolution** Before testing collisions for the high-resolution cloth model, we transform in parallel all first-level bounding volumes according to corresponding bone transformations. We test collisions between all lozenges and their nearby body part capsules $C_{P_k}$ in parallel. Penetration depths are also computed in parallel for every lozenge to update the corresponding bone translation.

**High-Resolution Cloth Reconstruction** The final cloth surface is reconstructed via eigenskin-based offsetting and matrix-palette skinning. Using the predicted PCA coefficients $c_l$ for per-example residual vectors, the displacement $u_j$ for vertex $v_j$ is obtained by $u_j = V_{jl}c_l$, where $V_{jl}$ has the components associated with $u_j$ in the PCA basis $V_l$. We upload $V_l$ as a vertex texture in advance, and reconstruct $u_j$ in the vertex shader before skinning. The final vertex position $v_j^f$ on the cloth surface is

$$v_j^f = \sum_k w_k^j M_k(\bar{v}_j^0 + u_j)$$

| Cloth | Motion | # Tri | # Low Tri | Bones | #Cluster | #RotCCA | #EigPCA | #Train | #Test | Error | fps | Prep |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Skirt 1 | Dance 1 | 38,502 | 200 | 330 | 100 | 3 | 20 | 120 | 601 | 2.5% | 268 | 13 min |
| Skirt 1 | Jump | 38,502 | 200 | 330 | 100 | 3 | 20 | 60 | 314 | 2.5% | 271 | 11 min |
| Skirt 2 | Dance 2 | 25,693 | 180 | 320 | 80 | 3 | 20 | 48 | 224 | 2.4% | 280 | 8 min |
| Dress | Dance 3 | 27,402 | 220 | 350 | 120 | 3 | 20 | 124 | 600 | 1.8% | 261 | 15 min |
| Dress | Walk | 27,402 | 220 | 350 | 80 | 3 | 20 | 47 | 239 | 1.6% | 266 | 10 min |
| TShirt | Dance 1 | 25,726 | 322 | 350 | 120 | 3 | 20 | 120 | 601 | 2.8% | 251 | 15 min |

Table 3.1: Statistics and Timing. All performance measurements were taken from a 3.0GHz Core 2 Duo processor with a nVidia Geforce GTX275 Graphics Processor. '#Tri' and '#Low Tri' refer to the number of triangles in the high and low resolution cloth models, respectively. '#Cluster' means the number of face clusters used for Eigenskin, '#RotCCA' means the number of CCA basis vectors used for mid-scale bone residual transformation regression, and '#EigPCA' means the number of PCA basis vectors used for representing fine-scale deformations within each cluster. '#Train' means the number of training examples, '#Test' means the number of testing examples, and 'Prep' means the total amount of time for all preprocessing steps. Error is computed using the average per-vertex error divided by the radius of the bounding sphere of the cloth.

Figure 3.8: Comparisons between ground truth and final deformation results from our method. The left images show the ground truth, and the right images shows our final deformation results.

where $w_k^j$ is a bone influence weight at $v_j$, and $\bar{v}_j^0$ is the rest-pose position of $v_j$. Here both $w_k^j$ and $\bar{v}_j^0$ can be preloaded to the GPU memory via vertex buffer objects (VBO). Since cloth wrinkles are highly dynamic, we recompute vertex normals on the fly every frame to guarantee accurate normals for wrinkle rendering.

## 3.6 Experimental Results

We have successfully tested our deformation transformer on different types of clothing and body movements (Figure 4.1). For each clothing, we directly constructed the low-resolution model using Maya, and then produced the high-resolution cloth via mesh subdivision. It should be feasible to generate the low-resolution cloth by simplifying a given high-resolution mesh as well. We then use Poser, a commercial software package for 3D character animation, to generate skinned character animations and their associated high resolution cloth simulations as training data. The cloth simulator in Poser is capable of simulating different types of cloth materials with different settings of simulation parameters such as cloth density, folding resistance, damping, etc. We utilize these features to produce high quality cloth animations as our training examples. For each type of high-resolution clothing, we also generate a corresponding

Most Responsive Less Responsive Most Damped

Figure 3.9: Deformation transformation results with distinct styles. Three high-resolution cloth simulations with different material properties are used as training examples. All of them are trained with the same coarse cloth simulation. Our method can adapt to different cloth material properties in the training examples and produce distinct cloth animation styles.

low-resolution cloth animation, which only needs to have roughly the same overall shape. We follow the method in [4, 74] to produce low-resolution cloth animations. The same method is applied when performing low-resolution cloth simulation on the GPU to ensure a low simulation overhead in our run-time cloth animation pipeline. The number of triangles in a coarse cloth model is typically between 200 to 300 triangles. We found this resolution sufficient to capture a rough shape of the high-resolution cloth without much performance penalty during runtime simulation.

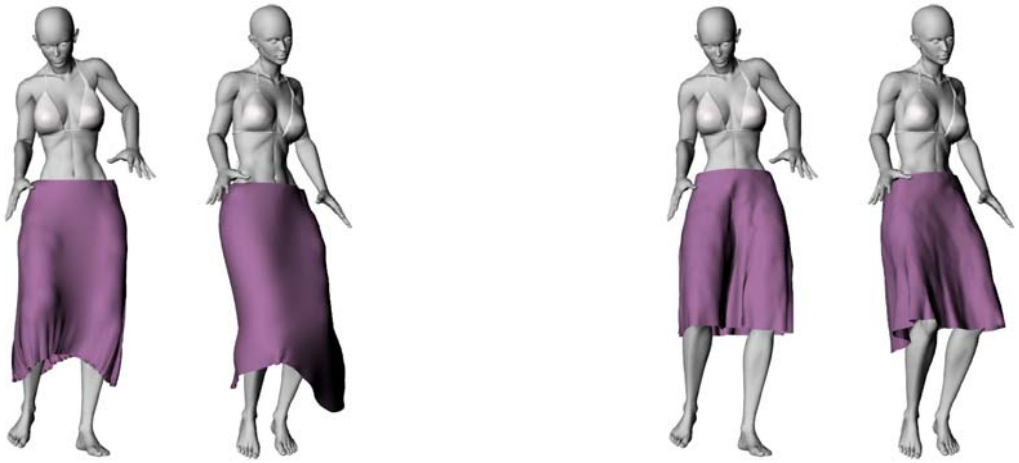
For each animation sequence, we select the training examples automatically by performing K-Mean clustering on the dihedral angles $\Theta$ of low-resolution cloth. From each cluster, the pose closest to the cluster center is then selected as the training example. Comparing to random selection, selecting the examples by clustering helps finding the more representative cloth deformations in the animation.

Training data from a high-resolution cloth simulation generated by Poser needs to be preprocessed and converted to our mid-scale and fine-scale deformation representations. In our experiments, we found that 300 to 400 proxy bones were suitable for capturing important mid-scale deformations in the training examples; and fine-scale details could be captured by 80 to 120 CPCA clusters with 20 basis vectors per cluster in our modified eigenskin technique (Table 5.1). In all our experiments, we applied the 3rd degree inhomogeneous polynomial kernel for CCA-based regression.

Our runtime stage includes both coarse cloth simulation and deformation transformation. Our runtime algorithm can achieve more than 250 frames per second on an nVidia Geforce GTX275 GPU when generating high quality cloth animations with a resolution between 25,000 and 38,000 triangles, as shown in Table 5.1. As can be seen, the number of triangles in our high-resolution cloth models is more than 100 times the number of triangles in our coarse cloth models.

Our trained transformer can produce high quality cloth deformations from a coarse cloth simulation. A comparison between novel cloth deformations from our transformer and the ground truth are shown in Figure 3.8. Our transformed cloth deformations are not only visually appealing, but also numerically accurate, as shown in the ”Error” column in Table 5.1. Our method is also capable of adapting to training examples with different cloth material properties. As shown in Figure 3.9, three different sets of training examples are simulated with different parameter settings to show varying responsiveness and wrinkles. The transformers trained using these different sets of examples are capable of capturing intrinsic material properties from the training data and produce cloth deformations with the corresponding material properties.

We have also compared our results with a direct simulation of high resolution cloth on the GPU using our CUDA implementation of the method in [4], which is similar to the method implemented in NVIDIA PhysX. The performance of direct simulation is about 80 FPS for the cloth model with 38K triangles. We found the major performance bottleneck was its iterative scheme for limiting spring length. While it takes only a few iterations for a coarse cloth, it may take many more iterations to generate acceptable results for a higher resolution cloth. As shown in 3.10, the simulated cloth still appears stretchy after 40 iterations, but its performance has already dropped to 60 FPS. Increasing the number of iterations could improve the simulation results, but would also further affect the overall performance. On the other hand, our hybrid method produces more inextensible cloth deformations with less computational cost.



Explicit Strain Limiting Our Method

Figure 3.10: A comparison between cloth animations generated from our method and the method in [4], where explicit integration with iterative strain limiting is used to produce nonstretchy cloth. Compared with explicit integration with strain limiting, our method can produce more inextensible cloth deformations similar to the training examples.

## 3.7 Conclusions and Discussion

We have introduced a hybrid method for real-time cloth animation. It relies on data-driven models to capture the relationship between cloth deformations at two resolutions. Such data-driven models are responsible for transforming low-quality simulated deformations at the low resolution into high-resolution cloth deformations with dynamically introduced fine details. Our data-driven transformation is trained using rotation invariant quantities extracted from the cloth models, and is independent of the simulation technique chosen for the lower resolution model. Our method achieves hundreds of frames per second when generating a high quality cloth animation from a synchronous coarse simulation.

**Limitations** Like other data-driven methods, our proposed method requires a preprocessing stage where a data-driven model is trained. In addition, once deformations from a coarse simulation fall outside the deformation subspace defined by the training examples, numerical accuracy cannot be guaranteed. Nevertheless, we found experimentally that reasonable visual results could still be generated in such circumstances. Another limitation of our current method is that cloth self-intersections are not handled. We would like to add this capability in future using a spatial partition scheme, such as a uniform voxel grid. When two nonadjacent proxy bones on the cloth intersect with the same voxel, a potential self-intersection occurs and can be resolved by pulling the bones along their negative normal directions. Such a self-intersection resolution scheme is not expected to consume many GPU cycles.

# Chapter 4

# Triangular Geometry Images for Level-of-Detail Control of Skinned Mesh

Geometry images resample meshes to represent them as texture for efficient GPU processing by forcing a regular parameterization that often incurs a large amount of distortion. Previous approaches broke the geometry image into multiple rectangular or irregular charts to reduce distortion, but complicated the automatic level of detail one gets from MIP-maps of the geometry image.

We introduce triangular-chart geometry images and show this new approach better supports the GPU-side representation and display of skinned dynamic meshes, with support for feature preservation, bounding volumes and view-dependent level of detail. Triangular charts pack efficiently, simplify the elimination of T-junctions, arise naturally from an edge-collapse simplification base mesh, and layout more flexibly to allow their edges to follow curvilinear mesh features. To support the construction and application of triangular-chart geometry images, this paper introduces a new spectral clustering method for feature detection, and new methods for incorporating skinning weights and skinned bounding boxes into the representation. This results in a ten-fold improvement in fidelity when compared to quad-chart geometry images.

## 4.1 Introduction

Traditional view-dependent LOD representations based on mesh simplification [12, 13, 14] rely on random-access mesh traversal with poor cache coherence. Geometry images [15] support efficient LOD display [16, 17, 18] by storing the mesh as a MIP-mapped texture image with better GPU cache coherence, but flattening a mesh into a single geometry image can create severe parametric distortion. Multi-chart geometry images [75] improve this distortion with feature sensitive clustering, but its irregular chart boundaries complicated coarser levels of LOD downsampling. Rectangle-chart geometry images [1, 76, 77] pack and down-sample better, but their rectangular shape constraint creates charts that cross prominent mesh feature lines and obfuscate these features at coarser levels of detail.

Triangle-chart geometry images offer a single solution that combines the cache coherence of geometry images, the lower distortion of multiple charts, the straightforward downsampling of a simple chart shape, and a feature preserving layout. The triangle's barycentric coordinates uniformly sample each chart, and provide multiple levels

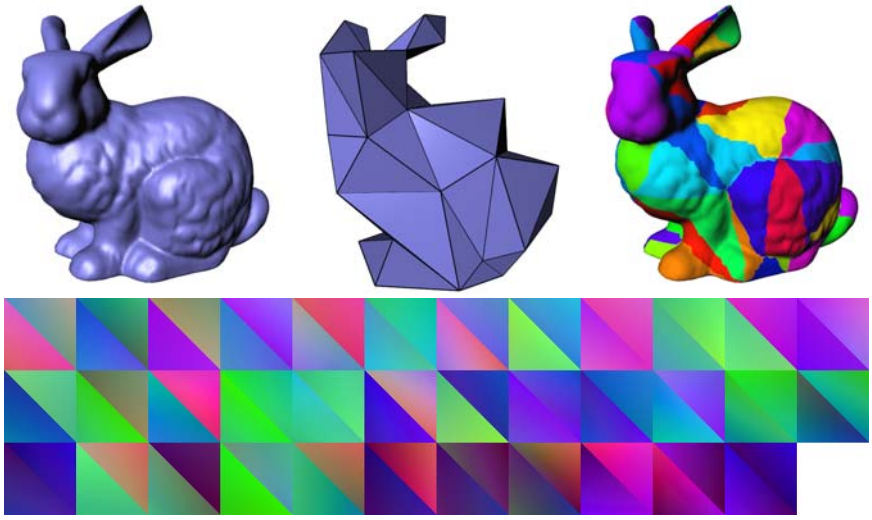


Figure 4.1: Our method builds triangular geometry images for feature-preserving LOD representation of both static and skinned meshes. It obtains a base complex and triangular patches from an original mesh using mesh simplification. The packed triangular geometry images are shown in the bottom.

of downsampling when the initial number of samples along the triangle edges is a power of two. Triangle charts benefit from a more flexible simplicial layout that, when compared to rectangle charts, avoids T-junctions, better supports feature-sensitive boundary alignment and packs (in pairs) just as easily and efficiently into an atlas. Our results show that triangle chart geometry images represent meshes with about 10% of the error incurred by rectangular chart approaches.

We support the feature preserving capabilities of triangle-chart geometry images with new algorithms for detecting feature curves in meshes and for organizing triangle charts to lie between these curves to preserve these features across levels of detail. We first detect features globally using a novel spectral face clustering on a mesh curvature estimate that identifies feature curves along cluster boundaries near local curvature maxima. A modified edge-collapse simplification forms vertex clusters within these feature curve boundaries. Each vertex cluster is parameterized onto a triangular domain and uniformly resampled to form a triangular geometry image.

We also augment geometry images with new support for the dynamic meshes found in modern videogames animated by linear-blend skinning deformations. In addition to the (rest-pose) vertex positions, we store vertex skinning weights in the geometry image, and downsample both for dynamic mesh LOD. To maintain spatiotemporal features at coarser levels of detail, we incorporate sequences of deformation transforms into the feature preservation metric to yield a space-time metric that clusters vertices with similar frame-to-frame transformations. This approach yields better shaped clusters and more accurate skinning than do hierarchical clustering [6] or SMA (Skinning Mesh Animations) [7]. We also surround each skinned triangle chart with an oriented bounding box whose corners are themselves skinned to deform with its contents. The screen size of a projected bounding box selects the optimal LOD resolution

for each triangular chart.

In summary, we introduce 1) a new type of triangular geometry images with feature preserving capabilities, 2) a spectral clustering method for effective curvilinear feature detection and deformation discontinuity detection, 3) GPU-based multi-resolution geometry image rendering for static and skinned meshes. The result is a view-dependent LOD representation for both static and skinned meshes stored and rendered entirely on the GPU to maximize throughput. It enables the convenience of geometry images to serve as a high-performance choice for representing characters, objects and scenes in games and virtual environments.

## 4.2 Related Work

The original geometry-image approach [15] cut a mesh into a single contractible component, mapped it onto a rectangular parametric domain and imposed a regular mesh sampling, often with high distortion. Multi-chart geometry images with irregular [75] or later rectangular [1, 76] boundaries reduced distortion by decomposing the input mesh into multiple pieces each individually parameterized and resampled. (Alternatively, [77] improved fidelity by subdividing a geometry image after parameterization into square charts sampled at different rates.) The construction of charts can utilize any of a number of mesh decomposition or clustering algorithms [78, 79, 80, 81, 82, 83], but they produce either rectangular or irregularly shaped charts. Mutiresolution analysis on triangle mesh [84] can be applied here to generate triangular charts. However, it is not straightforward to integrate our feature preserving scheme into its chart generation method. MAPS [8] constructed and parameterized a base domain of triangular charts, which serves as a working parameterization for our construction of feature-sensitive well-shaped triangle charts for geometry images, as described in Section 4.5.

Mesh Colors [85] uses barycentric coordinates to regularly sample textures over every triangle in an input mesh, packing the texture signal into a 1-D texture stream. Their approach focuses on atlas-free storage of a color texture whereas we focus on a geometric representation for view-dependent LOD whose atlas derives from a simplified base domain.

Several static mesh segmentation algorithms align cluster boundaries with feature lines [79, 86, 87]. They use curvature estimates to detect fragmented features, and connect these fragments into feature lines, but this gap filling can be ambiguous and sensitive to noise. Spectral clustering overcomes this sensitivity with a global approach to mesh segmentation [88, 80]. We improve this approach with new metrics that better detect crest lines and sharp points by processing the dual mesh in Section 4.4.2. We remove partial cluster boundaries that do not lie closely to local curvature maxima whereas other similar approaches handle these regions with flexible “fuzzy” boundaries [89].

Clustering and simplification for deforming meshes has been less explored. The quadric error metric [19] can be

extended [20, 21] for simplification of a mesh with multiple deformed poses into a pose-independent simplified mesh. Hierarchical face clustering has been performed in [22] to achieve pose-dependent simplification with higher visual quality for precomputed mesh deformation sequences. Hierarchical clustering [6] and a mean shift algorithm [7] can both yield pose-independent face clusters for deforming meshes, but Section 4.4.3 shows that our spectral clustering better localizes deformation discontinuities and preserves spatial coherence. To our knowledge, there is no previous work dealing with level-of-detail representation and control for skinned meshes and this paper is the first piece of work that applies multiresolution geometry images to skinned meshes.

Rendering throughput in modern GPUs implies that it is more important to optimally feed the graphics pipeline than fine-grain LOD adaptivity. Recent simplification-based LOD models focus on coarse-grained mesh resolution changes to minimize CPU usage and maximize GPU triangle throughput [90, 91, 92]. Our triangle-chart geometry image representation aligns well with this motivation and can maximize the GPU throughput of LOD models. First, it implicitly encodes vertex connectivity, avoiding the need to find and convert tri-strips. Second, the regular sampling of triangle-chart geometry images simplifies the stitching of neighboring patches differing by multiple levels of detail, whereas others allow only a single level difference, e.g. [91]. Third, it combines geometry and texture into a single multiresolution representation, avoiding the wasted storage of texture coordinates for every chart in video memory.

## 4.3 Overview

Our method produces triangular patches for an input mesh with patch boundaries following important geometric features on the mesh. These patches are converted to geometry images and used for dynamic level-of-detail rendering. The overall pipelines of our method for the preprocessing and runtime stages are summarized in Figure 4.2.

**Preprocessing** The first part of preprocessing extracts coherent curvilinear features on the mesh. For static meshes, curvilinear features in high curvature areas are detected. For skinned meshes, deformation discontinuities are also detected as additional features. These curvilinear features serve as constraints in a later stage where triangular patches are formed. Feature detection starts with spectral clustering [88, 80] using a similarity matrix based on curvature or deformation gradients. The boundary of these clusters serve as feature candidates. A subset of these feature candidates are retained as detected features. Since these initial features tend to be jagged, we further apply the graph-cut algorithm to refine the retained features.

