# Feature-Preserving Triangular Geometry Images for Level-of-Detail Representation of Static and Skinned Meshes

WEI-WEN FENG, BYUNG-UCK KIM and YIZHOU YU
University of Illinois at Urbana-Champaign
LIANG PENG
Intel Inc.
and
JOHN HART
University of Illinois at Urbana-Champaign

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 article 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 tenfold improvement in fidelity when compared to quad-chart geometry images.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Boundary representations*; I.3.7 [**Computer Graphics**]: Three Dimensional Graphics and Realism—*Animation*

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: Curvilinear features, spectral clustering, mesh simplification, mesh skinning

**ACM Reference Format:**

Feng, W.-W., Kim, B.-U., Yu, Y., Peng, L., and Hart, J. 2010. Feature-preserving triangular geometry images for level-of-detail representation of static and skinned meshes. ACM Trans. Graph. 29, 2, Article 11 (March 2010), 13 pages.
DOI = 10.1145/ 1731047.1731049 http://doi.acm.org/10.1145/1731047.1731049

## 1. INTRODUCTION

Traditional view-dependent LOD representations based on mesh simplification [Xia and Varshney 1996; Luebke and Erikson 1997; Hoppe 1997] rely on random-access mesh traversal with poor cache coherence. Geometry images [Gu et al. 2002] support efficient LOD display [Losasso and Hoppe 2004; Ji et al. 2005; Niski et al. 2007] 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 [Sander et al. 2003] improve this distortion with feature-sensitive clustering, but its irregular chart boundaries complicated coarser levels of LOD downsampling. Rectangle-chart geometry images [Purnomo et al. 2004; Carr et al. 2006; Yao and Lee 2008] 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 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

Authors' addresses: W.-W. Feng (corresponding author), B.-U. Kim, Y. Yu, Department of Computer Science, University of Illinois, 201 North Goodwin Avenue, Urbana, IL 61801; email: wfeng2@uiuc.edu; L. Peng, Intel Inc., 2200 Mission College Blvd., Santa Clara, CA 95054-1549; J. Hart, Department of Computer Science, University of Illinois, 201 North Goodwin Avenue, Urbana, IL 61801.

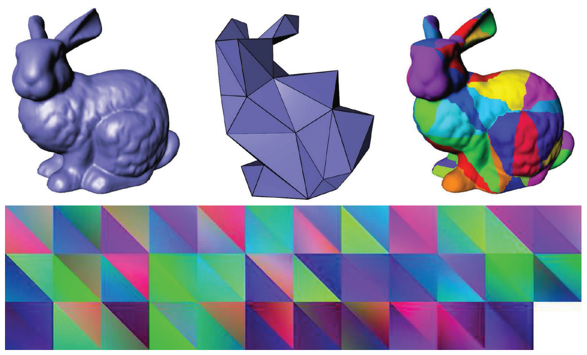

Fig. 1. Our method builds triangle-chart geometry images for feature-preserving LOD representation of both static and skinned meshes. It obtains a base complex and triangular charts from an original mesh using mesh simplification. The packed triangular geometry images are shown in the bottom.

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 [Wang et al. 2007] or SMA (Skinning Mesh Animations) [James and Twigg 2005]. 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, the contributions of this article include: (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 multiresolution 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.

## 2. RELATED WORK

The original geometry-image approach [Gu et al. 2002] 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. Multichart geometry images with irregular [Sander et al. 2003] or later rectangular [Purnomo et al. 2004; Carr et al. 2006] boundaries reduced distortion by decomposing the input mesh into multiple pieces, each individually parameterized and resampled. (Alternatively, Yao and Lee [2008] 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 [Sander et al. 2001; Lévy et al. 2002; Liu and Zhang 2004; Zhou et al. 2004; Julius et al. 2005; Yamauchi et al. 2005], but they produce either rectangular or irregularly shaped charts. Mutiresolution analysis on triangle mesh [Eck et al. 1995] 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 [Lee et al. 1998] 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 5.

Mesh Colors [Yuksel et al. 2008] uses barycentric coordinates to regularly sample textures over every triangle in an input mesh, packing the texture signal into a 1D 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 [Lévy et al. 2002; Lee et al. 2005; Zhang et al. 2005]. 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 [Fowlkes et al. 2004; Liu and Zhang 2004]. We improve this approach with new metrics that better detect crest lines and sharp points by processing the dual mesh in Section 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 [Katz and Tal 2003].

Clustering and simplification for deforming meshes has been less explored. The quadric error metric [Garland and Heckbert 1997] can be extended [Mohr and Gleicher 2003; DeCoro and Rusinkiewicz 2005] for simplification of a mesh with multiple deformed poses into a pose-independent simplified mesh. Hierarchical face clustering has been performed in Kircher and Garland [2005] to achieve pose-dependent simplification with higher visual quality for precomputed mesh deformation sequences. Hierarchical clustering [Wang et al. 2007] and a mean shift algorithm [James and Twigg 2005] can both yield pose-independent face clusters for deforming meshes, but Section 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 article 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 [Cignoni et al. 2004;

Borgeat et al. 2005; Hwa et al. 2005]. 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, for example, Borgeat et al. [2005]. Third, it combines geometry and texture into a single multiresolution representation, avoiding the wasted storage of texture coordinates for every chart in video memory.

