# Geometry Images

Xianfeng Gu
Harvard University

Steven J. Gortler
Harvard University

Hugues Hoppe
Microsoft Research

## Abstract

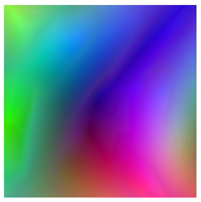
Surface geometry is often modeled with irregular triangle meshes. The process of remeshing refers to approximating such geometry using a mesh with (semi)-regular connectivity, which has advantages for many graphics applications. However, current techniques for remeshing arbitrary surfaces create only *semi-regular* meshes. The original mesh is typically decomposed into a set of disk-like charts, onto which the geometry is parametrized and sampled. In this paper, we propose to remesh an arbitrary surface onto a *completely regular* structure we call a *geometry image*. It captures geometry as a simple 2D array of quantized points. Surface signals like normals and colors are stored in similar 2D arrays using the same implicit surface parametrization — texture coordinates are absent. To create a geometry image, we cut an arbitrary mesh along a network of edge paths, and parametrize the resulting single chart onto a square. Geometry images can be encoded using traditional image compression algorithms, such as wavelet-based coders.

**Keywords:** remeshing, surface parametrization.

## 1 INTRODUCTION

Surface geometry is often modeled with irregular triangle meshes. The process of *remeshing* refers to approximating such geometry using a mesh with (semi)-regular connectivity (e.g. [3, 13]). Resampling geometry onto a regular structure offers a number of benefits. Compression is improved since the connectivity of the samples is implicit. Moreover, remeshing can reduce the non-uniformity of the geometric samples in the tangential surface directions, thus reducing overall entropy [10]. The regularity of sample neighborhoods helps in applying signal-processing operations and in creating hierarchical representations for multiresolution viewing and editing [14, 24].

However, current techniques for remeshing arbitrary surfaces create only *semi-regular* meshes. The original mesh is typically decomposed into a set of disk-like charts, onto which the geometry is parametrized and sampled. Although the sampling on each chart follows regular subdivision, the chart domains form an *irregular* network over the surface. This irregular domain network complicates processing, particularly for operations that require accessing data across neighboring charts. In contrast, texture data is typically represented in a completely regular fashion, as a (possibly compressed) 2D array of $[r, g, b]$ values. This distinction, among others, causes geometry and textures to be treated and represented quite differently by current graphics hardware.


Stanford bunny

In this paper, we propose to remesh an arbitrary surface onto a *completely regular* structure we call a *geometry image*. It captures geometry as a simple $n \times n$ array of $[x, y, z]$ values. Other surface attributes, such as normals and colors, are stored as additional square images, sharing the same domain as the geometry. Because the geometry and attributes share the same parametrization, the parametrization itself is implicit — "texture coordinates" are absent. Moreover, this parametrization fully utilizes the texture domain (with no wasted space). Geometry images can be encoded using traditional image compression algorithm, such as wavelet-based coders. Also, geometry images are ideally suited for hardware rendering. They may be transmitted to the graphics pipeline in a compressed form just like texture images. And, they eliminate expensive pointer-based structures such as indexed vertex lists.

Of course, arbitrary surfaces cannot generally be mapped directly onto a square image domain, because their topology can differ from that of a disk. The basic idea in our approach is to slice open the mesh along an appropriate set of cut paths, to allow the unfolding of the mesh onto a disk-like surface. The vertices and edges along the cut paths are represented redundantly (typically twice) along the boundary of this disk. Next, we parametrize this cut surface onto the square domain of the image, and sample the geometry at the 2D grid samples.