The second part of preprocessing generates triangular patches for the input mesh. We perform extreme simplification to the input mesh to obtain a base complex with a very small number of triangles each of which serves as the parametric domain of a triangular region over the input mesh. Thus the complete base complex serves as a global parametric domain for the entire input mesh. Curvilinear features detected from the previous stage are used as con-

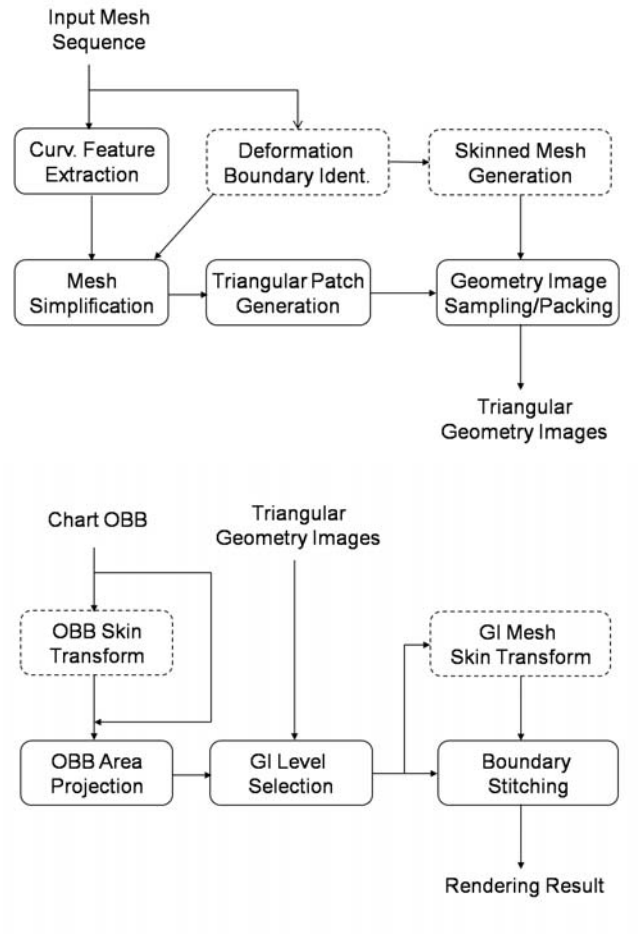



Figure 4.2: The pipelines of our method in the preprocessing and runtime stages.

straints in the simplification process. During mesh simplification, we apply MAPS [8] to figure out which triangle in the simplified mesh should be used as the parametric domain of a vertex in the input mesh as well as the barycentric coordinates of this vertex. Triangular patches on the input mesh can be obtained by mapping the edges of the base complex onto the input mesh. The obtained triangular patches on the input mesh are then sampled and packed into triangular geometry images. A mip-map hierarchy for each geometry image is also built to represent different levels of details.

**Run-Time LOD Rendering** At run time, a suitable level of detail is determined on the fly for each patch based on its screen projected area. An oriented bounding box (OBB) associated with each patch is used to approximate the projected area. To adapt our LOD calculation to skinned meshes, we need to dynamically estimate the screen projected area of the bounding box of each skinned patch. This is achieved by applying skinning to the bounding boxes as well and computing a set of bone influence weights for every corner of the OBBs. Once the suitable detail

levels are obtained, we render each patch at its corresponding level of detail on the GPU using its geometry image. To avoid cracks along patch boundaries, we apply automatic stitching on the GPU along boundaries of adjacent patches with different geometry image resolutions.

**Notation** We define a 3D mesh $M = (V, E, F)$ as a set of 3D positions $V = \{v_i = (x_i, y_i, z_i)\}$, mesh edges $E$ (a set of vertex pairs), and triangle faces $F$ (a set of vertex triples). To perform spectral face clustering and face-oriented graph-cut, we also define the dual graph $G = (F, D)$ of a mesh $M$, where $F$ is the set nodes in the dual, one for each face in $M$, and $D$ is the set of graph edges, each denoted by a pair of face nodes $(f_i, f_j)$ from $F$. Note that an edge in the dual graph can connect pairs of faces that are not adjacent to each other in the mesh. A path $P = (v_i, v_j)$ is defined as a set of connected mesh edges $E_p \subset E$ that connects $v_i$ and $v_j$. Given a skinned mesh with $n_b$ bones and $n_a$ frames of animation, we denote the set of bone transformations $\mathbf{T}_b^k$ and a set of skinning weights $w_i^b$, where $1 \leq b \leq n_b, 1 \leq k \leq n_a, 1 \leq i \leq |V|$ and $\sum_{b=1}^{n_b} w_i^b = 1$ for each vertex. The skinned position of $v_i$ at frame $k$ becomes $v_i^k = \sum_b (w_i^b \mathbf{T}_b^k) v_i$.

## 4.4 Curvilinear Feature Detection

We would like to preserve perceptually salient corner and curvilinear features during multiresolution resampling of the original mesh by aligning chart boundaries with such features. We explicitly detect a sparse set of salient corner and curvilinear features before chart generation. A common approach to feature line detection would apply local criteria first to detect fragmented feature points, followed by a gap-filling step to connect them. Since local feature detection is noisy, feature connection can be ambiguous and error-prone. In addition, in this process, salient feature lines do not necessarily have a higher priority to be discovered. In this section, we take a top-down approach instead by performing global spectral clustering with a new metric which takes into account local feature measurements. Salient feature lines are discovered as partial boundaries of the resulting triangle clusters. Our approach works very well with both static and deforming meshes. The reason that we use spectral clustering only for feature detection but not for chart generation is that it gives rise to irregularly shaped clusters not suited for triangular geometry images.

### 4.4.1 Spectral Clustering

Let $\mathcal{G} = (\mathcal{U}, \mathcal{E})$ be a weighted graph, where the set of nodes, $\mathcal{U} = \{u_1, u_2, ..., u_n\}$. An edge, $(u_i, u_j) \in \mathcal{E}$, has a weight $w(u_i, u_j)$ defined by the similarity between the location and attributes of the two nodes defining the edge. The idea is to partition the nodes into two subsets, $\mathcal{A}$ and $\mathcal{B}$, such that the following disassociation measure, the normalized

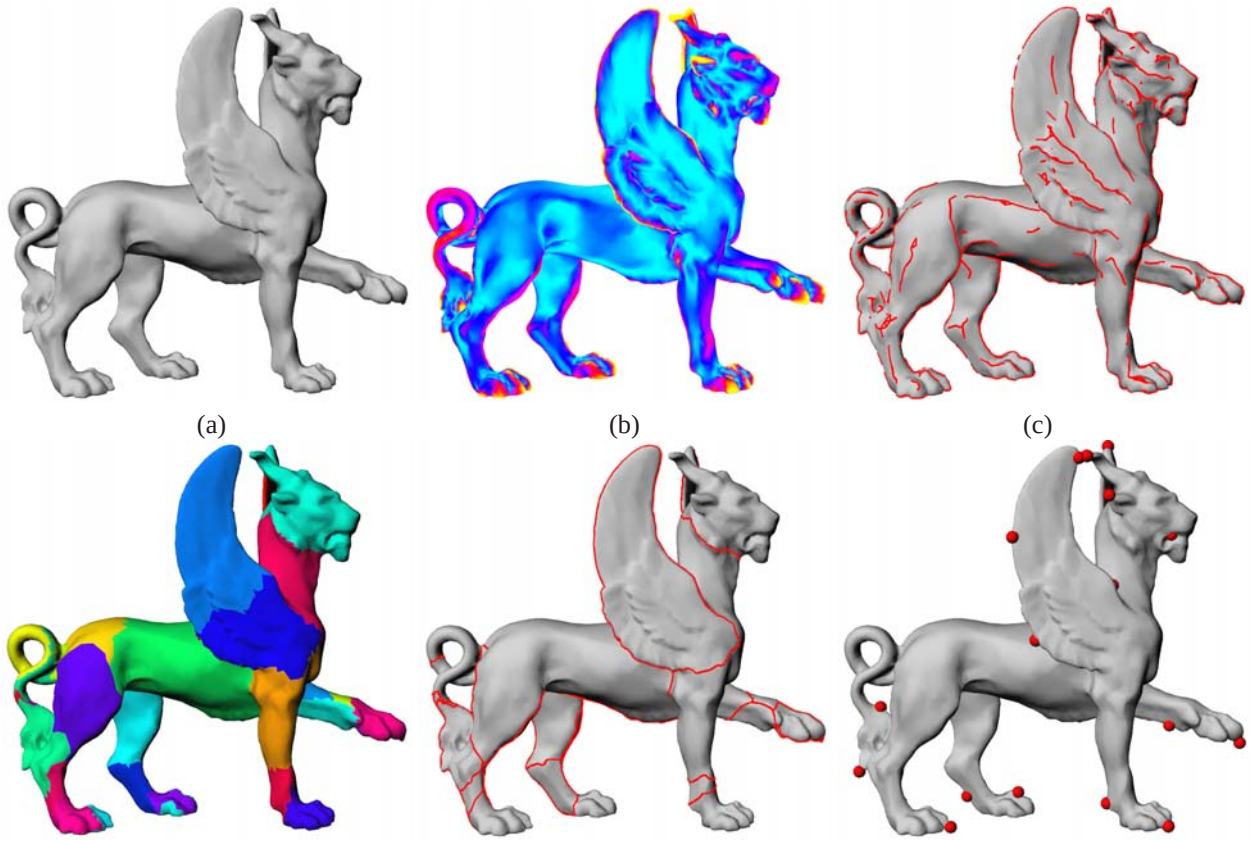


Figure 4.3: The overall process of our feature extraction method. Given a mesh in (a), we first robustly estimate its per-vertex curvatures, as shown in (b). The initial crest lines in (c) are noisy and disconnected. Spectral clustering is applied on the mesh based on curvature similarity to extract a set of clusters shown in (d). We only keep cluster boundaries with high curvature and refine them using a graph-cut algorithm to obtain the final feature lines in (e). A sparse set of corner points are also detected in high curvature regions as shown in (f).

cut, is minimized,

$$Ncut(\mathcal{A},\mathcal{B}) = \frac{cut(\mathcal{A},\mathcal{B})}{cut(\mathcal{A},\mathcal{U})} + \frac{cut(\mathcal{A},\mathcal{B})}{cut(\mathcal{B},\mathcal{U})} \tag{4.1}$$

where $cut(\mathcal{X},\mathcal{Y}) = \sum_{s\in\mathcal{X},t\in\mathcal{Y}} w(s,t)$ is the total connection from nodes in $\mathcal{X}$ to nodes in $\mathcal{Y}$.

To compute the optimal partition based on the above measure is NP-hard. However, it has been shown [93] that an approximate solution may be obtained by thresholding the eigenvector corresponding to the second smallest eigenvalue of the normalized Laplacian $\mathcal{L}$, which is defined as

$$\mathcal{L} = \mathbf{D}^{-1/2}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-1/2} = I - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}, \tag{4.2}$$

where $\mathbf{D}$ is a diagonal matrix with $\mathbf{D}(i,i) = \sum_j w(u_i,u_j)$, and $W$ is the weight matrix with $\mathbf{W}(i,j) = w(u_i,u_j)$.

Extensions to multiple groups may be realized through the use of multiple eigenvectors [88]. Let us first take the $N_e$ largest eigenvalues, $\lambda_1,\ldots,\lambda_{N_e}$, of $\mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ and their associated eigenvectors, $\mathbf{e}_1,\ldots,\mathbf{e}_{N_e}$. Let $\mathbf{M_e}$ be

Before After

Figure 4.4: The original feature lines from spectral clustering appear jaggy and may not align well with real features on the mesh. After applying the graph-cut algorithm, the refined feature lines become smoother and better localized.

a matrix with its $i$-th column set to $\mathbf{e}_i/\sqrt{\lambda_i}$. The rows of $\mathbf{M_e}$ define an embedding of the original graph nodes into the $N_e$-dimensional space. The underlying intuition is that pairwise distance in this $N_e$-dimensional space reflects the pariwise similarity defined by $\mathbf{W}$. Thus, partitioning the original graph nodes into multiple groups according to their pairwise similarity may be accomplished by running the K-means algorithm in this embedding space, which is referred to as spectral clustering. The Nyström method was applied in [88] to process large datasets with sparse sampling.

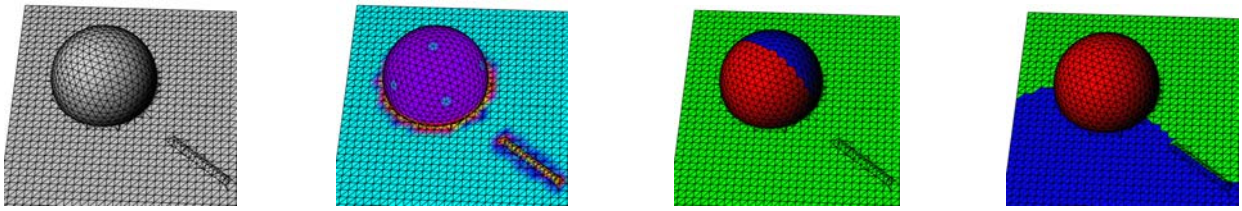
### 4.4.2 High Curvature Feature Extraction

We extract both feature lines and feature points using the process diagrammed in Figure 4.3.

*Curvilinear Feature Detection* For static meshes, we measure features with local curvature estimates [94]. A set of fragmented crestline segments are then obtained by finding local extrema of curvatures [95]. While it is tempting to directly use these crest lines as our feature lines, they are too noisy to represent large scale features. Instead we take these crest line segments into consideration when constructing the similarity matrix $W$ for spectral clustering. Specifically, given a dual graph $G = (F, D)$ with $n_f$ faces and $n_d$ edges, we define a new metric for the similarity term $w(i, j)$ in $W$ over each edge $d = (f_i, f_j)$ as

$$w(i,j) = \beta \exp\left(-\frac{|\kappa_i| + |\kappa_j|}{\sigma_\kappa}\right) \exp\left(-\frac{dist(f_i, f_j)}{\sigma_d}\right) \tag{4.3}$$

where $\beta$ is a scaling factor to emphasize the existence of crest line segments between two faces ($\beta = 0.1$ if there is a crest line between $f_i, f_j$ and $\beta = 1.0$ otherwise), $\kappa_i$ indicates the estimated mean curvature at $f_i$ and $\sigma_\kappa$ is the standard deviation of the absolute mean curvatures among all faces, $dist(f_i, f_j)$ is the geodesic distance between $f_i$ and $f_j$ measured as the total length of the dual edges, and $\sigma_d$ is the standard deviation of pairwise geodesic distance between faces. Since spectral clustering can be applied on a general graph without a valid mesh structure, we also add additional graph connections in the dual graph for nearby non-adjacent faces according to their pairwise distances



Curvature VSA Our Method

Figure 4.5: Illustration of the difference between our spectral clustering and variational shape approximation (VSA) [5]. The testing model has a high curvature feature, a narrow ridge, on the plane. VSA chooses to better approximate the overall shape by dividing the hemisphere into two clusters. Our method is better at feature detection and chooses to align a cluster boundary with the ridge.

$dist(f_i, f_j)$ defined above. Thus we increase the valence of each dual node to improve the results from spectral clustering. This similarity matrix for spectral clustering favors clusters with boundaries along crest line segments or high curvature regions on the mesh.

Other mesh clustering schemes, such as the one presented in Variational Shape Approximation (VSA) [5], can also be used for detecting high curvature features. VSA locally grows triangle clusters according to normal variations. It focuses on approximating the shape of the original mesh with flat regions, while our spectral clustering method focuses on detecting high curvature features on the mesh. The model shown in Figure 4.5 is used to demonstrate the difference between the two methods. This model has a narrow ridge on the plane, which should be regarded as a curvilinear feature. The two methods make different choices according to their clustering criteria. VSA chooses to better approximate the overall shape by dividing the hemisphere into two clusters, while our method chooses to align a cluster boundary with the ridge. In terms of shape approximation, VSA produces better results. However, spectral clustering is preferred in this paper because we would like to detect salient curvilinear features.

Once we have spectral clustering results, as shown in Figure 4.3(d), the cluster boundaries serve as initial candidates of curvilinear features. Although these boundaries roughly follow high curvature features, some of them might be jagged while others may not align precisely with local curvature maxima. Thus further improvement is necessary to identify accurately localized features. We refine initial cluster boundaries by applying a graph minimum s-t cut [89] with different edge weights. We thicken each initial boundary to a boundary region and form a dual graph $G$ using faces within the region as nodes. The actual size of this boundary region could affect the results of refined boundaries. If the size is too large, the new boundary could deviate to some high curvature regions far away from the current boundary. On the other hand, if the size is too small, there will be little room for graph cut refinement. In our experiment, we found that setting the boundary region to cover 10% of the triangles closest to the cluster boundary works well for all our examples. The weights on graph edges are computed using a combination of edge length and

absolute mean curvature. Specifically, we define the edge weight $g(i,j)$ between graph nodes $f_i, f_j$ as :

$$g(i,j) = \beta |e_{ij}| \exp\left(-\frac{|\kappa_i| + |\kappa_j|}{\sigma_\kappa}\right) \tag{4.4}$$

where $e_{ij}$ is the edge shared by $f_i$ and $f_j$ in the original mesh and $\beta$, $\kappa$ are defined similarly as in (4.3). As a result, we favor the shortest path that passes through high curvature regions. This local refinement makes cluster boundaries smoother and better aligned with local curvature maxima, as shown in Figure 4.4.

After refinement, we break cluster boundaries into non-branching segments by finding junctions with more than two incident boundaries. Since cluster boundaries are closed curves, some boundary segments may not lie near features, serving only to close the loop. We discard such segments by checking whether the average magnitude of mean curvature along a segment is smaller than a predefined threshold, which is set to the average magnitude of the largest 30% mean curvatures at all vertices. The remaining segments become the detected curvilinear feature lines. The setting of this threshold affects how many feature line segments will be used as constraints in the chart generation stage. If we set the curvature threshold too small, we may include unimportant boundaries as features. This would impose unnecessary constraints on chart generation, but not necessarily affect reconstruction errors.

*Corner Feature Detection* In addition to curvilinear features, we also identify a sparse set of feature points that are important for preserving sharp corners such as horns or finger tips. As shown in Figure 4.3(f), such corner points typically belong to high curvature regions. These points will also act as constraints in the chart generation process to ensure that the resulting base complex adequately covers these feature regions. We choose corner points as the vertices with the maximum absolute mean curvature in a local neighborhood. The size of this neighborhood is typically set to from 7 to 10 rings. The precise localization of these points is not crucial because the purpose of corner detection is to improve sampling rate by geometry images in high curvature regions. For each detected corner point, we further check whether its absolute mean curvature is sufficiently large and only keep the highest 10% as feature points. The same parameter setting works well for all our testing models and we achieve high reconstruction accuracy even when the corners are not positioned very accurately.

### 4.4.3 Deformation Discontinuity Identification

For dynamic meshes, we also detect deformation discontinuities and incorporate them as additional features for triangle chart generation. Deformation discontinuities can also be viewed as potential high curvature features since, at some frames of a deformation sequence, transformations of triangles across these places can differ significantly and high curvature features can form along these discontinuities. Deformation discontinuities can also aid the mesh skinning process, which needs to extract a set of proxy bones from a mesh deformation sequence [7].

Figure 4.6: Triangle clusters and their boundaries resulted from our new metric for spectral clustering using deformation gradients from the BALLET mesh animation.