## 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 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 [Fowlkes et al. 2004; Liu and Zhang 2004] using a similarity matrix based on curvature or deformation gradients. The boundaries 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 constraints in the simplification process. During mesh simplification, we apply MAPS [Lee et al. 1998] 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.

*Runtime LOD Rendering.* At runtime, 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.

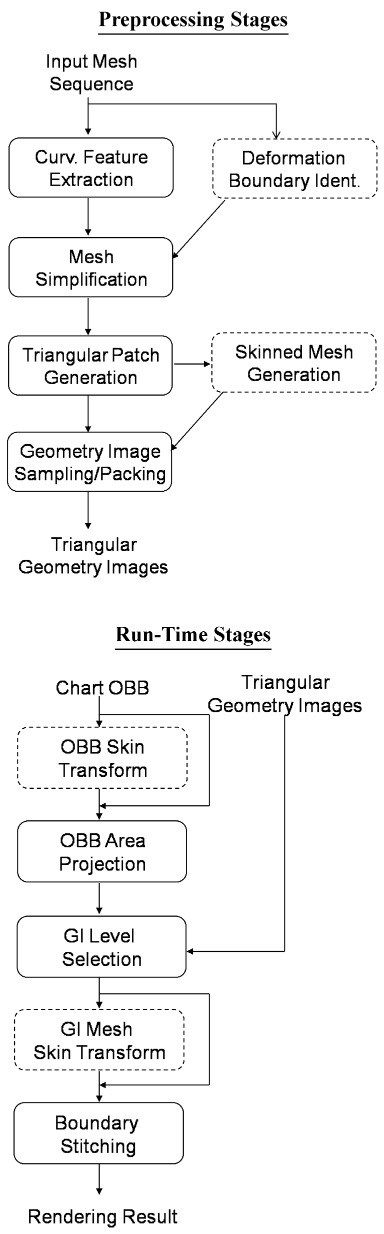

Fig. 2. The pipelines of our method in the preprocessing and runtime stages.

*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 connect $v_i$ and $v_j$. Given a skinned mesh with $n_b$ bones and $n_a$ frames of animation, we denote the set of bone transformations at frame $k$ as $\mathbf{T}_b^k$ 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. 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.

### 4.1 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 cut, is minimized:

$$Ncut(\mathcal{A}, \mathcal{B}) = \frac{cut(\mathcal{A}, \mathcal{B})}{cut(\mathcal{A}, \mathcal{U})} + \frac{cut(\mathcal{A}, \mathcal{B})}{cut(\mathcal{B}, \mathcal{U})}, \tag{1}$$

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 preceding measure is NP-hard. However, it has been shown [Shi and Malik 2000] 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

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

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 [Fowlkes et al. 2004]. 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 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 Fowlkes et al. [2004] to process large datasets with sparse sampling.

### 4.2 High Curvature Feature Extraction

We extract both feature lines and feature points using the process diagrammed in Figure 3.

*Curvilinear feature detection.* For static meshes, we measure features with local curvature estimates [Kalogerakis et al. 2007]. A set of fragmented crestline segments are then obtained by finding local extrema of curvatures [Stylianou and Farin 2004]. 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

$$w(i, j) = \beta \exp\left(-\frac{|\kappa_i| + |\kappa_j|}{\sigma_\kappa}\right) \exp\left(-\frac{dist(f_i, f_j)}{\sigma_d}\right), \tag{3}$$

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 nonadjacent faces according to their pairwise distances $dist(f_i, f_j)$ defined before. In our implementation, we add an additional dual edge between $f_i, f_j$ if $dist(f_i, f_j) < 2\bar{dist}$, where $\bar{dist}$ is the average distance between two adjacent faces. 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) [Cohen-Steiner et al. 2004], 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 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 article because we would like to detect salient curvilinear features.

Once we have spectral clustering results, as shown in Figure 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 [Katz and Tal 2003] 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

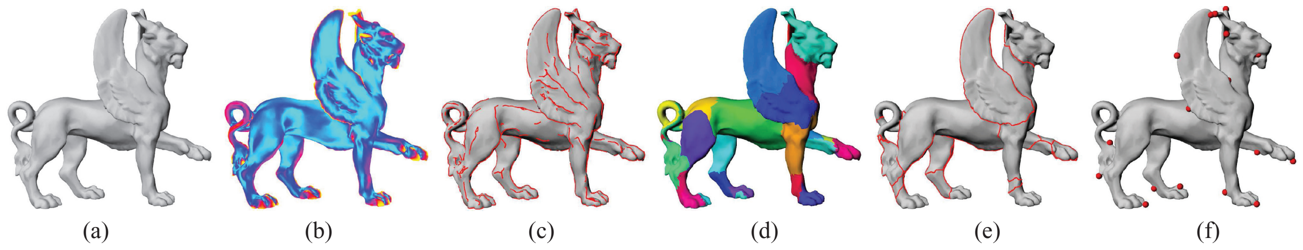

Fig. 3. The overall process of our feature extraction method. Given a mesh in (a), we first robustly estimate its per-vertex curvatures, as shown in (b). The initial crest lines in (c) are noisy and disconnected. Spectral clustering is applied on the mesh based on curvature similarity to extract a set 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).