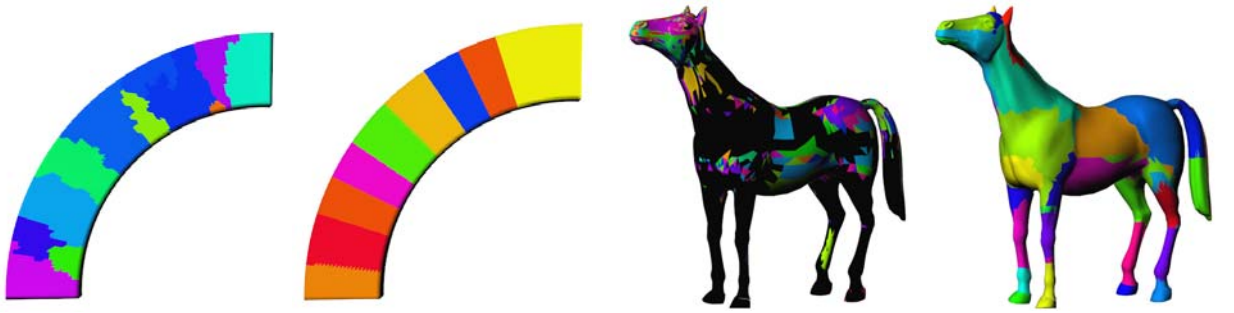
In this section, we introduce a novel metric for detecting deformation discontinuities using spectral clustering. We start by computing the deformation gradient $\Gamma_i^k$ [10] for each face $f_i$ at frame $k$, and use them in the formulation of the similarity matrix $W_b$ in spectral clustering. We define the similarity between two faces according to the similarity of their deformation gradients in all frames as follows,

$$w_b(i,j) = \exp\left(-\frac{\sum_{k=1}^{n_a} \|\Gamma_i^k - \Gamma_j^k\|}{n_a \sigma_\Gamma}\right) \exp\left(-\frac{dist(f_i, f_j)}{\sigma_d}\right), \tag{4.5}$$

where $n_a$ is the number of frames, and $\sigma_\Gamma$ is set to the standard deviation of deformation gradients across both spatial and temporal domains. The cluster boundaries formed using this criterion are also saved as feature lines for chart generation. An example of such feature lines detected as deformation discontinuities is shown in Figure 4.6. The reason for adding the deformation discontinuity as feature lines is to make sure the resulting triangular patches do not cross bending joints such as the elbow of a bending arm. If they were not added, the resulting geometry images could still well represent the static mesh, but would not be sufficiently accurate at lower resolutions during a skinned animation. This is because when a joint is being bent, a flat region around the joint in the static mesh may dynamically become a high curvature region that demands better sampling. Note that we do not break apart the cluster boundaries this time because subsequent proxy bone extraction requires closed regions.

Once we represent each cluster using one proxy bone, we can further compute bone transformations and bone influence weights in a way similar to previous methods that solve least-squares problems [7].

While previous work such as mean-shift clustering [7] or hierarchical clustering [6] can also effectively learn a set of proxy bones, certain drawbacks exist. As shown in Figure 4.7, mean-shift clustering cannot always find a suitable cluster for each face and might result in few and sparse clusters for animation sequences with extreme deformations. On the other hand, hierarchical clustering adapts a bottom-up scheme to greedily merge nearby clusters with similar transformations. While this yields a valid cluster membership for each face, it might fail to recognize



Hier. Clustering Our Method SMA Our Method

Figure 4.7: A comparison of our spectral clustering method with hierarchical clustering [6] and mean-shift clustering [7] on two animation sequences, bending and horse collapsing, respectively. For the horse model, mean-shift clustering fails to assign a large number of triangles (shown in black) to any clusters due to their highly deformable nature. Our method results in more regular cluster shapes and a spatially more coherent assignment of the triangles to the clusters.

global deformation characteristics since only local merging is performed at each step. Therefore it usually results in irregularly shaped clusters even for simple and well-behaved deformation such as bending. As shown in Figure 4.7, our method works better in identifying the deformation characteristics from bending and results in triangle clusters with a more regular shape. The advantage of our method lies in that it is a global clustering technique that better preserves spatial coherence. These properties make it more robust in learning the set of proxy bones for mesh animations with extreme deformations.

## 4.5 Triangular Geometry Image Construction

When an input mesh is converted to a set of triangular geometry images, there are two constraints we would like to impose for triangular chart generation. First, each chart boundary should be shared by exactly two adjacent charts without T-junctions. This constraint makes it straightforward to construct seamless atlases for geometry images and simplifies boundary stitching at the rendering stage. Second, chart boundaries should be aligned with detected curvilinear features. This constraint is important for level-of-detail rendering since it helps preserve features even at lower resolutions. We therefore design our chart generation process to enforce these constraints. Detailed description of feature detection can be found in Section 4.4.

### 4.5.1 Triangular Patch Generation

Many patch formation methods rely on cluster growth, but growing triangular clusters that everywhere share boundaries with exactly three other clusters would be difficult. Our triangle patch generation is based on mesh simplification and the progressive parameterization provided by MAPS [8]. We parameterize the original mesh over a base complex which is an extremely simplified version of the original mesh, and then compute patch boundaries on the original

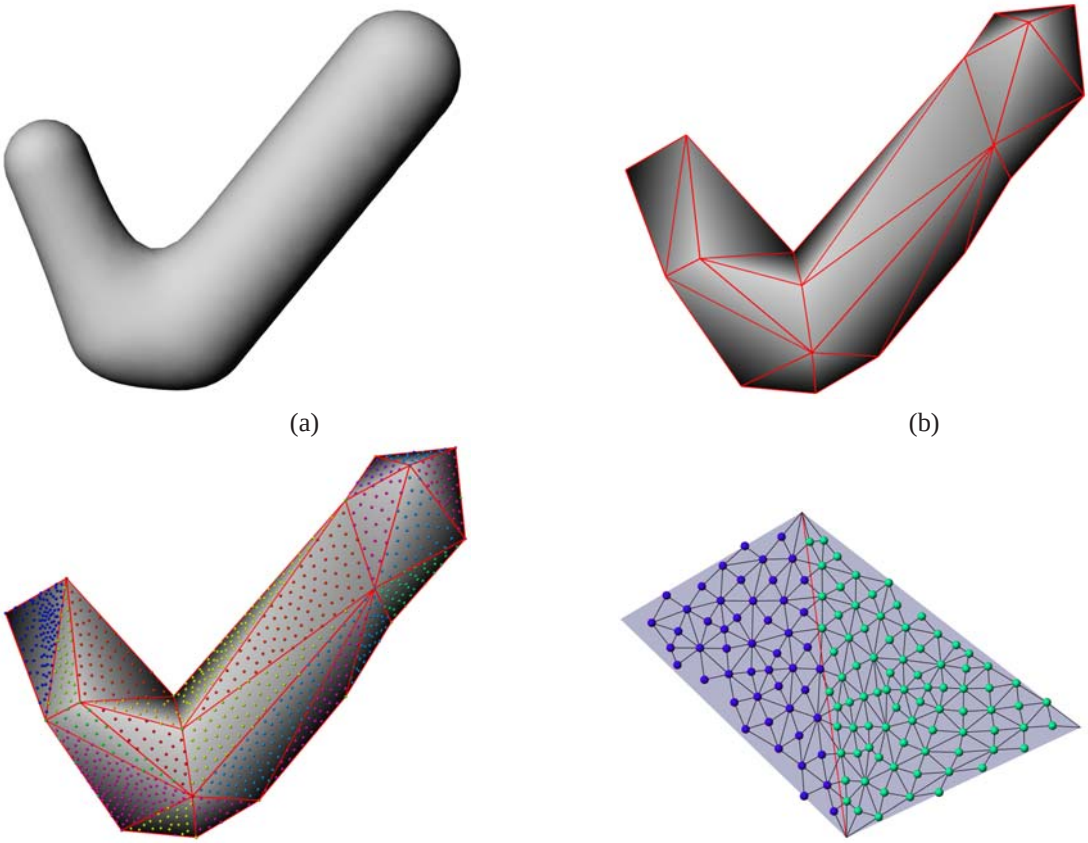


Figure 4.8: The overall process of triangle patch generation. Start from an input mesh in (a), we first perform mesh simplification to generate a base mesh in (b). During simplification, we apply MAPS [8] to progressively parameterize the input mesh over the base mesh, as shown in (c). We utilize this parameterization to define a path in a flattened mesh for each base domain edge, as shown in (d). These paths are mapped onto the original mesh to define the boundaries of triangle patches.

mesh to create triangular patches there. Every patch boundary on the original mesh corresponds to an edge in the base complex.

As shown in Figure 4.8, we simplify the input mesh to a base complex $M^s = (V^s, E^s, F^s)$ through series of "half-edge collapses" that use one of the edge's two original vertices as the new vertex position [19]. Thus $V^s \subset V$. We prevent the collapse of any edge that connects a feature vertex with a non-feature vertex, which ensures the base domain triangles do not cross feature curves and do not absorb feature points, while allowing feature curves themselves to be simplified. During mesh simplification, we progressively build a parameterization of the original mesh using MAPS [8].

MAPS is a global parameterization method that parameterizes an original mesh over a simplified base complex. Every vertex in the original mesh is assigned a membership to a base domain triangle $f^s \in F^s$ as well as barycentric coordinates $(\alpha, \beta, \gamma)$ with respect to $f^s$. Therefore vertices assigned to the same base domain triangle share the same coordinate frame, as shown in Figure 4.8(C). Moreover, vertices assigned to two adjacent base domain triangles

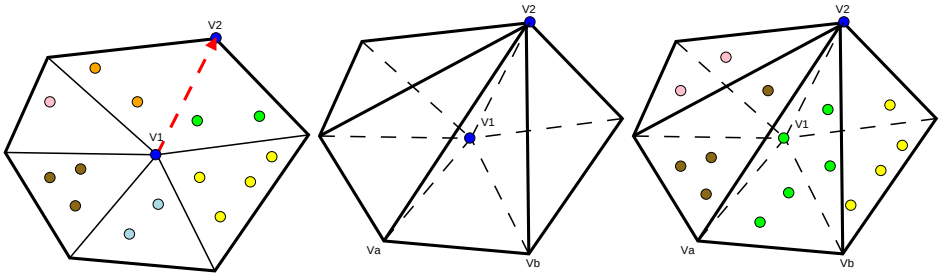



Figure 4.9: One step of MAPS parameterization. (A) For an edge $(v_1, v_2)$ with $v_1$ collapsed onto $v_2$, $V_d$ represents the subset of previously removed vertices (shown in colors) parameterized over $v_1$'s one-ring neighborhood. (B) After the edge collapse, the one-ring neighborhood has a new triangulation. We assign $v_1$ to a triangle $f = (v_2, v_a, v_b)$ in the new triangulation, and compute its barycentric coordinates. (C) Similarly, we reassign vertices in $V_d$ to triangles in this new triangulation and update their barycentric coordinates.

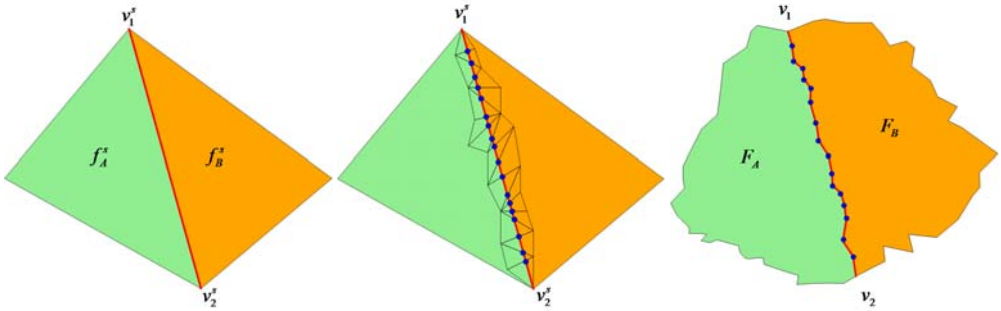


Figure 4.10: Overview of path generation. Every edge $e^s = (v_1^s, v_2^s)$ in the base mesh (left) has a corresponding path $P = (v_1, v_2)$ in the original mesh (right). To generate such a path, we flatten a local region on the original mesh and intersect $e^s$ with triangles in the flattened region. A straight path from $v_1^s$ to $v_2^s$ is traced by inserting Steiner vertices at the intersections (middle). This path is mapped back to the original mesh to form a path between $v_1$ and $v_2$.

$f_1^s, f_2^s \in F^s$ can also be expressed in the same coordinate frame by flattening $f_1^s$ and $f_2^s$ onto the same plane. This will become very useful when mapping a base domain edge to a path on the original mesh.

To obtain the aforementioned global parameterization, MAPs progressively generates and updates the memberships and barycentric coordinates of the removed vertices after each half-edge collapse. As shown in Figure 4.9(A), let $e = (v_1, v_2)$ be an edge to be collapsed, where $v_1$ will be moved to $v_2$. For the removed vertex $v_1$, MAPs first obtains the set of triangles, $F_{v_1}$, in its one-ring neighborhood, along with the complete set of previously removed vertices $V_d$ currently parameterized over $F_{v_1}$. It then flattens the one-ring neighborhood defined by $F_{v_1}$ using a harmonic map, which will result in a 2D coordinate frame to represent vertices in $F_{v_1}$. $v_1$ and all vertices in $V_d$ are then expressed in this 2D coordinate frame. After the edge collapse, as shown in Figure 4.9(B), the one-ring neighborhood defined by $F_{v_1}$ is retriangulated and the new set of triangles is denoted by $F'_{v_1}$. One can easily reassign $v_1$ or a vertex in $V_d$ to a triangle in $F'_{v_1}$ and recompute its new barycentric coordinates, as shown in Figure 4.9(C). This process is repeated for each edge collapse, and eventually all removed vertices become parameterized over base domain triangles in $F^s$.

Once we have the global parameterization from MAPS, patch generation becomes a straightforward process. As

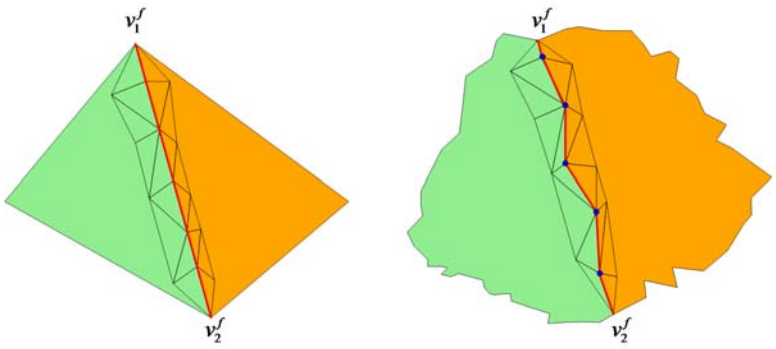



Figure 4.11: Feature preservation in MAPs parameterization. For a base domain edge $e^f = (v_1^f, v_2^f)$ collapsed from a feature curve $P^f$, all vertices on $P^f$ can be parameterized on $e^f$ (left). Thus the straight path in the flattened mesh directly corresponds to the feature curve $P^f$ in the original mesh.

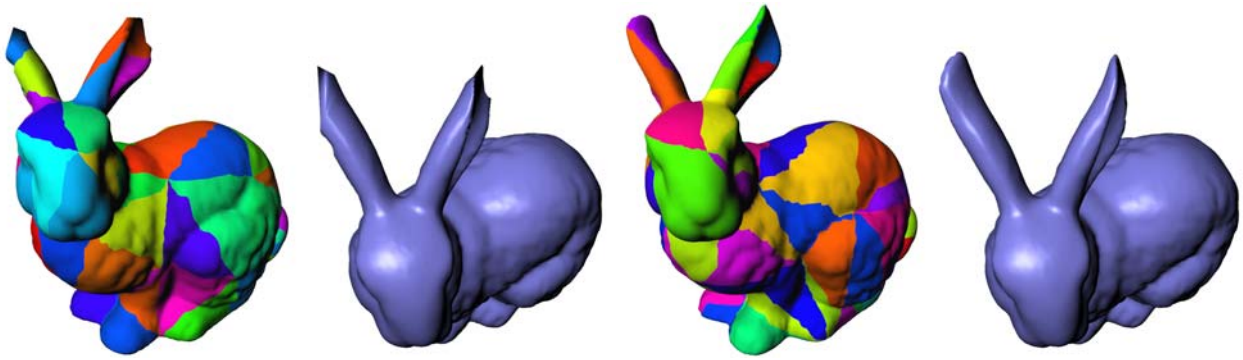
shown in Figure 4.10, a base domain edge $e^s = (v_1^s, v_2^s) \in E^s$ is shared by two base triangles $f_A^s$ and $f_B^s$, which can be unfolded to a planar quadrilateral with $e^s$ being one of its diagonals. The part of the original mesh parameterized over $f_A^s$ and $f_B^s$ can be flattened onto the same planar region. We collect the set of triangles $F_e \in F$ from the original mesh that intersect with $e^s$ in this flattened configuration. Tracing a path between $v_1^s$ and $v_2^s$ in the flattened mesh can be achieved by inserting Steiner vertices at those intersections. This is similar to previous work for tracing a path on a polygonal mesh [96, 97, 98]. This path determined by $e^s$ is finally mapped back to the original mesh to define a path between $v_1$ and $v_2$.

When building the MAPS parameterization, we detect and fix any triangle flips in the parametric domain [8] to ensure that a straight line in the parametric domain always maps to a topological line in the original mesh. Therefore the above method guarantees to produce a valid path for each base domain edge. The MAPs algorithm can also integrate feature-related constraints into the parameterization to ensure a traced path is aligned with a feature curve. As shown in Figure 4.11, for a base domain edge $e^f = (v_1^f, v_2^f)$ simplified from a feature curve $C^f$, all vertices on $C^f$ can be parameterized on $e^f$. Therefore the path corresponding to $e^f$ in the parametric domain can be mapped trivially to the feature curve $C^f$ on the original mesh.

Fig. 4.1 shows a base complex and triangular patches from an original mesh using mesh simplification. The packed triangular geometry images are also shown in the bottom. Fig. 4.12 validates how well patch generation preserves features by comparing the reconstruction quality with and without feature constraints. Ordinary simplification can obscure some features, such as the ears, which directly affects their sampling rate and can cause both numerical errors and visual artifacts in the geometry image representation.

### 4.5.2 Patch Parameterization and Packing

For each triangular patch, we parameterize it onto a 2D triangular domain by fixing its boundary onto the edges of a right triangle. We apply the parameterization algorithm in [99] to ensure no triangles flip in the embedding. The



Without Features With Features

Figure 4.12: Patches generated without feature constraints might not align their boundaries well with high curvature regions. Therefore the resulting reconstruction has more numerical and visual errors in these regions, such as the ears on the bunny.

resulting parameterization is then resampled onto a regular grid within the right triangle with $2^{d_{max}} + 1$ samples along every edge. Since each patch may have different geometric complexity, we determine a maximum resolution level $d_{max}$ for a patch according to its size and curvatures. Specifically, we set $d_{max}$ to be proportional to the curvature weighted sum of face areas of this patch. Trivially packing each single patch into its square bounding box would be a waste of space since it only occupies about half of the bounding box.

An intuitive way to improve packing efficiency is to exploit patch adjacency by packing pairs of neighboring patches into a single square image. Since two adjacent patches share the same geometric information along their shared boundary, a single shared boundary can be stored along the diagonal pixels in the image. One drawback of this method is that there will be leftover patches which do not have any neighboring patches to pair with. Therefore we need to allocate more space for these patches than necessary as they have to be packed individually. Moreover, if two adjacent patches have different maximum resolutions, we have to allocate a square region sufficiently large for the higher resolution patch to pack both of them together. Therefore this method is still suboptimal in terms of spatial efficiency.

Therefore we have chosen to pack two triangular patches that are not necessarily adjacent into a rectangular image of size $(2^{d_h} + 2) \times (2^{d_h} + 1)$, where $d_h$ represents the higher resolution of the two patches, for best spatial efficiency. The rectangular shape is due to the fact that pixels along the diagonals are from two separate patch boundaries and both boundaries need to be preserved. In this scheme, we maintain a sorted list of patches based on their maximum resolution, and always pack a pair of patches with closest maximum resolutions together. We take special care when building a geometry image pyramid to ensure that the downsampling filter only considers pixels from the same patch. This packing technique is similar to the schemes originally designed for texture atlases consisting of triangular texture maps [100, 101].

*Packing Skinning Parameters.* For skinned meshes, we also resample the bone influence weights, originally

stored at the vertices, onto the regular grid in a similar manner to geometry resampling. In order to improve run-time efficiency, we only keep the four largest bone influence weights for every grid point and store both the weights and their associated bone indices. This ensures that the total storage for resampled weights is fixed and independent of the total number of bones in the mesh.