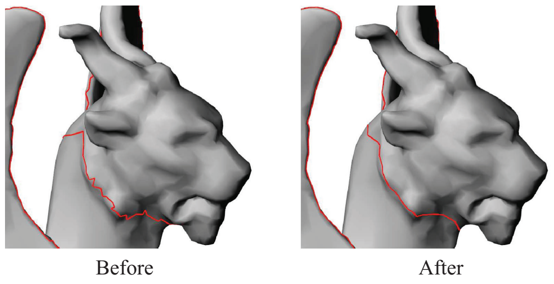

Fig. 4. The original feature lines from spectral clustering appear jaggy and may not align well with real features on the mesh. After applying the graph-cut algorithm, the refined feature lines become smoother and better localized.

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

$$g(i, j) = \beta |e_{ij}| \exp\left(-\frac{|\kappa_i| + |\kappa_j|}{\sigma_\kappa}\right), \tag{4}$$

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 (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.

After refinement, we break cluster boundaries into nonbranching 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

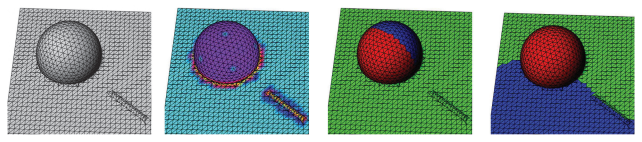

Fig. 5. Illustration of the difference between our spectral clustering and Variational Shape Approximation (VSA) [Cohen-Steiner et al. 2004]. The testing model has a high curvature feature, a narrow ridge, on the plane. VSA chooses to better approximate the overall shape by dividing the hemisphere into two clusters. Our method is better at feature detection and chooses to align a cluster boundary with the ridge.

preserving sharp corners such as horns or finger tips. As shown in Figure 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.

## 4.3 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 [James and Twigg 2005].

In this section, we introduce a novel metric for detecting deformation discontinuities using spectral clustering. We start by computing the deformation gradient $\Gamma_i^k$ [Sumner et al. 2005] 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

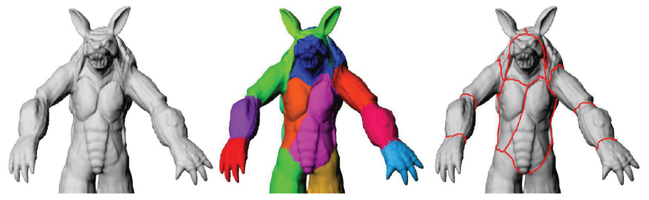

Fig. 6. Triangle clusters and their boundaries resulted from our new metric for spectral clustering using deformation gradients from the BALLET mesh animation.

in all frames as follows:

$$w_b(i, j) = \exp\left(-\frac{\sum_{k=1}^{n_a} \|\Gamma_i^k - \Gamma_j^k\|}{n_a \sigma_\Gamma}\right) \exp\left(-\frac{dist(f_i, f_j)}{\sigma_d}\right), \quad (5)$$

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 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 [James and Twigg 2005].

While previous work such as mean-shift clustering [James and Twigg 2005] or hierarchical clustering [Wang et al. 2007] can also effectively learn a set of proxy bones, certain drawbacks exist. As shown in Figure 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 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 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.

## 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.

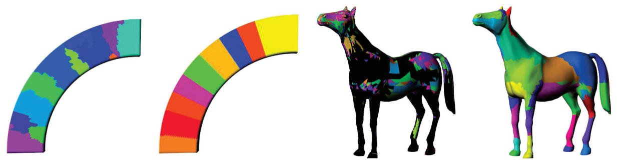

Fig. 7. A comparison of our spectral clustering method with hierarchical clustering [Wang et al. 2007] and mean-shift clustering [James and Twigg 2005] on two animation sequences, bending and horse collapsing, respectively. For the horse model, mean-shift clustering fails to assign a large number of 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.

### 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 [Lee et al. 1998]. 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 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 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 [Garland and Heckbert 1997]. Thus $V^s \subset V$. We prevent the collapse of any edge that connects a feature vertex with a nonfeature 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 [Lee et al. 1998].

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 8(c). Moreover, vertices assigned to two adjacent base domain triangles $f_1^s, f_2^s \in F^s$ can also be expressed in the same coordinate frame by flattening $f_1^s$ and $f_2^s$ onto the same plane. This parameterization will become very useful when mapping a base domain edge to a path on the original mesh. For detailed steps of building MAPS parameterization, please refer to the original paper [Lee et al. 1998].