## 4.6 GPU-Based Level-of-Detail Rendering

At run-time, we render each triangular patch in the form of a triangular geometry image. A suitable resolution of the geometry image is determined on the fly for each patch according to the projected screen area of its oriented bounding box (OBB). To further adapt our LOD selection scheme to a dynamically skinned mesh, the bounding box of each deformed patch is also deformed before its screen projected area being used to estimate LOD of the patch. To avoid cracks along boundaries, we apply an automatic boundary stitching method to connect adjacent patches with different resolutions in our GPU implementation.

### 4.6.1 Level of Detail Selection

We use an OBB to approximate the geometry on each patch when computing its LOD. The projected screen area of the OBB is used to determine an appropriate resolution for each patch. In order to compute the projected area of an OBB, only four of the eight corners of the OBB are transformed to form three new major axes of the transformed bounding box. Since there are always three adjacent faces of the OBB are visible, the projected area of the OBB can be computed with the cross-product of these major axes. However, this requires precomputing an extra set of bone influence weights for each corner point of the OBB from bone transformations and the actual OBB corner vertices in every input deformation sample data. At run-time, we apply new bone transformations to the OBB corners and use the deformed bounding box for LOD selection. Specifically, given a set of corners $c_i, i = 0 \dots 3$ which form the major axes of an OBB with $c_0$ being the pivoting point, and their skinning weights $w_i^b, b = 1 \dots n_b$, we can compute the projected area $A$ as

$$A = \sum_{i>k} \|(\hat{c_i} - \hat{c_0}) \times (\hat{c_k} - \hat{c_0})\| \tag{4.6}$$

where $\hat{c_i} = \mathbf{M_p} \left( \sum_{b=1}^{n_b} w_i^b \mathbf{T_b} c_i \right)$ is the screen projection of the transformed corner vertex $\mathbf{c_i}$ using the current camera view-projection matrix $M_p$. For a static mesh, we simply use $\hat{c_i} = \mathbf{M_p} c_i$ and apply the above equation to obtain the screen projected area. Once we obtain the projected area $A$, we determine the detail level for this patch according to both the area and its maximum detail level $d_{max}$. Since one higher resolution increases the number of rendered

triangles by four times, we compute the current level of detail $d$ as follows.

$$d = \max(\log_4(\alpha A), d_{max}) \tag{4.7}$$

where $\alpha$ is a scaling factor such that $\alpha A_0 = 4^{d_{max}}$ with $A_0$ equal to the screen size of the display window. This ensures that the coverage ratio of triangles over pixels is approximately constant at all detail levels. We implement both the dynamic skinning and LOD selection processes on the GPU using the CUDA programming environment and store the results into GPU video memory in preparation of geometry image rendering at the next stage.

Although it would be more straightforward to implement the above LOD selection on the CPU, the actual performance depends on the complexity of the LOD computation. For animated meshes, corners of their bounding boxes need to be skinned and projected to obtain their approximate screen projected area. This LOD computation becomes more significant when we perform LOD rendering for a number of animated characters, each with hundreds of charts. This computation can be performed much faster on the GPU and thus motivate our GPU implementation for LOD selection.

### 4.6.2 Geometry Image Rendering

Since the multi-chart mesh geometry datasets generated in our preprocessing stage is in the form of multi-resolution 2D images, they can be easily stored in the GPU video memory and preloaded as textures for real time rendering. Since the latest G80 hardware supports texture array extension, geometry images at the same resolution can be stored in the same texture array. This texture array data organization has effectively reduced the overhead incurred from calling the OpenGL API functions since the texture binding only need to be performed once for each detail level. Since we precompute all resolutions of geometry images, mip-mapping is disabled during rendering. Although we use textures for storing geometry images, they are primarily used as the medium for storage on the GPU. By turning off automatic texture filtering when accessing geometry images on the GPU, we access them as 2D arrays instead of filtered textures.

A set of triangular grids are precomputed at all necessary resolutions and packed as OpenGL Vertex Buffer Objects (VBOs). Each grid point is associated with a pair of (u, v) texture coordinates. Since the texture coordinates are independent from the actual geometry stored in the rectangle textures, they can be reused for different patches. These triangular grids are also stored and preloaded into the GPU video memory for the rendering pass. During rendering, we select a grid resolution corresponding to the chosen LOD of a patch and directly render it with texture mapping turned on. In a vertex shader program, the texture coordinates of each grid point are used to look up vertex positions in a geometry image. Additional information such as normal vectors, bone influence weights can also be looked up in a

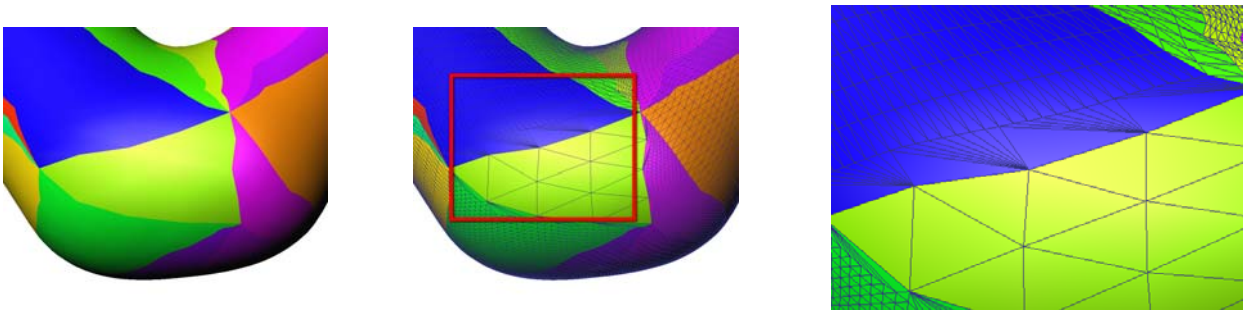


Figure 4.13: Illustration of our boundary stitching method. (Left) Visualization of triangle charts on the mesh. Each color represents a distinct chart. (Middle) Stitching result along chart boundaries. (Right) Closer view of the stitching result. Although two adjacent charts have a significant difference in LOD, the stitching results are guaranteed to be watertight.

similar manner. The final patch geometry can therefore be rendered in the vertex shader after skinning transformations have been applied.

### 4.6.3 Boundary Stitching

When different resolutions have been chosen for adjacent patches, cracks will occur at their common boundary. Since adjacent patches share the same geometry along their common boundary, we can perform simple stitching directly on the GPU by moving the vertices on the higher resolution border to those vertices on the lower resolution one. This is done by recalculating the texture coordinates for grid points on the higher resolution boundary so that they can be used to access the texels on the lower resolution one. With the latest shader API function texelFetch, we can access the integer texture coordinates directly within the range $(0 \dots w-1, 0 \dots h-1)$. Given two patches $P_i, P_j$ with resolution $r_i = 2^n + 1, r_j = 2^m + 1, n > m$ respectively, new texture coordinates $(\bar{u}, \bar{v})$ can be computed from the original texture coordinates $(u, v)$ along the shared boundary of $P_i$ as :

$$\bar{u} = u - u \bmod 2^{(n-m)}, \bar{v} = v - v \bmod 2^{(n-m)} \tag{4.8}$$

In our vertex shader implementation, the edge vertices along each boundary are identified. The texture coordinates of these vertices are then modified according to (4.8) to ensure that the vertex positions retrieved from the geometry image correctly align with the adjacent patch.

As shown in Figure 4.13, the above stitching method guarantees no seams along chart boundaries. This is because the boundary vertices of a chart at a lower detail level is always a subset of boundary vertices of any chart at a higher detail level. However, it is possible to have triangle flips when detail levels between adjacent charts vary significantly. In our results, this rarely happens and does not generate discernible artifacts.

Figure 4.14: Triangular charts and reconstructed meshes at varying levels of detail for a BALLET animation.

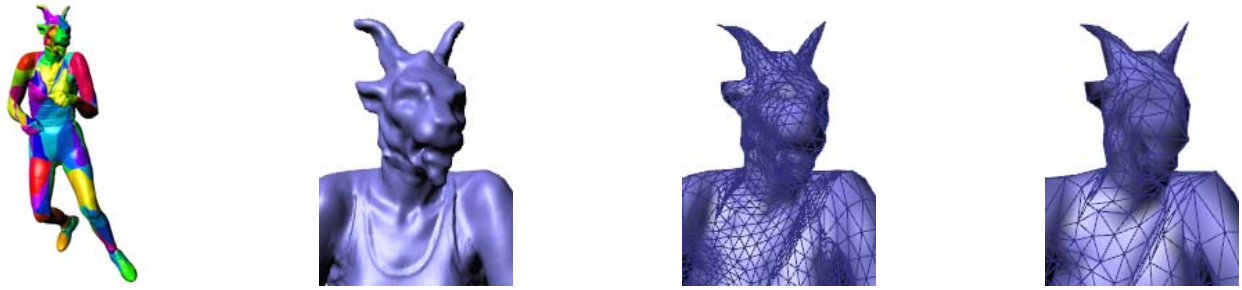


Figure 4.15: Triangular charts and reconstructed meshes at varying levels of detail for a BOXING animation.

## 4.7 Experimental Results

We have successfully tested our method on both static and deforming meshes. The triangular charts generated with our method and their reconstruction results can be found in Figures 4.12, 4.19, 4.14, and 4.15. The results for static meshes are shown in 4.12 and 4.19, and those for deforming meshes are shown in 4.14 and 4.15. Timing and statistics for the preprocessing steps can be found in Table 4.1. The skinning quality of oriented bounding boxes for triangular charts is shown Figure 4.16. Since we also treat deformation discontinuities as feature lines during chart generation, the resulting charts do not cross boundaries between regions that are primarily controlled by different bones. Thus we can accurately fit the bounding box deformations and use the skinned bounding boxes when estimating the level of detail for different poses.

| Examples | #Orig. Tris | Feature Time | Chart Time | #Charts | Max. Resolution | Data Size |
|---|---|---|---|---|---|---|
| V2 | 8K | 0 min | 1 min | 34 | $2^5$ | 0.5 MB |
| Bunny | 70K | 2 min | 4 min | 70 | $2^6$ | 5.2 MB |
| Isis | 100K | 3 min | 6 min | 120 | $2^6$ | 8 MB |
| Feline | 100K | 3 min | 9 min | 250 | $2^6$ | 17 MB |
| Ballet | 350K | 5 min | 19 min | 250 | $2^6$ | 42 MB |
| Boxing | 250K | 5 min | 12 min | 150 | $2^6$ | 24 MB |
| Grand Canyon | 6M | 15 min | 140 min | 840 | $2^7$ | 316 MB |

Table 4.1: Statistics and Timing. All performance measurements were taken from a 3.0GHz Pentium D processor. '#Orig. Tris.' means the number of triangles in the original mesh, 'Feature Time' means the pre-processing time for feature extraction, 'Chart Time' means the time for chart generation, '#Charts' means the number of resulting triangular charts, and 'Max Resolution' means the maximum resolution for each chart.

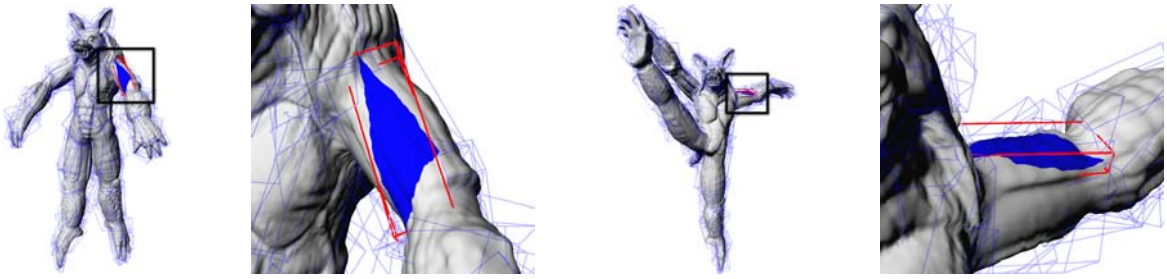
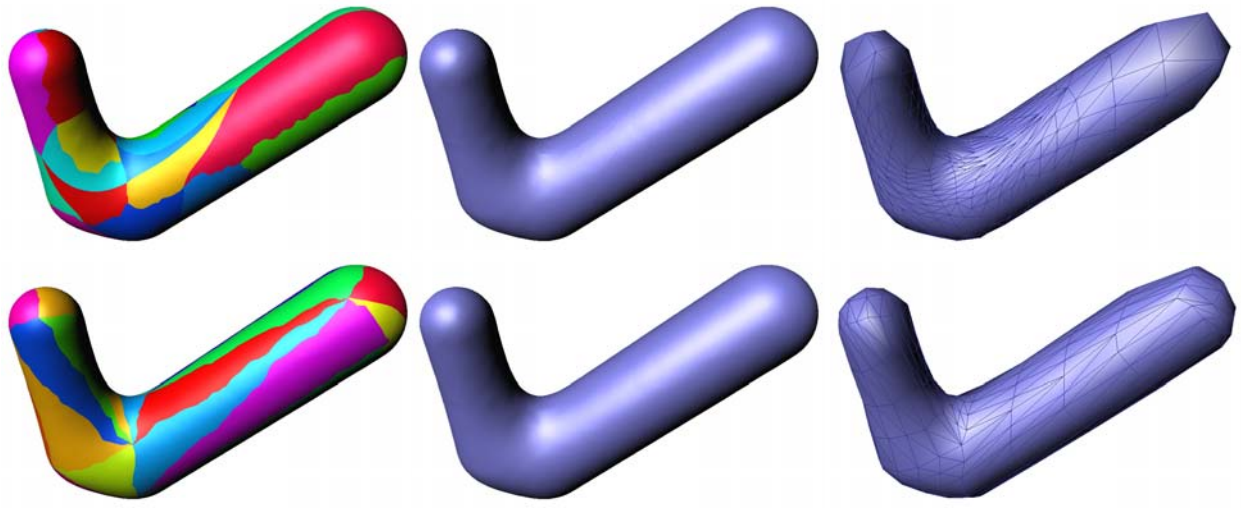


Figure 4.16: Skinning results of an oriented bounding box for different poses. The red bounding box is associated with the patch in blue color. The skinned corners of the bounding box adequately approximate the bounding volume of the deformed patch at every different pose. Therefore we can use the skinned bounding box to estimate the projected screen area and thus determine the detail level for this chart at every pose.



Charts 13K Vertices 200 Vertices

Figure 4.17: A comparison between our triangle-char geometry images (Bottom Row) and quad-chart geometry images [1](Top Row). For this simple model, both methods approximate the original mesh well at a high resolution. However, quad charts tend to have irregular shapes and produce lower quality results at lower resolutions.

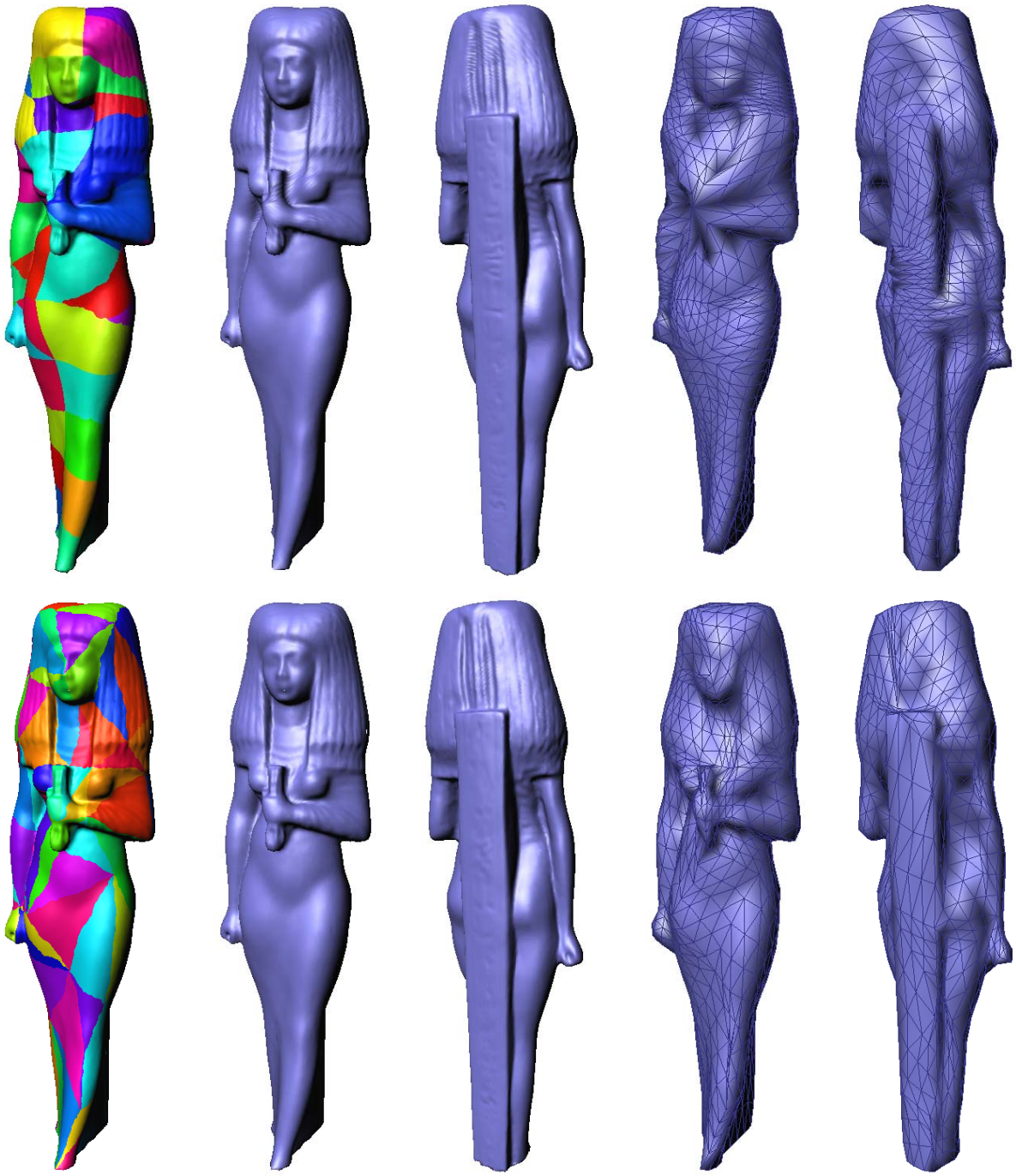
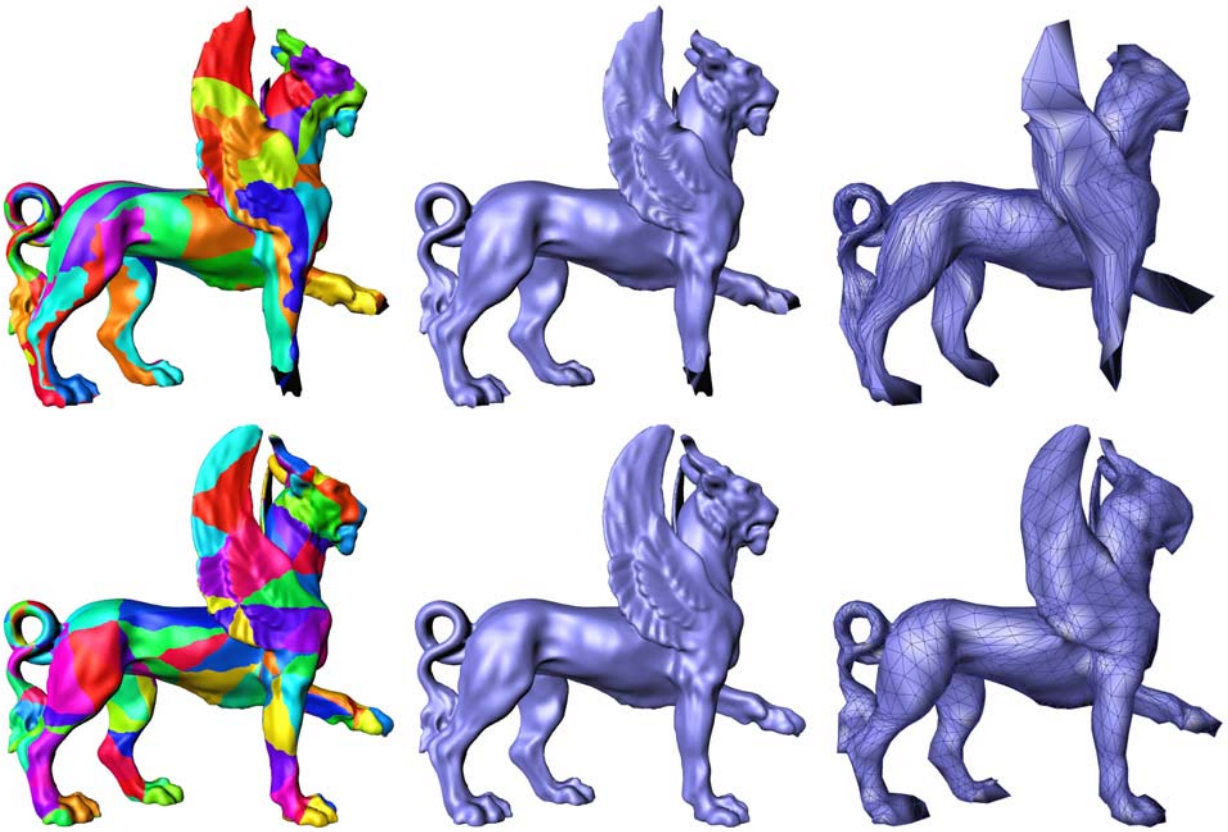



Figure 4.18: Another comparison between our triangle-chart geometry images (Bottom Row) and quad-chart geometry images [1] (Top Row) using the Isis model. Although this model has a relatively simple shape, it also contains sharp edges and semantic features. Both methods can produce a good reconstruction in a high resolution. However, quad charts fail to reconstruct important features faithfully in a low resolution while our method still gives a good approximation.



Charts 130K Vertices 1.5K Vertices

Figure 4.19: Another comparison between our triangle-chart geometry images (Bottom Row) and quad-chart geometry images [1](Top Row). With the same number of vertices, the reconstructed meshes from our method is more faithful to the original mesh than the quad-chart based method. The feature constraints in our method ensure that important features are preserved during mesh simplification and result in higher-quality charts.

We have compared both numerical errors and visual quality between triangular geometry images from our method and the quad-images generated from seamless texture atlas (STA) [1]. In our comparison, we use the same number of vertices for both methods and geometry images from both methods have an equivalent resolution. Numerical errors of the meshes reconstructed from geometry images are obtained using the method in [102] which computes the Hausdorff distance between the original mesh and the reconstructed mesh. As shown in Figures 4.17-4.19 and Table 4.2, triangular geometry images from our method give rise to smaller numerical errors and better visual reconstruction results, especially around feature regions with high curvature. We found that STA usually performs adequately for examples with a simple shape, such as the Isis model. However, for examples with a high genus or with protruding features, STA tends to produce poor results. Since the face clustering scheme in STA needs to satisfy multiple topology constraints and perform additional steps to generate a quadrangulation, the resulting shape of the charts are usually much more irregular, which in turn give rise to more distortion and lower quality surface reconstruction. While using adaptive charts could improve its quality, the main source of the distortion comes from badly shaped charts during

| Examples | #Tri. Charts | #Tri. Vetices | Tri. Error | #Quad Charts | #Quad. Vetices | Quad. Error |
|---|---|---|---|---|---|---|
| Bunny | 70 | 38K | 0.010 | 42 | 45K | 0.309 |
| Feline | 250 | 130K | 0.019 | 150 | 163K | 0.150 |
| Ballet | 250 | 130K | 0.0080 | 138 | 150K | 0.129 |
| Boxing | 140 | 82K | 0.007 | 78 | 85K | 0.074 |
| Isis | 120 | 67K | 0.0036 | 66 | 71K | 0.0146 |
| V2 | 34 | 19K | 0.0016 | 24 | 26K | 0.0022 |

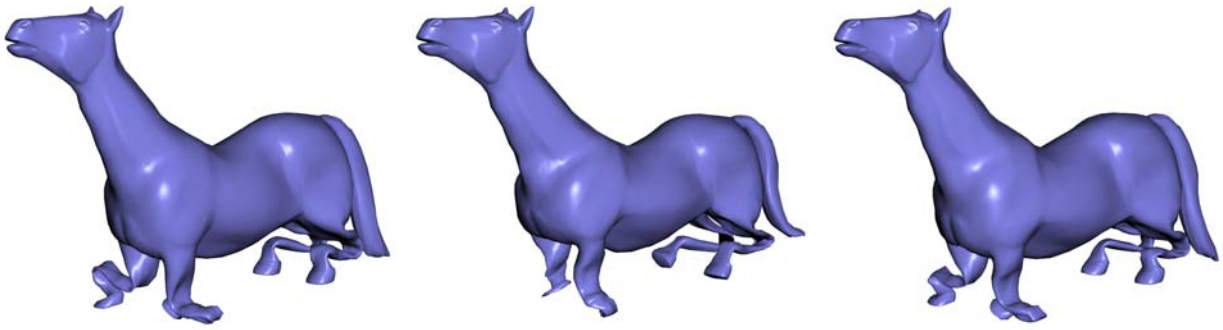
Table 4.2: Comparison of mesh reconstruction errors between our triangular geometry images and quad-chart based geometry images [1].

Ground Truth Hierarchical Cluster Our Method

Figure 4.20: Comparison of skinning quality between our method and the hierarchical clustering method in [6]. Our method produces results with less artifacts.

hierarchical clustering. Moreover, it is more difficult to integrate the feature constraints in our method into their face clustering scheme. On the other hand, since the generation of our triangular charts is based on triangle mesh simplification, it is straightforward to add feature constraints in the framework and ensure a sufficient number of charts in the feature regions. As shown in Figure 4.18, although quad charts produce a good reconstruction in a high resolution, it fails to reconstruct important features faithfully in a low resolution.

We have also compared the quality of mesh skinning using proxy bones extracted with spectral clustering. It has been shown in Section 4.4.3 that spectral clustering can extract higher-quality proxy bones faithful to the deformation structure. Here we further compare the resulting skin animations between our method and existing ones. In Figure 4.20, we show that the skinning results from spectral clustering are closer to the ground truth without noticeable



Ground Truth SMA Our Method

Figure 4.21: Comparison of skinning quality between our method and SMA [7]. Our extracted proxy bones fit the mesh sequence well while SMA fails in highly deformable regions including the bending legs.

Figure 4.22: Level-of-detail rendering of a large BOXING crowd.

| Demo Scene | #Total Tri. | Avg. Throughput | Avg. FPS |
|---|---|---|---|
| Boxing Crowd | 15.3M | 85M/s | 45 |
| Ballet Crowd | 12.8M | 84M/s | 40 |
| Bunnies & Felines | 22.1M | 105M/s | 27 |
| Grand Canyon | 6.8M | 110M/s | 50 |

Table 4.3: Performance of our LOD rendering system on four composed large scenes. The first two scenes have large collections of dynamically animated meshes using linear blend skinning, and the last two are static scenes. Performance were measured from nVidia Geforce 8800GTS 640MB VRAM.

artifacts, while the irregular cluster shapes from hierarchical clustering, adopted in [6], tend to cause more obvious artifacts. In Figure 4.21, our result is compared with that from SMA [7] on an extreme horse collapsing sequence. Since our method produces a clustering that is more spatially coherent, the skinning results are more faithful to the ground truth. On the other hand, SMA tends to produce sparse clusters on highly deformable meshes and fails to reconstruct the deformation at the front legs. To clearly show the differences, all results in this comparison were generated from skinning only without applying displacement corrections as proposed in [7].

We have created four large scenes of both static and skinned models to demonstrate our level-of-detail rendering system. Two examples of these scenes are shown in Figures 4.23 and 4.22. The rendering performance and other statistics of these scenes can be found in Table 4.3. It can be seen that including a large number of dynamically animated objects using our LOD representation only moderately compromises the rendering performance. Adding additional unique characters would surely increase the required amount of video memory and increase the number of rendering passes. However, since each skinned character requires only about 40MB of texture memory, as shown in Table 4.1, the current generation GPU should be able to hold up to tens of distinct high-resolution characters in its texture memory. If only a few distinctive characters are required for a scene, there should be plenty of memory left

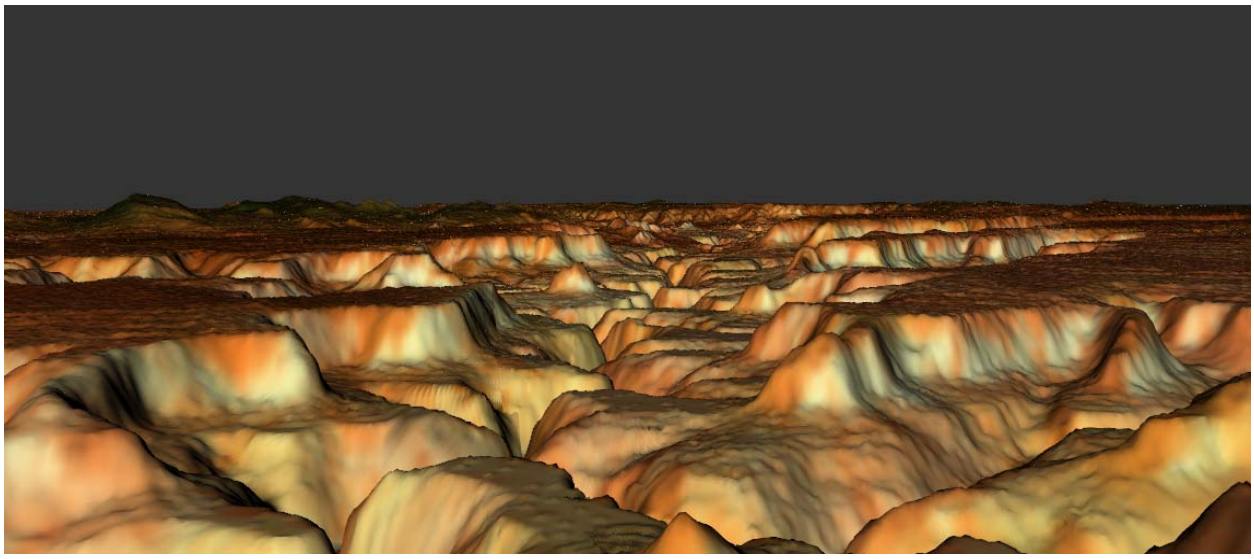


Figure 4.23: Level-of-detail rendering of a terrain navigation using the Grand Canyon dataset.

for other GPU operation such as texturing and shading.

## 4.8 Conclusions and Future Work

In this paper, we introduce multi-chart triangular geometry images for GPU-based LOD representation of both static and deforming objects. To fulfill the promises of triangular geometry images, we have developed a series of algorithms for the detection of curvilinear features, for the construction of such geometry images and their LOD representations, as well as for GPU-based LOD rendering. We have also generalized these algorithms for dynamically skinned meshes.

There are limitations that we would like to address in our future work. First, the optimal resolution level is only determined once per frame for each triangular geometry image. This coarse-grain scheme does not hinder rendering performance for geometry images with a reasonable size. However, when the geometry images become excessively large, a single resolution per patch would not be sufficiently adapted at the desired level of detail in each local surface region. It would be more practical to integrate a dynamic subdivision scheme on the geometry images as in[18] and render different subimages using different resolutions. Second, it is assumed in this work that all the geometry images can be preloaded into the GPU video memory for maximal rendering throughput. Since most of models used in our demo required only at most 40 MB of video memory, this assumption will not pose as a problem for rendering a scene with tens of distinct characters using the current generation of GPUs. However, for rendering many different characters in a high resolution, the amount of required video memory might exceed the capacity of the GPU. In future, we would like to develop an out-of-core system for gigantic mesh models that cannot fit into the GPU video memory. A dynamic paging scheme should be designed to swap geometry images between the video memory, system memory

and hard drives.

# Chapter 5

# Precomputed Radiance Transfer for Skinned Mesh

Computer games and real-time applications frequently adopt mesh skinning as a deformation technique for virtual characters and articulated objects. Rendering skinned models with global shading effects, such as interreflection and subsurface scattering, using precomputed radiance transfer enables high-quality real-time display of dynamically deformed objects. In this approach, we need to precompute radiance transfer for many sampled poses. Resulting datasets reach hundreds of gigabytes, and are orders of magnitude larger than those for a static object. This paper presents simple but effective large-scale data management techniques so that runtime data communication, decompression and interpolation can be performed efficiently and accurately. Specifically, we have developed a mesh clustering technique based on spectral graph partitioning to facilitate interpolation from nearest neighbors and an incremental clustering method for transfer matrix compression. By exploiting additional data redundancies among different sampled poses, we can achieve higher compression ratios with the same fidelity. Our incremental clustering can make the runtime cost of per-frame data decompression and interpolation satisfy a prescribed upper bound. As a result, we can achieve real-time performance using the massive precomputed data and an efficient runtime algorithm.

## 5.1 Introduction

Precomputed radiance transfer (PRT) provides the opportunity to produce compelling realism with global shading effects in real time. Originally developed for static scenes [23, 24], PRT has been subsequently extended to fixed animation sequences as well as deformable objects with shading effects caused by detailed surface features but without cast shadows [25]. It has also inspired techniques that produce soft shadows for dynamic scenes [26, 27] as well as algorithms for real-time lighting design [28] and cinematic relighting [29]. However, the generalization of PRT to dynamic scenes with global shading effects, such as interreflection and subsurface scattering, has not been very successful. Our goal is to overcome this limitation on the class of deformable objects generated by skinning [49, 20], which is the most widely adopted deformation technique for virtual characters and articulated objects in computer games and real-time applications.

In skinning, there is a set of rigid “bones”. Surface deformations are defined as functions of the rigid movements of

Figure 5.1: PRT-based real-time rendering of dynamically deformed objects with global shading effects.

nearby bones [49, 20, 7]. Such a nature gives rise to deformations at two different scales. The relative position among large surface segments, such as the limbs of a virtual character, may undergo large-scale changes. Nevertheless, large-scale movements are highly correlated. For example, a virtual character is typically designated with a few classes of whole body movements, such as walking, running and dancing. Each class of movements exhibit a high degree of coordination among various body parts [103]. Subspaces that enclose relevant poses can be identified with a covariance analysis on sample movements. At a smaller scale, deformations within each surface region, such as the bulging of muscles, are smoothly varying, and spatially close points share similar deformations.

In this paper, we focus on real-time PRT techniques that can achieve realistic global shading effects, such as interreflection and subsurface scattering, on glossy or translucent surfaces deformed by skinning. Most importantly, such surface deformation is not precomputed, but dynamically generated. To be able to produce these global shading effects, we take the example-based approach that draws many samples in the pose subspaces for a particular object and precomputes radiance transfer for them. Note that during runtime a dynamically generated pose as well as its associated surface deformation may be different from all the sampled poses. Therefore, some type of interpolation from nearest neighbors in the pose space is inevitable.

An important question is what criteria or distance metrics we should use to search for these nearest neighbors. Pose similarity should obviously be considered. But pose similarity alone is insufficient. Even a subspace of poses typically has multiple dimensions. Any practical number of sampled poses only give rise to a sparse sampling within such a space, which results in large interpolation errors. On the other hand, the configuration of a single surface segment lies in a much lower dimensional space. Therefore, we search for nearest neighbors for each surface segment separately and then perform interpolation using different nearest neighbors for different surface segments. When searching for such segment-wise nearest neighbors, we also consider similarity in terms of local PRT data (i.e. the precomputed radiance transfer matrices). Unlike local geometry, radiance transfer matrices over a surface segment actually account

for global effects such as scattering and interreflections caused by other surface segments.

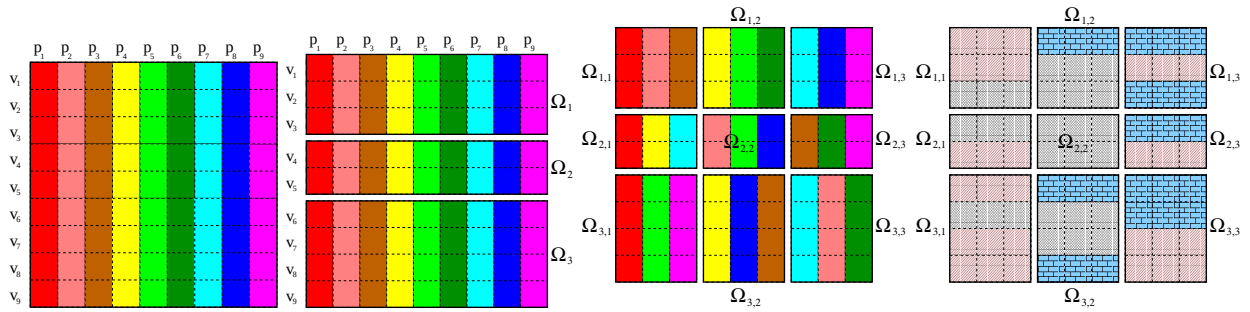
Another tough challenge we face is the sheer size of the data generated by precomputing. Because we need to precompute radiance transfer for every sampled pose, resulting datasets reach hundreds of gigabytes, and are orders of magnitude larger than those for a static object. We need clustering and compression methods that are better suited for such large-scale data for the following reasons. First, compressed datasets need to fit into the system memory. Otherwise, dynamically loading data from hard disks during runtime would be intolerably slow. Second, because loading all compressed data into the GPU memory has become infeasible, effective data compression can reduce per-frame communication overhead between the CPU and GPU. Furthermore, these goals should be achieved without compromising rendering quality or increasing per-frame data decompression cost, which is directly related to the frame rate we can achieve.

In this paper, we present effective clustering and compression schemes for precomputed radiance transfer matrices so that the aforementioned runtime data communication, decompression and interpolation can be performed efficiently and accurately. First, we have developed a data segmentation scheme to facilitate interpolation from nearest neighbors. Once all the meshes associated with the sampled poses have been consistently segmented in the surface domain, every group of corresponding surface segments are further evenly divided into small groups in the pose domain using spectral graph partitioning and a nonlinear measure that computes similarities in terms of both pose and radiance transfer matrices. Second, we have developed a revised clustered PCA algorithm for transfer matrices. It incrementally creates new PCA clusters for data associated with new pose configurations. By exploiting additional data redundancies among different sampled poses, our method can achieve a higher compression ratio with the same approximation error. Our incremental clustering can make the cost of per-frame data decompression and interpolation satisfy a prescribed upper bound.

In addition, we have designed an efficient runtime algorithm that takes advantage of the precomputed clustering and compression results while effectively distributing the workload among multiple rendering passes. As a result, high-quality real-time rendering with global shading effects is achieved.

## 5.2 Overview

Since a dynamic articulated object may have a number of linked parts moving simultaneously, interactively computing light transport, including glossy reflection, interreflection, shadowing and subsurface scattering, under environment illumination for such objects is extremely challenging. Therefore, state-of-the-art techniques can only dynamically generate soft shadows [27], all-frequency shadows [104], or ambient occlusion values [105]. There exist other limitations with these techniques. In fact, the algorithm in [104] can interactively generate shadows for one moving part only, and only diffuse shading has been demonstrated on original object surfaces in [27].




(a) Input (b) Consistent Segmentation (c) Mesh Segment Clustering (d) Revised Clustered PCA

Figure 5.2: A multistage pipeline for data segmentation and clustering in the joint spatio-pose space. (a) PRT data for all combinations of poses and vertices can be arranged into a large-scale matrix. (b) Consistent segmentation horizontally divides the matrix into multiple submatrices. (c) Mesh segment clustering reorganizes columns of each submatrix into multiple smaller clusters. (d) Revised clustered PCA projects each row of the clusters onto the basis vectors of the same PCA cluster to facilitate runtime interpolation. Vertices in the same PCA cluster are shown with the same pattern.