Once we have the global parameterization from MAPS, patch generation becomes a straightforward process. As shown in Figure 9, a base domain edge $e^s = (v_1^s, v_2^s) \in E^s$ is shared by

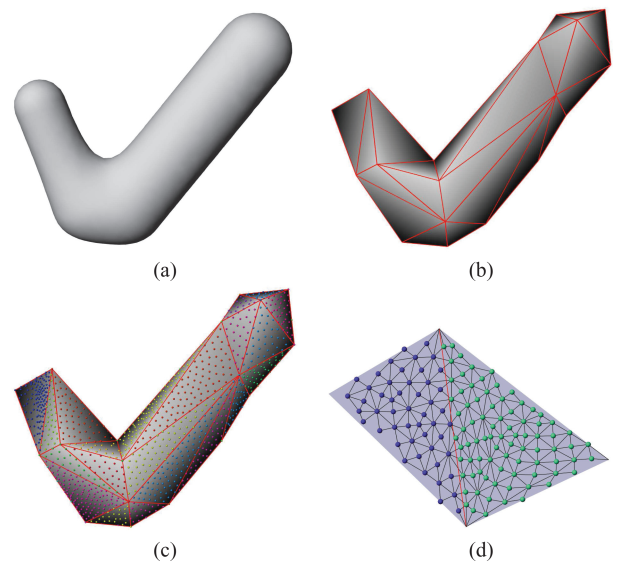

Fig. 8. The overall process of triangle patch generation. Start from an input mesh in (a), we first perform mesh simplification to generate a base mesh in (b). During simplification, we apply MAPS [Lee et al. 1998] to progressively parameterize the input mesh over the base mesh, as shown in (c). We utilize 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.

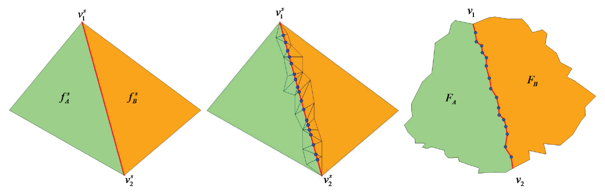

Fig. 9. Overview of path generation. Every edge $e^s = (v_1^s, v_2^s)$ in the base mesh (left) has a corresponding path $P = (v_1, v_2)$ in the original mesh (right). To generate such a path, we flatten a local region on the original mesh and intersect $e^s$ with triangles in the flattened region. A 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 $v_2$.

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 [Kraevoy et al. 2003; Kraevoy and Sheffer 2004; Schreiner et al. 2004]. 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 [Lee et al. 1998] to ensure that a straight line in the parametric domain always maps to a topological line in the original mesh. Therefore the preceding 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 [Lee et al. 1998]. Therefore a feature edge in the base complex can be mapped trivially to the corresponding feature curve on the original mesh.

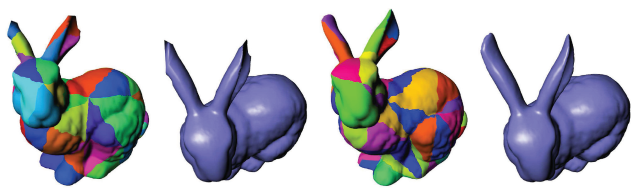

Fig. 10. Patches generated without feature constraints might not align their boundaries well with high curvature regions. Therefore the resulting reconstruction has more numerical and visual errors in these regions, such as the ears on the bunny.

Figure 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. Figure 10 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.

## 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 Floater [2003] to ensure no triangles flip in the embedding. The 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 [Soucy et al. 1996; Carr and Hart 2002].

*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 runtime 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.

## 6. GPU-BASED LEVEL-OF-DETAIL RENDERING

At runtime, 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 is 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.

### 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 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 runtime, we apply new bone transformations to the OBB corners and use the deformed bounding box for LOD selection. Specifically, given a set of corners $c_i, i = 0 \ldots 3$ which form the major axes of an OBB with $c_0$ being the pivoting point, and their skinning weights $w_i^b, b = 1 \ldots n_b$, we can compute the projected area $A$ as

$$A = \sum_{i>k} \|(\hat{c}_i - \hat{c}_0) \times (\hat{c}_k - \hat{c}_0)\|, \tag{6}$$

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 previous 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:

$$d = \max(\log_4(\alpha A), d_{max}), \tag{7}$$

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.

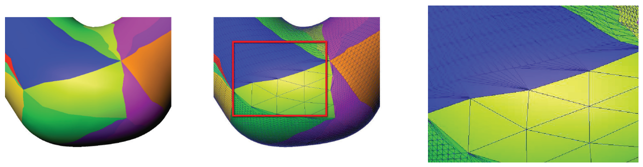
Fig. 11. Illustration of our boundary stitching method. (Left) Visualization of triangle charts on the mesh. Each color represents a distinct chart. (Middle) Stitching result along chart boundaries. (Right) Closer view of the stitching result. Although two adjacent charts have a significant difference in LOD, the stitching results are guaranteed to be watertight.

Although it would be more straightforward to implement the preceding 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.

### 6.2 Geometry Image Rendering

Since the multichart mesh geometry datasets generated in our preprocessing stage are in the form of multiresolution 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 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 similar manner. The final patch geometry can therefore be rendered in the vertex shader after skinning transformations have been applied.