To achieve all global shading effects on glossy or translucent articulated objects, we take the approach that draws many samples in the pose subspaces for a particular object, precomputes radiance transfer for all of them, and interpolates the precomputed transfer matrices during runtime. Note that the high-level approach of the rendering stage in [106] is similar to ours. Nevertheless, they only devoloped methods for diffuse surfaces and small elasticity-based deformations while our articulated objects typically generate deformations at a much larger scale. In this paper, we adopt the basic PRT framework for glossy objects presented in [23, 24] which account for global shading effects caused by low-frequency environment lighting using a truncated spherical harmonic basis.

Since we need to sample poses and perform interpolation, we briefly discuss our pose sampling strategy here even though it is not the focus of this paper. We took multiple representative sequences from the CMU motion capture database [107]. Each sequence represents a distinct type of whole body motion, such as boxing, dancing, running and walking. A pose consists of all the joint angles in a specific skeletal configuration of the object, and there is a pose at every frame of these sequences. We ran clustered principal component analysis (CPCA) on all the poses and extracted a few pose subspaces. Since we would like to obtain pose samples that are distributed more uniformly and widely within each subspace than the input poses, we resample each subspace by performing stratified Monte Carlo sampling on the PCA coefficients. Let $\mathcal{C} = \{p_1, p_2, ..., p_{m_p}\}$ be the set of resampled poses. Note that we need to generate a deformed surface mesh for every sampled pose using skinning and then precompute radiance transfer matrices for these deformed meshes. All the deformed meshes have the same number of vertices and the same connectivity among the vertices. Only the vertex positions have been altered. Let $\Upsilon^j = \{v_1^j, v_2^j, ..., v_{n_v}^j\}$ be the complete set of vertices in the deformed mesh for a single pose $p_j \in \mathcal{C}$. We can arrange all the vertices within $\Upsilon^j$ into a single column vector, and arrange such column vectors for all poses into a matrix, $\mathcal{M}$. Each row of vertices within $\mathcal{M}$ are actually corresponding vertices on different meshes (Fig. 5.2(a)).

We perform a consistent mesh segmentation across all the deformed meshes using the technique in [7] to obtain a set of segments for every mesh. A consistent segmentation partitions different meshes into the same number of segments and corresponding vertices on different meshes belong to corresponding segments (Fig. 5.2(b)). Let $\{\Upsilon_k^j | 1 \leq k \leq n_r\}$ be the segmentation of $\Upsilon^j$ so that $\Upsilon^j = \bigcup_k \Upsilon_k^j$. Next, we group corresponding segments on different meshes together and define $\Omega_k = \{\Upsilon_k^j | 1 \leq j \leq M\}$ for $1 \leq k \leq n_r$. We can imagine that each $\Omega_k$ is actually a submatrix of $\mathcal{M}$, and occupies a subset of rows within $\mathcal{M}$. We further run spectral graph partitioning (Section 5.3) on each $\Omega_k$ independently to reorganize its columns and evenly distribute them into smaller clusters each of which typically has 3-8 segments. Let $\{\Omega_{k,g} | 1 \leq g \leq n_c\}$ be the clusters from dividing $\Omega_k$ so that $\Omega_k = \bigcup_g \Omega_{k,g}$. If we move the segments of each cluster together, we can imagine each cluster as a submatrix where each column consists of vertices from the same mesh segment and each row consists of corresponding vertices from different segments within the same cluster (Fig. 5.2(c)). The mesh segments within each cluster should exhibit two types of similarity. If $\Upsilon_k^{j_1}$ and $\Upsilon_k^{j_2}$ belong to the same cluster, their corresponding poses, $p_{j_1}$ and $p_{j_2}$, should be similar, and radiance transfer matrices at corresponding vertices of $\Upsilon_k^{j_1}$ and $\Upsilon_k^{j_2}$ should also be similar. After data segmentation, radiance transfer matrices over all sampled poses are finally compressed using a revised clustered PCA algorithm.

During runtime, given an input skeletal configuration as well as the associated surface mesh, which shares the same segmentation discussed above, we first use the skeleton to find the most similar pose among the samples. Suppose this most similar pose is $p_j$, which is then used to find the cluster $\Omega_{k,g_k^j}$ where each mesh segment, $\Upsilon_k^j (1 \leq k \leq n_r)$, belongs. Radiance transfer matrices at each row of corresponding vertices within each retrieved cluster of mesh segments are finally interpolated to generate estimated transfer matrices for the vertices on the input mesh. There are two major reasons to perform the aforementioned clustering on each group of segments $\Omega_k (1 \leq k \leq n_r)$. First, the resulting clusters can accelerate runtime nearest neighbor search. We directly consider the mesh segments in the same cluster as approximate nearest neighbors. Second, such clustering can accelerate runtime transfer matrix interpolation once transfer matrices at corresponding mesh vertices in the same cluster of mesh segments are compressed using the same set of PCA basis vectors (Section 5.4 and Fig. 5.2(d)).

## 5.3 Mesh Segment Clustering

In this section, we apply a spectral graph partitioning algorithm, called *normalized cut* [93], to dividing each $\Omega_k (1 \leq k \leq n_r)$ into smaller clusters of mesh segments. In the current context, clustering based on normalized cut has important advantages. First, unlike linear subspace methods, such as PCA and clustered PCA, normalized cut is based on local pairwise similarity. Therefore, it is better suited for interpolation based on nearest neighbors. Second, normalized cut can easily handle nonlinear similarity measures and can easily integrate multiple similarity measures together to achieve a tradeoff among them. Third, a similarity measure in normalized cut is typically defined as

a Gaussian, which is a widely used radial basis function. Such a nonlinear measure is consistent with pose-based skin deformation techniques that use radial basis functions [49]. The normalized cut algorithm has recently been successfully applied to the compression of motion capture sequences [108]. In the following section, we briefly introduce this algorithm first.

### 5.3.1 Normalized Cut Framework

Let $\mathcal{G} = (\mathcal{U}, \mathcal{E})$ be a weighted graph, where the set of nodes, $\mathcal{U} = \{u_1, u_2, ..., u_n\}$. An edge, $(u_i, u_j) \in \mathcal{E}$, has a weight $w(u_i, u_j)$ defined by the similarity between the location and attributes of the two nodes defining the edge. The idea is to partition the nodes into two subsets, $\mathcal{A}$ and $\mathcal{B}$, such that the following disassociation measure, the normalized cut, is minimized.

$$Ncut(\mathcal{A}, \mathcal{B}) = \frac{cut(\mathcal{A}, \mathcal{B})}{asso(\mathcal{A}, \mathcal{U})} + \frac{cut(\mathcal{A}, \mathcal{B})}{asso(\mathcal{B}, \mathcal{U})} \tag{5.1}$$

where $cut(\mathcal{A}, \mathcal{B}) = \sum_{s \in \mathcal{A}, t \in \mathcal{B}} w(s, t)$ is the total connection from nodes in $\mathcal{A}$ to nodes in $\mathcal{B}$; $asso(\mathcal{A}, \mathcal{U}) = \sum_{s \in \mathcal{A}, t \in \mathcal{U}} w(s, t)$ is the total connection from nodes in $\mathcal{A}$ to all nodes in the graph; and $asso(\mathcal{B}, \mathcal{U})$ is similarly defined. This measure works much better than $cut(\mathcal{A}, \mathcal{B})$ because it favors relatively balanced subregions instead of cutting small sets of isolated nodes in the graph.

To compute the optimal partition based on the above measure is NP-hard. However, it has been shown [93] that a good approximation can be obtained by relaxing the discrete version of the problem to a continuous one which can be solved using eigendecomposition techniques. Let $\mathbf{y}$ be the indicator vector of a partition. Each element of $\mathbf{y}$ takes two discrete values to indicate whether a particular node in the graph belongs to $\mathcal{A}$ or $\mathcal{B}$. If $\mathbf{y}$ is relaxed to take on continuous real values, it can be shown that the optimal solution can be obtained by solving the following generalized eigenvalue system,

$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y} \tag{5.2}$$

where $\mathbf{D}$ is a diagonal matrix with $\mathbf{D}(i, i) = \sum_j w(u_i, u_j)$, $W$ is the weight matrix with $\mathbf{W}(i, j) = w(u_i, u_j)$. The eigenvector corresponding to the second smallest eigenvalue is the optimal indicator vector in real space. A suboptimal partition can be obtained by first allowing $\mathbf{y}$ to take on continuous real values, solving the above generalized eigenvalue system for $\mathbf{y}$, and then searching for the best threshold to partition the real-valued elements of $\mathbf{y}$ into two subgroups. There exist different criteria to guide this threshold search. By default, one would like to use the cost function in (5.1) as the criterion so that the threshold can minimize this cost. Alternatively, as in our case, one may look for two evenly subdivided subgroups. Then, the threshold should be the median of the elements in $\mathbf{y}$. In either case, it is a

one-dimensional search that can be performed very quickly. The two resulting subregions from this partition can be recursively considered for further subdivision. This algorithm can be used to solve different clustering or segmentation problems by choosing different edge weights [109].

### 5.3.2 Mesh Segment Clustering

In the current context, we would like to partition every group of corresponding mesh segments, $\Omega_k (1 \leq k \leq n_r)$ into multiple smaller clusters. We set up a complete graph for a given group of segments. A node in the graph corresponds to a mesh segment from the group. The attributes of a node include the complete skeletal configuration corresponding to that node as well as the radiance transfer matrices at all the vertices within that segment.

The weight $w(u_i, u_j)$ over an edge $(u_i, u_j)$ is the product of two similarity terms. One measures the overall similarity between the transfer matrices associated with the two nodes of the edge. The other measures the similarity between the poses associated with the two nodes. Both similarity terms can be in the form of a Gaussian distribution. Overall, $w(u_i, u_j)$ is a local measure of how likely the nodes belong to the same partition. $w(u_i, u_j)$ is close to 1 for nodes which are likely to belong together, and close to 0 for nodes which are likely to be separated, as judged purely from local evidence available at the two nodes.

Overall similarity regarding both transfer matrices and pose configuration is formulated as

$$w(u_i, u_j) = \exp\left(-\frac{\sum_l \|\mathbf{T}_l^i - \mathbf{T}_l^j\|_F^2}{2\sigma_t^2} - \frac{\sum_{b \in S} \|\mathbf{R}_b^i - \mathbf{R}_b^j\|_F^2}{2\sigma_p^2}\right) \tag{5.3}$$

where $u_i$ and $u_j$ are two graph nodes each of which has a corresponding mesh segment, $\mathbf{T}_l^i$ denotes the transfer matrix at vertex $v_l$ in the mesh segment at node $u_i$, $S$ represents the complete skeleton of an articulated object, $b$ is a "bone" in this skeleton, $\mathbf{R}_b^i$ denotes the local rigid body transform at this "bone" in the pose corresponding to node $u_i$, and $\mathbf{T}_l^j$ and $\mathbf{R}_b^j$ are defined similarly. We use the Frobenius norm for both transfer matrices and local rigid body transforms. Parameters $\sigma_t$ and $\sigma_p$ are automatically determined from the respective standard deviations of the root mean squared differences of corresponding transfer matrices or local transforms within a pair of corresponding mesh segments. These parameters can be further adjusted to reflect the relative weighting between the two similarity measures.

Once the graph is set up, it is recursively partitioned into two subgraphs using the aforementioned normalized cut algorithm. Binary partition only needs one eigenvector with the second smallest eigenvalue with respect to (5.2). Since we always use the same number of neighbors to perform runtime interpolation and these neighbors are always from the same cluster, we enforce final clusters from graph subdivision have an equal size. This is achieved by enforcing the two subgraphs from every binary partition have an equal size. Recursive partition terminates when the size of the subgraphs reaches a given threshold, which is typically 4 in our experiments.

## 5.4 Transfer Matrix Compression

We need to precompute a radiance transfer matrix at every vertex of the mesh for every pose, which gives rise to an amount of data that is three orders of magnitude larger than that for a static object. However, there exist extra redundancies across different meshes that we can exploit by compressing transfer matrices from different meshes together. Nevertheless, there are two important requirements that need to be satisfied. First, the cost of per-frame data decompression should be bounded to guarantee runtime performance. Second, since runtime interpolation will be performed among transfer matrices at corresponding vertices within the same cluster of mesh segments, they should be compressed in such a way that facilitates such interpolation.

We have developed a revised clustered PCA algorithm with incremental cluster creation to overcome these difficulties. Note that incremental cluster creation is different from incremental singular value decomposition algorithms [110] that incrementally update the basis vectors of a cluster. Clustered PCA divides the original dataset into a few clusters each of which is approximated separately using a truncated PCA basis. In this context, the first requirement translates to an upper bound on the number of distinct PCA clusters used by per-frame data since all our PCA clusters have a fixed number of basis vectors. The second requirement is satisfied by using the basis vectors of the same PCA cluster to approximate all transfer matrices falling on the same row of the same cluster of mesh segments (Fig. 5.2(d)).

Our incremental algorithm goes through the set of poses in a sequential order and incrementally creates new PCA clusters as necessary for each additional pose while guaranteeing the total number of distinct PCA clusters used by that pose is below a prescribed upper bound, $m_{cpf}$. Suppose we are looking at pose $p_j$. Let $E = \{K_i | 1 \leq i \leq m_{pca}\}$ be the set of existing PCA clusters each of which has a list of transfer matrices that have been assigned to it. The mesh for $p_j$ has a set of vertices $\Upsilon^j$, which has been partitioned into segments, $\{\Upsilon^j_k | 1 \leq k \leq n_r\}$. Suppose $\Upsilon^j_k \in \Omega_{k,g^j_k}$, where $\Omega_{k,g^j_k}$ is a cluster of mesh segments, as defined in Section 5.2. Since vertices on other mesh segments in $\Omega_{k,g^j_k}$ might have been assigned PCA cluster memberships, according to the second requirement, corresponding vertices in $\Upsilon^j_k$ should have the same membership as well. Thus, we can divide $\{\Omega_{k,g^j_k} | 1 \leq k \leq n_r\}$ into two subgroups, $\{\Omega_{k,g^j_k} | k \in I_a\}$ and $\{\Omega_{k,g^j_k} | k \in I_b\}$, where vertices of the mesh segments in the first subgroup of clusters have been assigned to PCA clusters in $E$ but vertices associated with the second subgroup have not. The second subgroup of clusters are called active clusters. Let $E_j = \{K_i | i \in I_j\} \subseteq E$ be the set of PCA clusters already used by vertices from pose $p_j$ and $\rho^j_{assigned}$ be the percentage of the vertices from $p_j$ that have been assigned PCA clusters. When $1 - \rho^j_{assigned} \leq r\frac{m_{cpf} - |E_j|}{m_{cpf}}$, where $r$ is a ratio typically set to 2, we perform an incremental clustering step; otherwise, the number of new PCA clusters that can be created within the per-frame budget is too few compared to the number of unassigned vertices and, therefore, we need to perform a reclustering step to reinitialize a sufficient number of new PCA clusters better suited for the pose under consideration.

### 5.4.1 Incremental Clustering

We create $m_{cpf} - |E_j|$ new PCA clusters, and every vertex of the segments in the second subgroup is assigned to either one of the new clusters or one of the existing clusters in $E_j$. We have revised the original clustered PCA algorithm to achieve this goal. In the revised CPCA algorithm, we assign every row of corresponding vertices within the same cluster of segments to the same PCA cluster. According to the mesh clustering process presented in the previous section, the transfer matrices at such corresponding vertices should already have a high degree of similarity. Therefore, assigning them to the same PCA cluster would not sacrifice much accuracy, as confirmed by our experiments.

The criterion to determine cluster membership is the cumulative squared approximation errors contributed by all the vertices in the row. There are still two alternating steps in the revised CPCA algorithm. In the first step, every row of vertices in an active cluster of segments, $\Omega_{k,g_k^j}(k \in I_b)$, is assigned to the PCA cluster that produces minimal cumulative squared errors. In the second step, the basis vectors of the new PCA clusters are updated. Thus, in this revised algorithm, existing clusters in $E_j$ can accept new members, but their previously existing members cannot change their memberships. The basis vectors of these clusters are updated only when these clusters have accepted a significant number of new members. If we define a cost function as the summed squared approximation errors within all $m_{cpf}$ clusters, it is straightforward to show that this cost function monotonically decreases during each of the above two steps. Therefore, the revised CPCA algorithm converges.

### 5.4.2 Reclustering

In the reclustering step, to guarantee high-quality approximation within the per-frame budget, we simply choose to generate $m_{cpf}$ entirely new PCA clusters for the current pose by running the original CPCA algorithm. Before doing that, those rows of vertices associated with the first subgroup of clusters of mesh segments, i.e. $\{\Omega_{k,g_k^j} | k \in I_a\}$, need to be removed from the PCA clusters where they have been previously assigned.

### 5.4.3 Cluster Merging

Because reclustering may split among multiple new clusters vertices that previously belong to the same PCA cluster, it may cause the number of PCA clusters used by another pose to exceed our per-frame budget. Therefore, we perform a cluster merging step once our incremental algorithm has generated clusters for all the poses. As a preprocessing step, we first measure distance between pairwise PCA clusters. We only need to consider pairs of clusters that have members from the same pose, and only consider those poses that have used more clusters than they should. The distance between two PCA clusters can be measured by the $L^2$-norm of the difference between their corresponding projection matrices [111]:

$$d(\Psi, \Phi) = \|\mathbf{P}_\Psi - \mathbf{P}_\Phi\|_2, \tag{5.4}$$

where $\mathbf{P}_\Psi = \mathbf{U}_\Psi \mathbf{U}_\Psi^T$ is the projection matrix of cluster $\Psi$, $\mathbf{U}_\Psi$ is a matrix whose columns are the basis vectors of $\Psi$, and $\mathbf{P}_\Phi$ is defined similarly.

During each iteration of the merging process, a pair of PCA clusters with the minimal distance is chosen. This pair of clusters is merged only if at least one pose shared by their members has more clusters than the per-frame budget. Once merged, distances between the new cluster and all relevant existing clusters need to be computed. This process is repeated until the number of clusters used by all poses has fallen below the upper bound, $m_{cpf}$. There exist efficient and accurate algorithms for merging two clusters without going back to the raw data in these original clusters. We apply the merging algorithm in [112]. At the end of merging, the coefficient vector of every transfer matrix needs to be recomputed according to its final cluster membership.

## 5.5 Runtime Algorithm

The shading equation we follow is largely similar to the one in [24]. Although our implementation is in three color channels, in the following, we explain the shading process using a single channel. The final radiance from a surface point along a specific viewing direction $\mathbf{V}$ and under a specific low-frequency global lighting vector $\mathbf{L}$ can be formulated as

$$\Gamma = \mathbf{V}^T \mathbf{B}\mathbf{T}\mathbf{L}, \tag{5.5}$$

where $\mathbf{B}$ is a BRDF matrix, $\mathbf{T}$ is the radiance transfer matrix with an integrated rotation from the global frame to the local frame [23], and the lighting vector $\mathbf{L}$ is a coefficient vector computed by projecting an environment map onto SH bases. The BRDF matrix is obtained by first discretizing a continuous BRDF, $B(v, s)$, in both viewing and lighting directions to obtain an intermediate matrix whose rows correspond to different viewing directions and columns to different lighting directions. Then each row of the intermediate matrix is projected onto the spherical harmonic (SH) bases.

In this paper, we always use 25 spherical harmonic bases for both lighting and BRDF. Inspired by PCA-based separable approximations of an arbitrary BRDF [113], we further factorize the BRDF matrix using singular value decomposition (SVD) and represent it as a product of a view map, $\mathbf{H}$, and a light map, $\mathbf{G}$, $\mathbf{B} = \mathbf{H}\mathbf{G}^T$. There is an important distinction between our factorization and that in [114, 115]. We decompose the BRDF matrix after spherical harmonic projection while they directly decompose the original BRDF for all-frequency rendering. Since we have also found experimentally that a 4-term approximation can produce sufficiently accurate results, both $\mathbf{H}$ and $\mathbf{G}$ are chosen to have only four columns. With the resolution of the BRDF view map set to $32 \times 32$ and the number of SH bases being 25, $\mathbf{H}$ and $\mathbf{G}$ are represented as $1024 \times 4$ and $25 \times 4$ matrices, respectively.

Suppose during runtime we need to estimate the transfer matrix $\tilde{\mathbf{T}}_v$ at vertex $v$ for a new incoming pose whose

most similar sampled pose is $p_j$, which is found quickly using a Kd-tree. Let $v$ belongs to the $k$-th mesh segment, which further belongs to a cluster of mesh segments, $\Omega_{k,g_k^j}$. Then $\tilde{\mathbf{T}}_v$ can be interpolated from the transfer matrices defined at the corresponding vertices of $v$ on the mesh segments in $\Omega_{k,g_k^j}$ as follows.

$$\tilde{\mathbf{T}}_v = \sum_l \alpha_l \mathbf{T}_v^{j_l} = \sum_l \alpha_l \left( \sum_i c_i^{v,j_l} \mathbf{U}_i^l \right), \tag{5.6}$$

where $\mathbf{T}_v^{j_l}$ represents the transfer matrix at the corresponding vertex of $v$ on a mesh segment in $\Omega_{k,g_k^j}$, and $\alpha_l$ is its interpolation coefficient. The interpolation coefficients are implemented using normalized radial basis functions (NRBFs) [116], and need to be computed only once for each mesh segment. Since we still use clustered PCA to approximate transfer matrices, $\{\mathbf{U}_i^l | i = 1, ..., n_b\}$ represent the set of PCA basis matrices whose linear combination approximates $\mathbf{T}_v^{j_l}$, and $c_i^{v,j_l} (i = 1, ..., n_b)$ are the PCA coefficients. Because our revised CPCA algorithm assigns different $\mathbf{T}_v^{j_l}$'s from the same row of $\Omega_{k,g_k^j}$ to the same PCA cluster, we can drop the superscript in $\{\mathbf{U}_i^l | i = 1, ..., n_b\}$, and (5.6) becomes

$$\tilde{\mathbf{T}}_v = \sum_i \left( \sum_l \alpha_l c_i^{v,j_l} \right) \mathbf{U}_i = \sum_i \lambda_i \mathbf{U}_i, \tag{5.7}$$

where $\lambda_i = \sum_l \alpha_l c_i^{v,j_l}$. This means we can simply interpolate scalar PCA coefficients to avoid reconstructing multiple transfer matrices.

Substituting (5.7) into (5.5), we obtain

$$\Gamma = \mathbf{V}^T \mathbf{H} \mathbf{G}^T \left( \sum_i \lambda_i \mathbf{U}_i \right) \mathbf{L} \tag{5.8}$$

$$= \left(\mathbf{V}^T \mathbf{H}\right) \Big( \sum_i \lambda_i \big( \overbrace{\underbrace{(\mathbf{G}^T \mathbf{U}_i)}_{\mathbf{Q}_i} \mathbf{L}}^{\mathbf{Z}} \big) \Big), \tag{5.9}$$

where the parentheses in (5.9) indicate the order of evaluation we use, and $\mathbf{Q}_i$, $\mathbf{S}_i$ and $\mathbf{Z}$ are intermediate variables.

The computation of (5.9) is partitioned into multiple stages, including precomputing, a runtime CPU pass and two runtime GPU passes. We have implemented the GPU passes using DirectX 9 API.

• In the precomputing stage, we multiply the $4 \times 25$ matrix, $\mathbf{G}^T$, with each $25 \times 25$ matrix, $\mathbf{U}_i$, to obtain a $4 \times 25$ matrix, $\mathbf{Q}_i$. Such computation is performed for the 8 basis matrices we use for each PCA cluster. Since the size of $\mathbf{Q}_i$ is smaller than a basis matrix, this step actually helps improve our overall data compression ratios.

• During the runtime CPU pass, we select the $\mathbf{Q}_i$'s needed by the current frame and multiply each of them with the current $25 \times 1$ lighting vector, $\mathbf{L}$, to obtain a $4 \times 1$ vector, $\mathbf{S}_i$. The time complexity of this step is proportional to the number of PCA clusters needed by the current frame. The resulting vectors, $\mathbf{S}_i$'s, are saved as a 2D texture. In

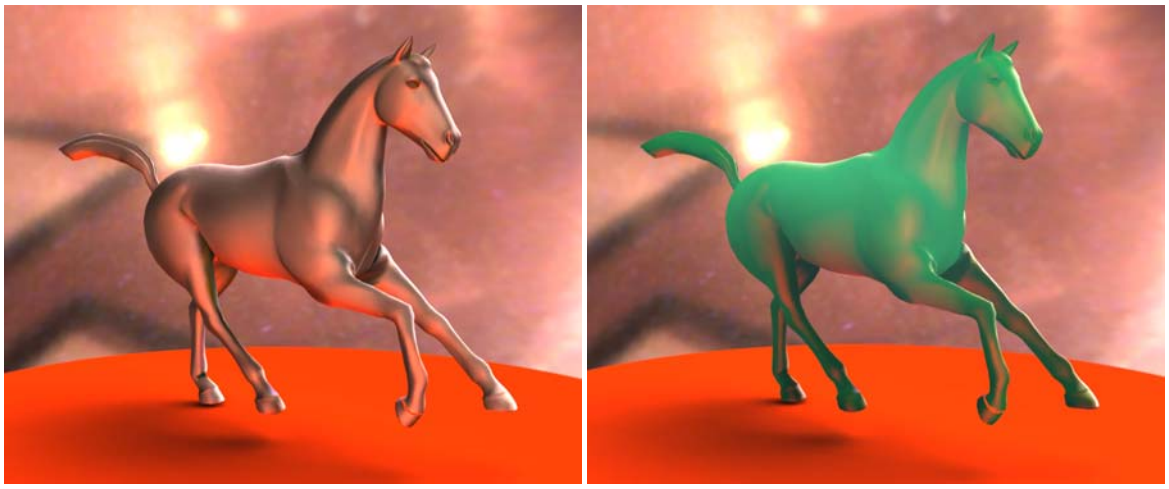


Figure 5.3: Renderings from our method. Left: a glossy deforming mesh. Right: a translucent deforming mesh. Both images exhibit global shading effects, including soft shadows, diffuse and specular interreflections. The right image also exhibits subsurface scattering.

addition, we compute $\lambda_i$'s by interpolating relevant PCA coefficients as in (5.7). This step needs to be done for every vertex, and is performed on the CPU because the available bandwidth between the system memory and GPU does not permit us to transmit the data before interpolation. To improve runtime performance, we have implemented this pass using double threads on a dual-core processor. The resulting coefficients are saved as textures as well. Since only a subset of the clusters are selected in each frame, all the computed textures during this pass become small enough to be transmitted to the GPU on a per-frame basis.

• In the first GPU pass, we compute $\mathbf{Z} = \sum_i \lambda_i \mathbf{S}_i$ for every vertex, where $\lambda_i$ and $\mathbf{S}_i$ are passed from the previous CPU pass. Although this is purely vertex-based processing, considering the performance limitation of vertex textures on DirectX 9 generation GPUs, we choose to implement it in a pixel shader program via a GPGPU technique by drawing a quad covering as many pixels as the total number of vertices. Every pixel in the quad computes its own $\mathbf{Z}$ vector using the textures representing $\mathbf{S}_i$'s and $\lambda_i$'s. These $\mathbf{Z}$ vectors from the pixel shader are saved back to the GPU video memory as a 2D texture map using multiple render targets.

• In the second GPU pass, we use a pixel shader program to first compute $\mathbf{V}^T\mathbf{H}$ for every pixel. Since we use bilinear interpolation in the viewing direction, $\mathbf{V}$ has four nonzero entries, and $\mathbf{V}^T\mathbf{H}$ is computed as a linear blend of four row vectors of $\mathbf{H}$. Meanwhile, the per-vertex $\mathbf{Z}$ vectors from the previous pass are linearly interpolated at every pixel in the GPU pipeline implicitly. At the end, we perform pixelwise multiplication between the row vector, $\mathbf{V}^T\mathbf{H}$, and the interpolated $\mathbf{Z}$ vector to obtain the final radiance for each color channel.

In addition to opaque surfaces, we also support translucent objects with subsurface scattering (Fig. 5.3). We follow the algorithms in [117] for precomputing single and multiple scattering. The precomputed results can still be represented using transfer matrices except that the BRDF matrix in (5.5) should be replaced with a matrix that

| | #vertices | #poses | # mesh segs | prt coefficients computation time | segment clustering | | revised CPCA | |
|---|---|---|---|---|---|---|---|---|
| | | | | | #clusters | cpu time | #clusters | cpu time |
| Armadillo | 33,000 | 1024 | 19 | 77hrs | 2432 | 42hrs | 36000 | 91hrs |
| Boxer | 32,000 | 1024 | 11 | 64hrs | 2816 | 38hrs | 96000 | 115hrs |
| Horse | 19,000 | 48 | 27 | 2hrs | 648 | 30mins | 2984 | 1.7hrs |

Table 5.1: Statistics for Precomputation. We utilize the cluster servers in our institution to accelerate the computation for PRT coefficients. In the Boxer example, every mesh segment cluster contains 4 segments. There are 72000 PCA clusters on the character model and additional 24000 clusters for a floor plane. In the Armadillo example, every mesh segment cluster contains 8 segments, which have the side effect that the total number of PCA clusters is much reduced. The small number of sampled poses for the Horse example gives rise to much smaller numbers of mesh segment clusters and PCA clusters, and an overall much smaller dataset. Every PCA cluster in these examples has eight $25 \times 25$ basis matrices.

| | Armadillo | Boxer | Horse |
|---|---|---|---|
| original data size | 253GB | 215GB | 3.06GB |
| compressed data size | 1.7GB | 1.54GB | 76MB |
| data used in demo | 200MB | 430MB | 76MB |
| frame rate | 30 | 30 | 45$\sim$50 |

Table 5.2: Compression Result and Performance. As shown, we have achieved a compression ratio of around 140 on large examples. Because we generated the poses with enough variations, the demo animations from our paper usually require only a small subset of the sampled poses for real-time interpolation. The Cook-Torrance BRDF model is used for Armadillo and the Phong model is used for both Boxer and Horse. All performance measurements were taken from a 3.0GHz PentiumD with nVidia Geforce 7900GTX 512MB VRAM.

accounts for light coming from both sides of the surface.

## 5.6 Experimental Results

We have successfully experimented with three examples. Renderings from our method can be found in Figs. 5.1, 5.3 and 5.4. For each example, we start with a static mesh and a few deformed versions of this mesh. A skinning model with blending weights is trained from these deformed versions using the technique in [7], which also produces a consistent segmentation of the meshes. Meanwhile, we obtain a number of MoCAP sequences from the CMU database [107], and align the skeleton used in the MoCAP sequences with the segmented meshes. As discussed in Section 5.2, we resample pose subspaces to obtain a database of sampled poses. Each of the sampled poses can generate a deformed version of the original mesh using the trained skinning model, and we precompute radiance transfer matrices for each of them using ray-tracing [118]. The statistics of these precomputed datasets are shown in Table 5.1. As we can see, two of the examples are very large. Each of them has more than 200GB of raw PRT data. Our data processing algorithms actually divide such large-scale raw data into smaller chunks and only load into the memory one chunk at a time. Once we have run the mesh segment clustering algorithm and the revised CPCA algorithm, we quantize each PCA coefficient down to 16 bits. In this way, we can achieve a compression ratio of around 140 without losing much

| | Our Scheme | Pose-Space |
|---|---|---|
| 1 | 8851 | 10661 |
| 2 | 5307 | 11015 |
| 3 | 6104 | 8167 |
| 4 | 8348 | 9686 |
| 5 | 9819 | 13922 |
| Avg | 7685.2 | 10690.2 |

Table 5.3: Comparisons of approximation errors between our interpolation scheme for transfer matrices and pose-space based interpolation. The first five rows show the errors for five randomly generated poses, and the last row shows the average errors among the five.

visual fidelity.

With such a precomputed and compressed dataset, we can render dynamically deformed versions of the original mesh in real time. We have experimented with two possible methods to generate dynamically deformed meshes. In the first method, the user can interactively adjust the pose of the skeleton, and every adjusted pose produces a new deformed mesh. In the second method, we can take a new MoCAP sequence with the same skeletal structure and use the poses in this sequence to deform the mesh. Note that these poses are different from all previously sampled poses. In both cases, the transfer matrices for the dynamically deformed mesh are interpolated from the precomputed data in real time and the deformed mesh is shaded instantly on the GPU using the interpolated transfer matrices. Since it is not very convenient to interactively produce interesting new poses without referencing MoCAP data, most of our demonstrations use the second method.

### 5.6.1 Validation

**Frame Rate**

Frame rate is by far the most important goal. An important reason that we can achieve real-time performance with at least 30 frames per second is that transfer matrix interpolation is actually performed through the interpolation of their scalar PCA coefficients, as formulated in (5.7). This is facilitated by both mesh segment clustering and the revised CPCA algorithm. We did a comparison on the two large examples between our technique and a different version that does not use the same set of PCA bases to approximate transfer matrices at corresponding vertices within the same cluster of mesh segments. The latter only achieved at most 8 frames per second, more than three times slower than our version.

**Interpolation Accuracy**

We interpolate transfer matrices for each mesh segment independently using the cluster of mesh segments it belongs. Our clusters of mesh segments are formed considering similarity in both transfer matrices and pose. In contrast, given

a new pose, a conventional scheme finds a few nearest sampled poses, and then simply interpolates all transfer matrices for the new pose from the same set of nearest sampled poses. It is thus a global scheme that always uses the same set of neighbors for different mesh segments.

We have compared our interpolation scheme with this pose-space scheme on our examples. During the precomputing stage of these comparisons, we use the same number of PCA clusters to compress the raw PRT data in both cases. During runtime, we randomly generate new poses that further produce new deformed meshes, which are rendered using the two interpolation schemes which choose different sets of neighbors. We also directly ray-trace the deformed meshes to produce the ground truth. Table 5.3 lists summed errors of the interpolated transfer matrices for a few randomly generated poses. The errors of our interpolation scheme are consistently lower. However, differences in numerical errors may not reflect the true differences in visual quality. We further compare the rendered images from these interpolation schemes in Figs. 5.4 and 5.5. The global scheme produces obvious artifacts on certain parts of the surface while the results from our scheme are visually comparable to the ground truth.

**Compression Quality**

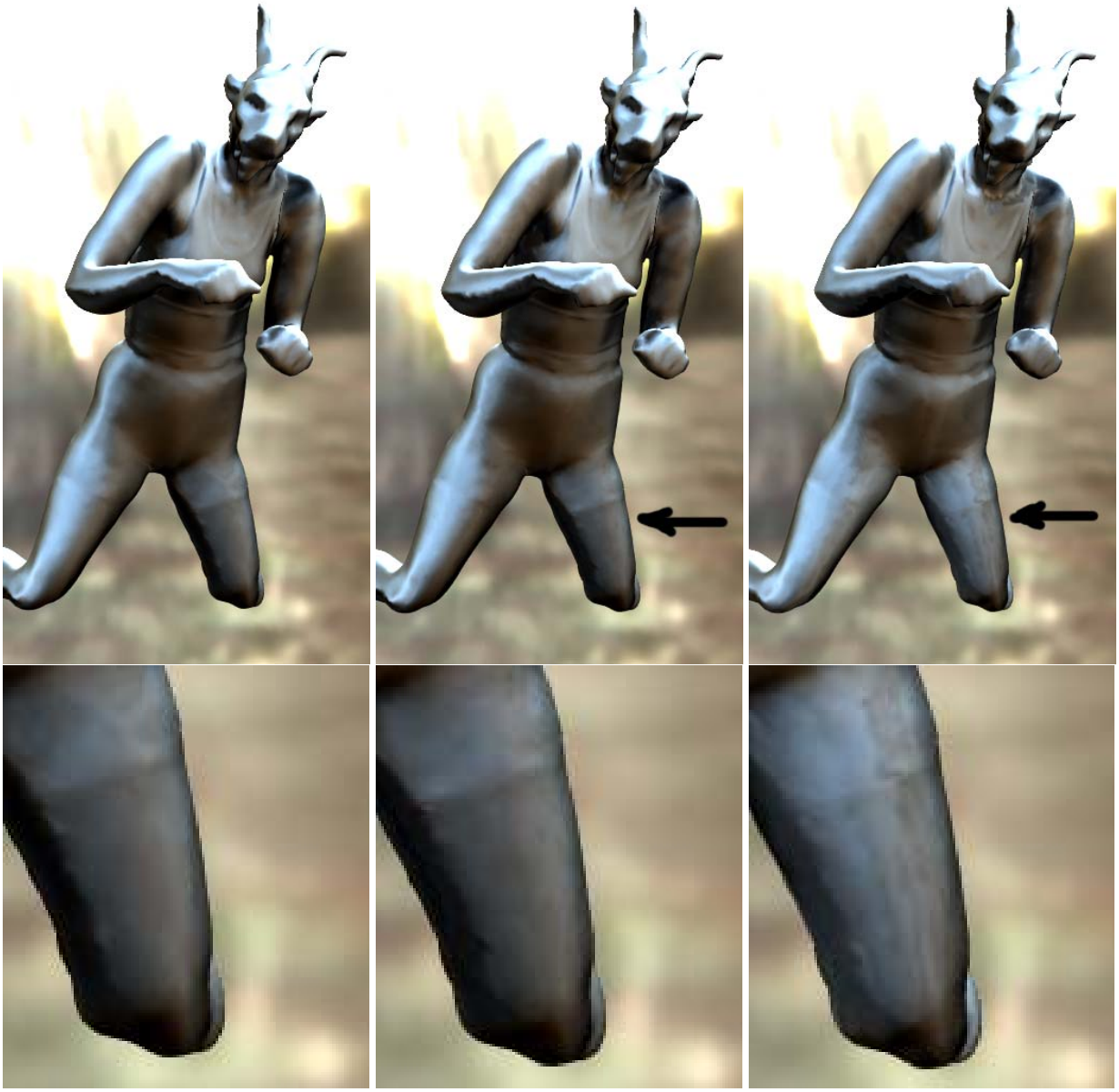
We further validate the compression quality of our revised CPCA algorithm by comparing it with running the original CPCA on each pose independently. Because an important goal of our algorithm is to satisfy the two requirements discussed in Section 5.4 instead of achieving maximum compression ratios, without incremental cluster creation, it should generate larger approximation errors than the original CPCA. However, incremental cluster creation exploits data redundancies across different poses and, therefore, can compensate the negative effects produced by the requirements we impose on compression. As a result, with the same total number of PCA clusters, our revised CPCA actually achieves smaller approximation errors than running CPCA independently on each pose. For example, on the BOXER dataset, when there are a total of 72000 clusters over 1024 poses, the average error per pose is 4489 for our algorithm and 5516 for independent CPCA. More importantly, the latter cannot satisfy the second requirement, therefore, would significantly lower the frame rate. As shown in Fig. 5.6, our revised algorithm also produces results with better visual quality and without obvious boundaries among PCA clusters.

## 5.7 Conclusions and Future Work

We have presented effective data clustering and compression techniques as well as an efficient runtime algorithm that can achieve high-quality real-time rendering of dynamically skinned models using precomputed radiance transfer. Our techniques can reduce the amount of precomputed data to a managable size, and achieve a compression ratio of 140 on large-scale datasets with hundreds of gigabytes of raw data. Meanwhile, they also facilitate runtime data communication, decompression and interpolation. Our algorithms and results have demonstrated that using an example-based

approach for PRT-based rendering of dynamic objects with glossy or translucent materials is both feasible and practical.

There are limitations with our current algorithms and implementation. First, we are limited to low-frequency environment lighting. We would like to investigate in future whether it is feasible to extend our work to all-frequency lighting using nonlinear wavelet approximation [119, 120]. Second, we would like to model interactions among multiple objects using subspaces and investigate precomputed radiance transfer for such interactions. Third, our implementation is partially limited by the memory capacity and streaming bandwidth of the current generation of GPUs. We expect these aspects improved in the future generations and even better runtime performance achieved on them.