### 6.3 Boundary Stitching

When different resolutions have been chosen for adjacent patches, cracks will occur at their common boundary. Since adjacent patches

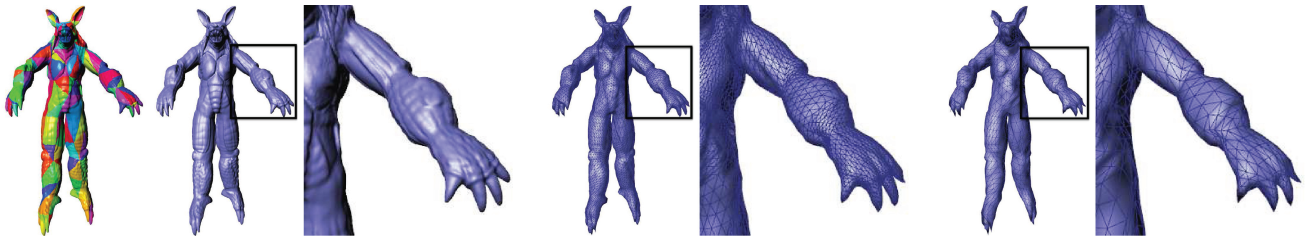

Fig. 12. Triangular charts and reconstructed meshes at varying levels of detail for a BALLET animation.

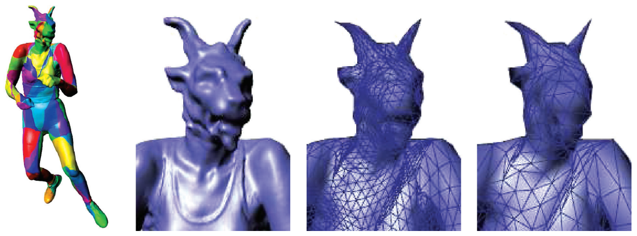

Fig. 13. Triangular charts and reconstructed meshes at varying levels of detail for a BOXING animation.

Table I. Statistics and Timing

| Examples | #Orig. Tris | Feature Time | Chart Time | #Charts | Max. Res. | 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 |

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 preprocessing 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.

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 $(0 \ldots w-1, 0 \ldots h-1)$. 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

$$\bar{u} = u - u \bmod 2^{(n-m)}, \bar{v} = v - v \bmod 2^{(n-m)}. \quad (8)$$

In our vertex shader implementation, the edge vertices along each boundary are identified. The texture coordinates of these vertices are then modified according to (8) to ensure that the vertex positions retrieved from the geometry image correctly align with the adjacent patch.

As shown in Figure 11, the preceding 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.

## 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 10, 17, 12, and 13. The results for static meshes are shown in 10 and 17, and those for deforming meshes are shown in 12 and 13. Timing and statistics for the preprocessing steps can be found in Table I. The skinning quality of oriented bounding boxes for triangular charts is shown Figure 14. Since we also treat deformation discontinuities as feature lines during chart generation, the resulting charts do not cross

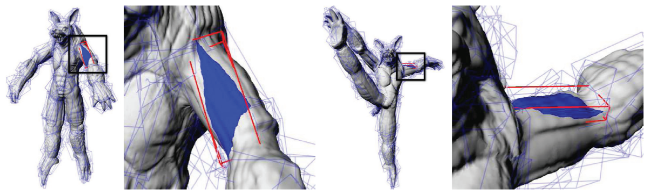

Fig. 14. Skinning results of an oriented bounding box for different poses. The red bounding box is associated with the patch in blue color. The skinned corners of the bounding box adequately approximate the bounding volume of the deformed patch at every different pose. Therefore we can use the skinned bounding box to estimate the projected screen area and thus determine the detail level for this chart at every pose.

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.

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) [Purnomo et al. 2004]. 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 Cignoni et al. [1998] which computes the Hausdorff distance between the original mesh and the reconstructed mesh. As shown in Figures 15 through 17 and Table II, 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

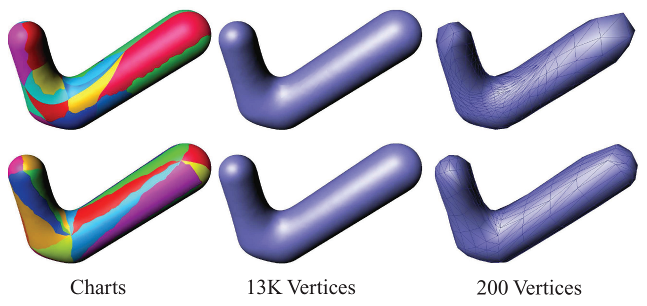

Fig. 15. A comparison between our triangle-chart geometry images (Bottom Row) and quad-chart geometry images [Purnomo et al. 2004](Top Row). For this simple model, both methods approximate the original mesh well at a high resolution. However, quad charts tend to have irregular shapes and produce lower-quality results at lower resolutions.

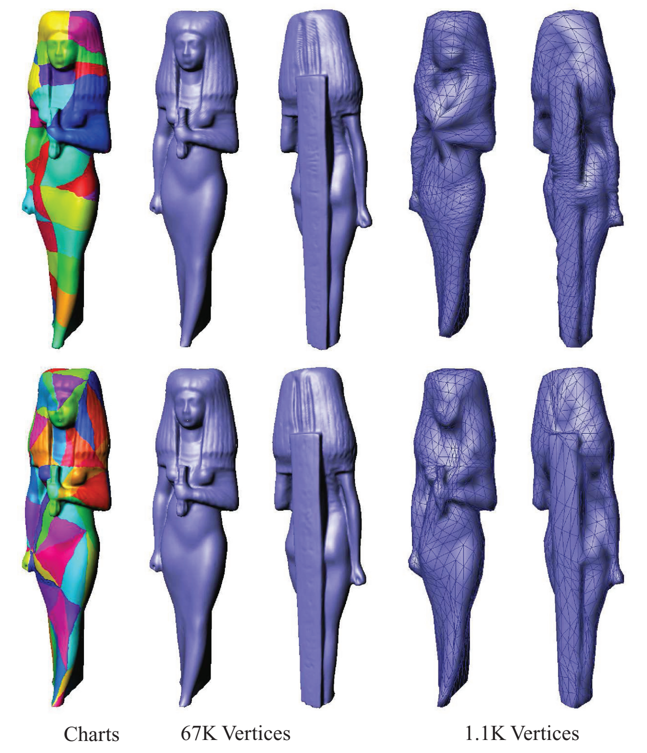

Fig. 16. Another comparison between our triangle-chart geometry images (Bottom Row) and quad-chart geometry images [Purnomo et al. 2004] (Top Row) using the Isis model. Although this model has a relatively simple shape, it also contains sharp edges and semantic features. Both methods can produce a good reconstruction in a high resolution. However, small artifacts can be noticed on the back of the head for quad charts due to irregular chart boundaries. Quad charts also fail to reconstruct important features faithfully in a low resolution while our method still gives a good approximation.

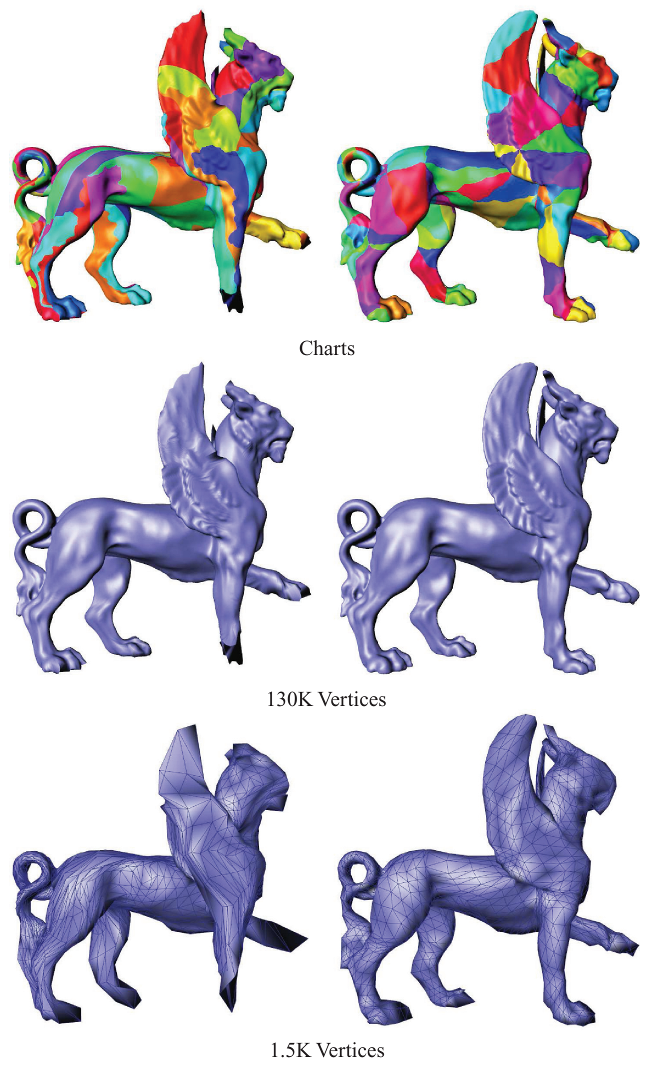

Fig. 17. Another comparison between our triangle-chart geometry images (Right Column) and quad-chart geometry images [Purnomo et al. 2004](Left Column). With the same number of vertices, the reconstructed meshes from our method are more faithful to the original mesh than the quad-chart based method. The feature constraints in our method ensure that important features are preserved during mesh simplification and result in higher-quality charts.

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 shapes 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 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 16, 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 4.3 that spectral clustering can extract higher-quality proxy bones faithful to the deformation structure. Here we further

Table II. Comparison of Reconstruction Error

| 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 |

Comparison of mesh reconstruction errors between our triangular geometry images and quad-chart-based geometry images [Purnomo et al. 2004].

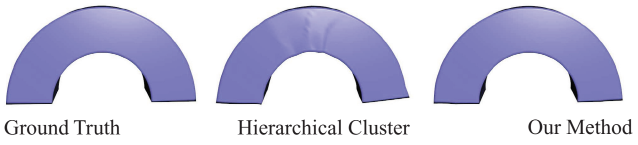

Fig. 18. Comparison of skinning quality between our method and the hierarchical clustering method in Wang et al. [2007]. Our method produces results with less artifacts.