Ground Truth | Our Method | Pose-Space

Figure 5.4: A comparison between our interpolation scheme and pose-space based transfer matrix interpolation. Since pose-space interpolation only considers similarity in global pose configurations without accounting for similarity among transfer matrices themselves, visually noticeable artifacts occur.

Ground Truth Our Method Pose-Space

Figure 5.5: Another comparison between our interpolation scheme and pose-space based transfer matrix interpolation in the self-occlusion area. Pose-space interpolation fails to capture some self-occluded effects and has more visual artifacts

Ground Truth Independent CPCA Our Revised CPCA

Figure 5.6: A comparison between our revised CPCA and running CPCA independently on different poses. Note that our revised CPCA satisfies additional requirements, and incrementally creates new clusters. With the same total number of PCA clusters (72000 clusters over 1024 poses in this example), our algorithm produces visually better results while independent CPCA produces visible boundary effects.

# References


[1] B. Purnomo, J.D. Cohen, and S. Kumar. Seamless texture atlases. In *Proceedings of the Eurographics/SIGGRAPH Symposium on Geometry Processing*, pages 65–74, 2004.

[2] Li Zhang, Noah Snavely, Brian Curless, and Steven M. Seitz. Spacetime faces: High-resolution capture for modeling and animation. *ACM Transactions on Graphics*, 23(3):548–558, 2004.

[3] Ladislav Kavan, Rachel McDonnell, Simon Dobbyn, Jiri Zara, and Carol O'Sullivan. Skinning arbitrary deformations. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 53–60, 2007.

[4] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2), 2007.

[5] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *ACM Transactions on Graphics*, 23(3):905–914, 2004.

[6] R.Y. Wang, K. Pulli, and J. Popović. Real-time enveloping with rotational regression. *ACM Transactions on Graphics*, 26(3):73.1–73.9, 2007.

[7] D.L. James and C.D. Twigg. Skinning mesh animations. *ACM Transactions on Graphics*, 24(3):399–407, 2005.

[8] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. MAPS: Multiresolution adaptive parameterization of surfaces. *Computer Graphics Proceedings (SIGGRAPH 98)*, pages 95–104, 1998.

[9] S.I. Park and J.K. Hodgins. Capturing and animating skin deformation in human motion. *ACM Transactions on Graphics*, 25(3):881–889, 2006.

[10] R.W. Sumner, M. Zwicker, C. Gotsman, and J. Popović. Mesh-based inverse kinematics. *ACM Transactions on Graphics*, 24(3):488–495, 2005.

[11] K.G. Der, R.W. Sumner, and J. Popović. Inverse kinematics for reduced deformable models. *ACM Transactions on Graphics*, 25(3):1174–1179, 2006.

[12] J.C. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In *Proceedings of the 7th Conference on Visualization*, pages 327–334, 1996.

[13] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 1997*, pages 199–208, 1997.

[14] H. Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH 1997*, pages 189–198, 1997.

[15] X. Gu, S.J. Gortler, and H. Hoppe. Geometry images. *ACM Transactions on Graphics*, 21(3):355–361, 2002.

[16] F. Losasso and H. Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Transactions on Graphics*, 23(3):769–776, 2004.

[17] J. Ji, E. Wu, S. Li, and X. Liu. Dynamic LOD on GPU. In *Proceedings of the Computer Graphics International*, pages 108–114, 2005.

[18] K. Niski, B. Purnomo, and J. Cohen. Multi-grained level of detail using a hierarchical seamless texture atlas. In *Symposium on Interactive 3D Graphics and Games*, pages 153–160, 2007.

[19] M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 1997*, pages 209–216, 1997.

[20] A. Mohr and M. Gleicher. Building efficient, accurate character skins from examples. *ACM TOG*, 22(3):562–568, 2003.

[21] C. DeCoro and S. Rusinkiewicz. Pose-independent simplification of articulated meshes. In *Symposium on Interactive 3D Graphics*, pages 17–24, 2005.

[22] S. Kircher and M. Garland. Progressive multiresolution meshes for deforming surfaces. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 191–200, 2005.

[23] P.-P. Sloan, J. Kautz, and J. Snyder. Precompted radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM TOG*, 21(3):527–536, 2002.

[24] P.-P. Sloan, J. Hall, J. Hart, and J. Snyder. Clustered principal components for precomputed radiance transfer. *ACM TOG*, 22(3):382–391, 2003.

[25] P.-P. Sloan, B. Luna, and J. Snyder. Local, deformable precomputed radiance transfer. *ACM TOG*, 24(3):1216–1224, 2005.

[26] K. Zhou, Y. Hu, S. Lin, B. Guo, and H. Shum. Precomputed shadow fields for dynamic scenes. *ACM TOG*, 24(3):1196–1201, 2005.

[27] Z. Ren, R. Wang, J. Snyder, K. Zhou, X. Liu, B. Sun, P.-P. Sloan H. Bao, Q. Peng, and B. Guo. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM TOG*, 25(3):977–986, 2006.

[28] A.W. Kristensen, T. Akenine-Möller, and H.W. Jensen. Precomputed local radiance transfer for real-time lighting design. *ACM TOG*, 24(3):1208–1215, 2005.

[29] M. Hasan, F. Pellacini, and K. Bala. Direct-to-indirect transfer for cinematic relighting. *ACM TOG*, 25(3):1089–1097, 2006.

[30] H. Hotelling. Relations between two sets of variates. *Biometrika*, 28:321–377, 1936.

[31] Thomas Melzer, Michael Reitera, and Horst Bischofb. Appearance models based on kernel canonical correlation analysis. *Pattern Recognition*, 36(9):1961–1971, 2003.

[32] Ladislav Kavan, Steven Collins, Jiri Zara, and Carol O'Sullivan. Skinning with dual quaternions. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 39–46, 2007.

[33] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: shape completion and animation of people. *ACM Transactions on Graphics*, 24(3):408–416, 2005.

[34] X. Shi, K. Zhou, Y. Tong, M. Desbrun, H. Bao, and B. Guo. Mesh puppetry: Cascading optimization of mesh deformation with inverse kinematics. *ACM Transactions on Graphics*, 26(3):81.1–81.10, 2007.

[35] O. Weber, O. Sorkine, Y. Lipman, and C. Gotsman. Context-aware skeletal shape deformation. *Computer Graphics Forum (Eurographics 2007)*, 26(3):265–274, 2007.

[36] P. Joshi, W.C. Tien, M. Desbrun, and F. Pighin. Learning controls for blend shape based realistic facial animation. In *Proceedings of the 2003 Eurographics/SIGGRAPH symposium on computer animation*, pages 162–174, 2003.

[37] Zhigang Deng, Pei-Ying Chiang, Pamela Fox, and Ulrich Neumann. Animating blendshape faces by cross-mapping motion capture data. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 43–48, 2006.

[38] M. Lau, J. Chai, Y.-Q. Xu, and H.-Y. Shum. Face poser: Interactive modeling of 3d facial expressions using model priors. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA 2007)*, pages 161–170, August 2007.

[39] Ryan White, Keenan Crane, and David Forsyth. Capturing and animating occluded cloth. *ACM Transactions on Graphics*, 26(3):34.1–34.8, 2007.

[40] Scott Kircher and Michael Garland. Editing arbitrarily deforming surface animations. *ACM Transactions on Graphics*, 25(3):1098–1107, 2006.

[41] W. Xu, K. Zhou, Y. Yu, Q. Tan, Q. Peng, and B. Guo. Gradient domain editing of deforming mesh sequences. *ACM Transactions on Graphics*, 26(3):84.1–84.10, 2007.

[42] S. Kircher and M. Garland. Free-form motion processing. *ACM Transactions on Graphics*, To Appear.

[43] K. Grochow, S.L. Martin, A. Hertzmann, and Z. Popivic. Style-based inverse kinematics. *ACM TOG*, 23(3):520–529, 2004.

[44] J. Chai and J.K. Hodgins. Performance animation from low-dimensional control signals. *ACM TOG*, 24(3):686–696, 2005.

[45] M. Dontcheva, G. Yngve, and Z. Popivic. Layered acting for character animation. *ACM TOG*, 22(3):409–416, 2003.

[46] Mark Meyer and John Anderson. Key point subspace acceleration and soft caching. *ACM Transactions on Graphics*, 26(3):74.1–74.8, 2007.

[47] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics (special issue for SIGGRAPH 2004)*, 23(3):641–648, 2004.

[48] nVidia CUDA. Compute unified device architecture (cuda). http://developer.nvidia.com/object/cuda.html.

[49] J.P. Lewis, M. Cordner, and N. Fong. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Computer Graphics Proceedings, Annual Conference Series*, 2000.

[50] B. Allen, B. Curless, and Z. Popović. Articulated body deformation from range scan data. *ACM Transactions on Graphics*, 21(3):612–619, 2002.

[51] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis. Scape: Shape completion and animation of people. *ACM TOG*, 24(3):408–416, 2005.

[52] Bernd Bickel, Mario Botsch, Roland Angst, Wojciech Matusik, Miguel Otaduy, Hanspeter Pfister, and Markus Gross. Multi-scale capture of facial geometry and motion. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 33, 2007.

[53] Wan-Chun Ma, Andrew Jones, Jen-Yuan Chiang, Tim Hawkins, Sune Frederiksen, Pieter Peers, Marko Vukovic, Ming Ouhyoung, and Paul Debevec. Facial performance synthesis using deformation-driven polynomial displacement maps. *ACM Trans. Graph.*, 27(5), 2008.

[54] Wei-Wen Feng, Byung-Uck Kim, and Yizhou Yu. Real-time data driven deformation using kernel canonical correlation analysis. *ACM Transactions on Graphics*, 27(3):91:1–9, 2008.

[55] Jernej Barbič and Doug L. James. Real-time subspace integration for st. venant-kirchhoff deformable models. *ACM Trans. Graph.*, 24(3), 2005.

[56] Sang Il Park and Jessica K. Hodgins. Data-driven modeling of skin and muscle deformation. *ACM Trans. Graph.*, 27(3):96:1–6, 2008.

[57] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proc. of SIGGRAPH'98*, pages 43–54, 1998.

[58] Kwang-Jin Choi and Hyeong-Seok Ko. Stable but responsive cloth. *ACM Trans. Graphics*, 21(3), 2002.

[59] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2003.

[60] Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. Efficient simulation of inextensible cloth. *ACM Trans. Graph.*, 26(3):49, 2007.

[61] Arnulph Fuhrmann, Clemens Gro$\beta$, and Volker Luckas. Interactive animation of cloth including self collision detection. In *WSCG '03*, 2003.

[62] Konstantinos Dinos Tsiknis. Better cloth through unbiased strain limiting and physics-aware subdivision. Master's thesis, University of British Columbia, 2004.

[63] M. Oshita and A. Makinouchi. Real-time cloth simulation with sparse particles and curved faces. In *Proc. of Computer Animation*, pages 62–83, 2001.

[64] Frederic Cordier and Nadia Magnenat-Thalmann. Real-time animation of dressed virtual humans. *Computer Graphics Forum*, 21(3):862–870, 2002.

[65] Frederic Cordier and Nadia Magnenat-Thalmann. A data-driven approach for real-time clothes simulation. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, 2004.

[66] Thomas Stumpp, Jonas Spillmann, Markus Becker, and Matthias Teschner. A geometric deformation model for stable cloth simulation. In *Workshop on Virtual Reality Interaction and Physical Simulation*, 2008.

[67] Olaf Etzmu$\beta$, Michael Keckeisen, and Wolfgang Stra$\beta$er. A fast finite element solution for cloth modelling. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, 2003.

[68] J. Rodriguez-Navarro and A. Susin. Non structured meshes for cloth gpu simulation using fem. In *Workshop on Virtual Reality Interaction and Physical Simulation*, 2006.

[69] R.W. Sumner and J. Popović. Deformation transfer for triangle meshes. *ACM Transactions on Graphics*, 23(3):397–403, 2004.

[70] M. Gleicher. Retargetting motion to new characters. In *SIGGRAPH 98 Proceedings*, pages 33–42, 1996.

[71] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Transactions on Graphics*, 24(3), 2005.

[72] S. Kircher and M. Garland. Free-form motion processing. *ACM Transactions on Graphics*, 27(2):1–13, 2008.

[73] P.G. Kry, D.L. James, and D.K. Pai. Eigenskin: real time large deformation character skinning in hardware. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA 2002)*, pages 153–159, 2002.

[74] Cyril Zeller. Cloth simulation on the gpu. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, page 39, New York, NY, USA, 2005. ACM.

[75] P.V. Sander, Z.J. Wood, S.J. Gortler, J. Snyder, and H. Hoppe. Multi-chart geometry images. In *Proceedings of the Eurographics/SIGGRAPH Symposium on Geometry Processing*, pages 146–155, 2003.

[76] N. Carr, J. Hoberock, K. Crane, and J. Hart. Rectangular multi-chart geometry images. In *Proceedings of the Eurographics/SIGGRAPH Symposium on Geometry Processing*, pages 181–190, 2006.

[77] Chih-Yuan Yao and Tong-Yee Lee. Adaptive geometry image. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):948–960, 2008.

[78] P.V. Sander, J. Snyder, S.J. Gortler, and H. Hoppe. Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 409–416, 2001.

[79] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, 21(3):362–371, 2002.

[80] R. Liu and H. Zhang. Segmentation of 3d meshes through spectral clustering. In *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications*, pages 298– 305, 2004.

[81] K. Zhou, J. Snyder, B. Guo, and H.-Y. Shum. Iso-charts: Stretch-driven mesh parameterization using spectral analysis. In *Proceedings of the Eurographics/SIGGRAPH Symposium on Geometry Processing*, pages 45 – 54, 2004.

[82] D. Julius, V. Kraevoy, and A. Sheffer. D-charts: Quasi-developable mesh segmentation. *Computer Graphics Forum*, 24(3):981–990, 2005. Proceedings of Eurographics.

[83] H. Yamauchi, S. Gumhold, R. Zayer, and H.-P. Seidel. Mesh segmentaion driven by Gaussian curvature. *Visual Computer*, 21:659–668, 2005.

[84] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Computer Graphics Proceedings (SIGGRAPH 95)*, pages 173–182, 1995.

[85] Cem Yuksel, John Keyser, and Donald H. House. Mesh colors. Technical Report tamu-cs-tr-2008-4-1, Dept. of CS, Texas A&M, 2008.

[86] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H.-P. Seidel. Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design*, 22(5):444–465, 2005.

[87] E. Zhang, K. Mischaikow, and G. Turk. Feature-based surface parameterization and texture mapping. *ACM Transaction on Graphics*, 24(1):1–27, 2005.

[88] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the Nyström method. *IEEE Trans. Pat. Anal. Mach. Intell.*, 26(2):214–225, 2004.

[89] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics*, 22(3):954–961, 2003.

[90] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Transactions on Graphics*, 23(3):796–803, 2004.

[91] L. Borgeat, G. Godin, F. Blais, P. Massicotte, and C. Lahanier. GoLD: interactive display of huge colored and textured models. *ACM Transactions on Graphics*, 24(3):869–877, 2005.

[92] L.M. Hwa, M.A. Duchaineau, and K.I. Joy. Real-time optimal adaptation for planetary geometry and texture: 4-8 tile hierarchies. *IEEE Trans. Vis. Comput. Graph*, 11(4):355–368, 2005.

[93] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pat. Anal. Mach. Intell.*, 22(8):888–905, 2000.

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

[95] Georgios Stylianou and Gerald Farin. Crest lines for surface segmentation and flattening. *IEEE Transactions on Visualization and Computer Graphics*, 10(5):536–544, 2004.

[96] V. Kraevoy, A. Sheffer, and C. Gotsman. Matchmaker: Constructing constrained texture maps. *ACM Transactions on Graphics*, 22(3):326–333, 2003.

[97] V. Kraevoy and A. Sheffer. Cross-parameterization and compatible remeshing of 3d models. *ACM Transactions on Graphics*, 23(3):861–869, 2004.

[98] J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe. Inter-surface mapping. *ACM Transactions on Graphics*, 23(3):870–877, 2004.

[99] Michael S. Floater. Mean value coordinates. *Comput. Aided Geom. Des.*, 20(1):19–27, 2003.

[100] M. Soucy, G. Godin, and M. Rioux. A texture-mapping approach for the compression of colored 3d triangulations. *The Visual Computer*, 12:503–514, 1996.

[101] N.A. Carr and J.C. Hart. Meshed atlases for real-time procedural solid texturing. *ACM Transactions on Graphics*, 21(2):106–131, 2002.

[102] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[103] A. Safonova, J.K. Hodgins, and N.S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM TOG*, 23(3):512–519, 2004.

[104] W. Sun and A. Mukherjee. Generalized wavelet product integral for rendering dynamic glossy objects. *ACM TOG*, 25(3):955–966, 2006.

[105] A.G. Kirk and O. Arikan. Precomputed ambient occlusion for character skins. SIGGRAPH 2006 Sketches, 2006.

[106] D.L. James and K. Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM TOG*, 22(3):879–887, 2003.

[107] Lib. MoCAP. CMU graphics lab motion capture database. http://mocap.cs.cmu.edu.

[108] O. Arikan. Compression of motion capture databases. *ACM TOG*, 25(3):890–897, 2006.

[109] J. Malik, S. Belongie, T. Leung, and J. Shi. Contour and texture analysis for image segmentation. *Int'l Journal of Computer Vision*, 43(1):7–27, 2001.

[110] M. Brand. Incremental singular value decomposition of uncertain data with missing values. In *Proc. European Conference on Computer Vision (Vol. I)*, pages 707–720, 2002.

[111] G.H. Golub and C.F. Van Loan. *Matrix Computations*. The Johns Hopkins Unversity Press, Baltimore, third edition, 1996.

[112] P. Hall, D. Marshall, and R. Martin. Merging and splitting eigenspace models. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(9):1042–1049, 2000.

[113] J. Kautz and M. McCool. Interactive rendering with arbitrary brdfs using separable approximations. In *Eurographics Workshop on Rendering*, pages 281–292, 1999.

[114] X. Liu, P.-P. Sloan, H.-Y. Shum, and J. Snyder. All-frequency precomputed radiance transfer for glossy objects. In *Eurographics Symposium on Rendering*, pages 337–344, 2004.

[115] R. Wang, J. Tran, and D.P. Luebke. All-frequency relighting of non-diffuse objects using separable brdf approximation. In *Eurographics Symposium on Rendering*, pages 345–354, 2004.

[116] O. Nelles. *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*. Springer Verlag, 2000.

[117] R. Wang, J. Tran, and D. Luebke. All-frequency interactive relighting of translucent objects with single and multiple scattering. *ACM TOG*, 24(3):1202–1207, 2005.

[118] M. Pharr and G. Humphreys. *Physically Based Rendering*. Morgan Kaufmann, 2004.

[119] R. Ng, R. Ramamoorthi, and P. Hanrahan. All-frequency shadows using non-linear wavelet lighting approximation. *ACM TOG*, 22(3):376–381, 2003.

[120] R. Ng, R. Ramamoorthi, and P. Hanrahan. Triple product integrals for all-frequency relighting. *ACM TOG*, 23(3):477–487, 2004.

# Vita

Wei-Wen Feng was born on July 18, 1980 in Taipei, Taiwan. He attended National Chiao Tung University in Hsin-Chu, Taiwan and earned a Bachelor of Science degree in Computer Science in 2002. He then served the mandatory military service in the Army Special Force Command. During the service, he successfully completed the paratrooper training and survived five jumps from the airplane ( with parachute ). After the military service, he left Taiwan to pursue the graduate study at University of Illinois at Urbana-Champaign. Yizhou Yu was Wei-Wen's Ph.D advisor at UIUC and has guided him through many challenges during his Ph.D study. Wei-Wen has completed various research projects in computer graphics, which have resulted in publications in conferences such as SIGGRAPH. His research interests include mesh deformation, cloth animation, and real-time rendering.