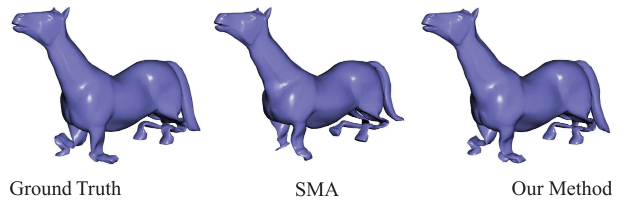

Fig. 19. Comparison of skinning quality between our method and SMA [James and Twigg 2005]. Our extracted proxy bones fit the mesh sequence well while SMA fails in highly deformable regions including the bending legs.

compare the resulting skin animations between our method and existing ones. In Figure 18, we show that the skinning results from spectral clustering are closer to the ground truth without noticeable artifacts, while the irregular cluster shapes from hierarchical clustering, adopted in Wang et al. [2007], tend to cause more obvious artifacts. In Figure 19, our result is compared with that from SMA [James and Twigg 2005] 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 James and Twigg [2005].

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 21 and 20. The rendering performance and other statistics of these scenes can be found in Table III. 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 I, 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

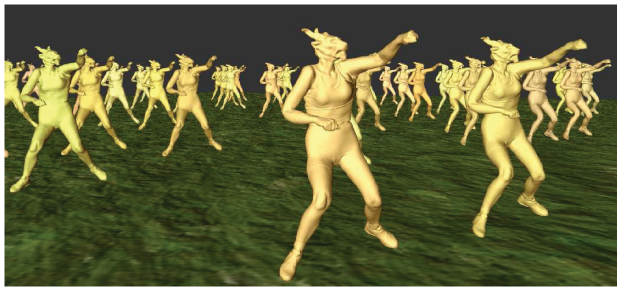

Fig. 20. Level-of-detail rendering of a large BOXING crowd.

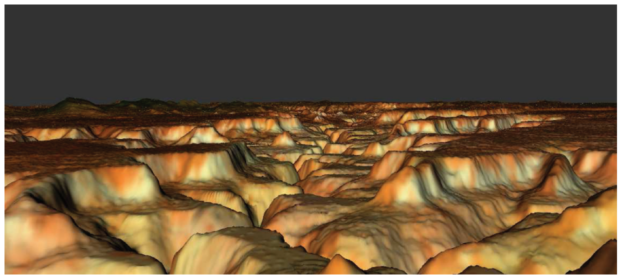

Fig. 21. Level-of-detail rendering of a terrain navigation using the Grand Canyon dataset.

Table III. Performance of Our LOD Rendering System

| 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 |

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.

required for a scene, there should be plenty of memory left for other GPU operations such as texturing and shading.

## 8. CONCLUSIONS AND FUTURE WORK

In this article, we introduce multichart 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 Niski et al. [2007] and render different subimages using different resolutions.

Although we did not notice any popping artifacts with our current scheme, this adaptive subdivision may also help producing a smoother transition between detail levels. 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 40MB 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.

## REFERENCES

BORGEAT, L., GODIN, G., BLAIS, F., MASSICOTTE, P., AND LAHANIER, C. 2005. GoLD: Interactive display of huge colored and textured models. *ACM Trans. Graph. 24*, 3, 869–877.

CARR, N. AND HART, J. 2002. Meshed atlases for real-time procedural solid texturing. *ACM Trans. Graph. 21*, 2, 106–131.

CARR, N., HOBEROCK, J., CRANE, K., AND HART, J. 2006. Rectangular multi-chart geometry images. In *Proceedings of the Eurographics/SIGGRAPH Symposium on Geometry Processing*. 181–190.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2004. Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Trans. Graph. 23*, 3, 796–803.

CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. 1998. Metro: Measuring error on simplified surfaces. *Comput. Graph. Forum 17*, 2, 167–174.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. *ACM Trans. Graph. 23*, 3, 905–914.

DECORO, C. AND RUSINKIEWICZ, S. 2005. Pose-Independent simplification of articulated meshes. In *Proceedings of the Symposium on Interactive 3D Graphics*. 17–24.

ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution analysis of arbitrary meshes. In *Computer Graphics Proceedings (SIGGRAPH 95)*. 173–182.

FLOATER, M. S. 2003. Mean value coordinates. *Comput. Aided Geom. Des. 20*, 1, 19–27.

FOWLKES, C., BELONGIE, S., CHUNG, F., AND MALIK, J. 2004. Spectral grouping using the Nyström method. *IEEE Trans. Pat. Anal. Mach. Intell. 26*, 2, 214–225.

GARLAND, M. AND HECKBERT, P. 1997. Surface simplification using quadric error metrics. In *Proceedings of the SIGGRAPH*. 209–216.

GU, X., GORTLER, S., AND HOPPE, H. 2002. Geometry images. *ACM Trans. Graph. 21*, 3, 355–361.

HOPPE, H. 1997. View-Dependent refinement of progressive meshes. In *Proceedings of the SIGGRAPH*. 189–198.

HWA, L., DUCHAINEAU, M., AND JOY, K. 2005. Real-Time optimal adaptation for planetary geometry and texture: 4-8 tile hierarchies. *IEEE Trans. Vis. Comput. Graph 11*, 4, 355–368.

JAMES, D. AND TWIGG, C. 2005. Skinning mesh animations. *ACM Trans. Graph. 24*, 3, 399–407.

JI, J., WU, E., LI, S., AND LIU, X. 2005. Dynamic LOD on GPU. In *Proceedings of the Computer Graphics International*. 108–114.

JULIUS, D., KRAEVOY, V., AND SHEFFER, A. 2005. D-charts: Quasi-developable mesh segmentation. *Comput. Graph. Forum 24*, 3, 981–990.

KALOGERAKIS, E., SIMARI, P., NOWROUZEZAHRAI, D., AND SINGH, K. 2007. Robust statistical estimation of curvature on discretized surfaces. In *Proceedings of the Eurographics/SIGGRAPH Symposium on Geometry Processing*. 13–22.

KATZ, S. AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph. 22*, 3, 954–961.

KIRCHER, S. AND GARLAND, M. 2005. Progressive multiresolution meshes for deforming surfaces. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 191–200.

KRAEVOY, V. AND SHEFFER, A. 2004. Cross-Parameterization and compatible remeshing of 3d models. *ACM Trans. Graph. 23*, 3, 861–869.

KRAEVOY, V., SHEFFER, A., AND GOTSMAN, C. 2003. Matchmaker: Constructing constrained texture maps. *ACM Trans. Graph. 22*, 3, 326–333.

LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. 1998. MAPS: Multiresolution adaptive parameterization of surfaces. In *Computer Graphics Proceedings (SIGGRAPH 98)*. 95–104.

LEE, Y., LEE, S., SHAMIR, A., COHEN-OR, D., AND SEIDEL, H.-P. 2005. Mesh scissoring with minima rule and part salience. *Comput. Aided Geom. Des. 22*, 5, 444–465.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph. 21*, 3, 362–371.

LIU, R. AND ZHANG, H. 2004. Segmentation of 3d meshes through spectral clustering. In *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications*. 298–305.

LOSASSO, F. AND HOPPE, H. 2004. Geometry clipmaps: Terrain rendering using nested regular grids. *ACM Trans. Graph. 23*, 3, 769–776.

LUEBKE, D. AND ERIKSON, C. 1997. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of the SIGGRAPH*. 199–208.

MOHR, A. AND GLEICHER, M. 2003. Deformation sensitive decimation. Tech. rep., University of Wisconsin Graphics Group.

NISKI, K., PURNOMO, B., AND COHEN, J. 2007. Multi-grained level of detail using a hierarchical seamless texture atlas. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*. 153–160.

PURNOMO, B., COHEN, J., AND KUMAR, S. 2004. Seamless texture atlases. In *Proceedings of the Eurographics/SIGGRAPH Symposium on Geometry Processing*. 65–74.

SANDER, P., SNYDER, J., GORTLER, S., AND HOPPE, H. 2001. Texture mapping progressive meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. 409–416.

SANDER, P., WOOD, Z., GORTLER, S., SNYDER, J., AND HOPPE, H. 2003. Multi-chart geometry images. In *Proceedings of the Eurographics/SIGGRAPH Symposium on Geometry Processing*. 146–155.

SCHREINER, J., ASIRVATHAM, A., PRAUN, E., AND HOPPE, H. 2004. Inter-surface mapping. *ACM Trans. Graph. 23*, 3, 870–877.

SHI, J. AND MALIK, J. 2000. Normalized cuts and image segmentation. *IEEE Trans. Pat. Anal. Mach. Intell. 22*, 8, 888–905.

SOUCY, M., GODIN, G., AND RIOUX, M. 1996. A texture-mapping approach for the compression of colored 3d triangulations. *The Visual Comput. 12*, 503–514.

STYLIANOU, G. AND FARIN, G. 2004. Crest lines for surface segmentation and flattening. *IEEE Trans. Visualiz. Comput. Graph. 10*, 5, 536–544.

SUMNER, R., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. 2005. Mesh-based inverse kinematics. *ACM Trans. Graph. 24*, 3, 488–495.

WANG, R., PULLI, K., AND POPOVIĆ, J. 2007. Real-time enveloping with rotational regression. *ACM Trans. Graph. 26*, 3, 1174–1179.

XIA, J. AND VARSHNEY, A. 1996. Dynamic view-dependent simplification for polygonal models. In *Proceedings of the 7th Conference on Visualization*. 327–334.

YAMAUCHI, H., GUMHOLD, S., ZAYER, R., AND SEIDEL, H.-P. 2005. Mesh segmentaion driven by Gaussian curvature. *Visual Comput. 21*, 659–668.

YAO, C.-Y. AND LEE, T.-Y. 2008. Adaptive geometry image. *IEEE Trans. Visualiz. Comput. Graph. 14*, 4, 948–960.

YUKSEL, C., KEYSER, J., AND HOUSE, D. H. 2008. Mesh colors. Tech. rep. tamu-cs-tr-2008-4-1, Department of Computer Science, Texas A&M.

ZHANG, E., MISCHAIKOW, K., AND TURK, G. 2005. Feature-Based surface parameterization and texture mapping. *ACM Trans. Graph. 24*, 1, 1–27.

ZHOU, K., SNYDER, J., GUO, B., AND SHUM, H.-Y. 2004. Iso-Charts: Stretch-Driven mesh parameterization using spectral analysis. In *Proceedings of the Eurographics/SIGGRAPH Symposium on Geometry Processing*. 45–54.

Received September 2008; revised July 2009; accepted November 2009