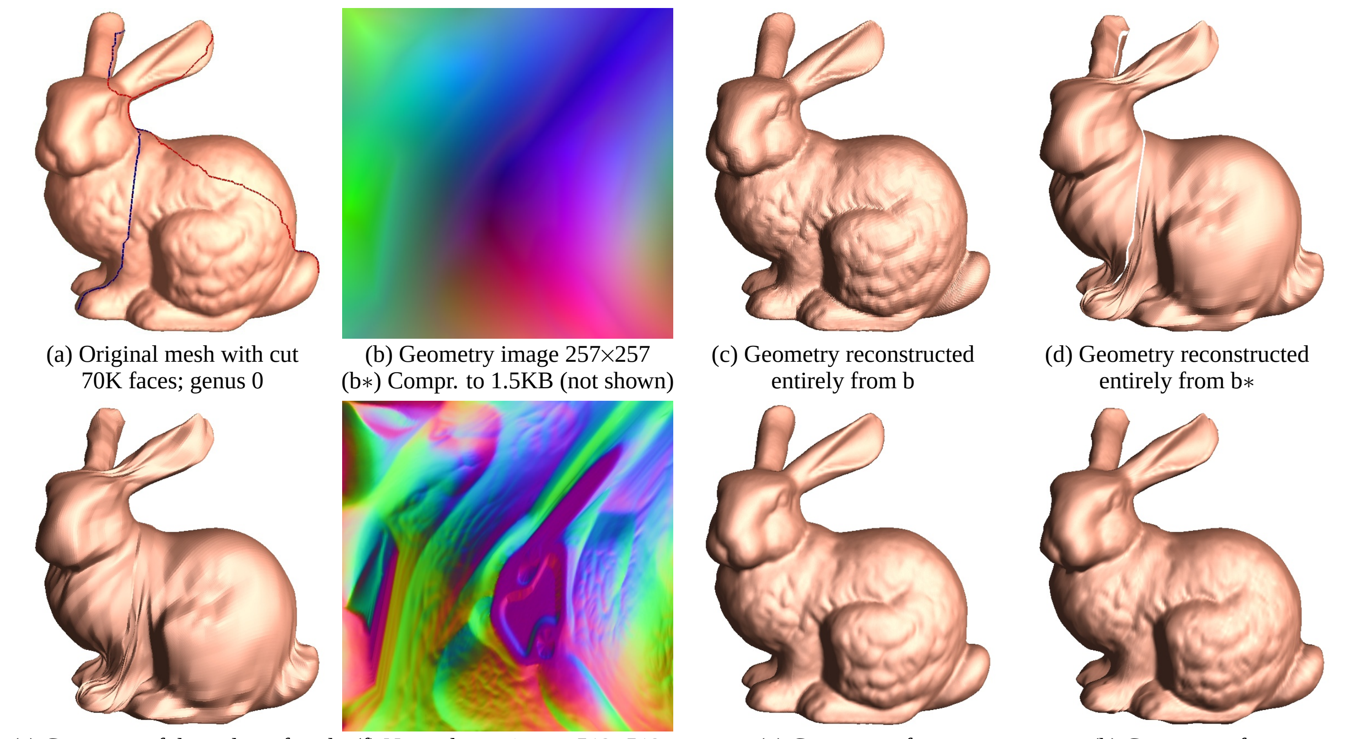
Representing surfaces as geometry images presents challenges:

- A cut must be found that opens the mesh into a topological disk, and that also permits a good parametrization of the surface within this disk. We describe an effective, automatic method for cutting arbitrary 2-manifold meshes (possibly with boundaries).
- The image boundary must be parametrized such that the reconstructed surface matches exactly along the cut, to avoid cracks. Traditional texture mapping is more forgiving in this respect, in that color discontinuities at boundaries are less noticeable.
- The parametrization must evenly distribute image samples over the surface, since undersampling would lead to geometric blurring. We do not make a technical contribution in this area, but simply apply the geometric-stretch parametrization of [18, 17].
- Straightforward lossy compression of the geometry image may introduce tears along the surface cut. We allow fusing of the cut by encoding the cut topology as a small data sideband.

Geometry images have the following limitations:

- They cannot represent non-manifold geometry.
- Unwrapping an entire mesh as a single chart can create parametrizations with greater distortion and less uniform sampling than can be achieved with multiple local charts, particularly for surfaces of high genus.

In this paper, we describe an automatic system for converting arbitrary meshes into geometry images and associated attribute maps (Figure 1). We demonstrate that they form a practical and elegant representation for a variety of graphical models (Figure 7).



(a) Original mesh with cut 70K faces; genus 0

(b) Geometry image 257×257 (b∗) Compr. to 1.5KB (not shown)

(c) Geometry reconstructed entirely from b

(d) Geometry reconstructed entirely from b∗

(e) Geometry of d topology-fused using sideband data

(f) Normal-map image 512×512 (f∗) Compr. to 24KB (not shown)

(g) Geometry of c normal-mapped using f

(h) Geometry of e normal-mapped using f∗

Figure 1: Creation, compression, and rendering of a geometry image. Images b∗ and f∗ (not shown) are compressed using an image wavelet-coder. Geometry image is 12-bit $[x, y, z]$ visualized as $[r, g, b]$. Normal-map image is 8-bit $[n_x, n_y, n_z]$ visualized as $[r, g, b]$.

## 2 PREVIOUS WORK

There exist several schemes for semi-regular remeshing of arbitrary surfaces. Eck et al. [3] achieve remeshing by cutting a mesh into multiple charts using a Voronoi-like decomposition. Each chart is parametrized using a harmonic map, sampled using a regular triangular subdivision pattern, and compressed using a triangular wavelet construction [14]. Khodakovsky et al. [10] use the MAPS scheme [13] to partition the mesh into charts and create the chart parametrizations. They obtain impressive compression results using zero-tree coding of local-frame wavelet coefficients. Lee et al. [12] create a multi-chart domain using mesh simplification. They define a subdivision surface over this domain and fit it to the original surface. The fit residual is expressed as a semi-regular scalar displacement map over the smooth subdivision surface. Guskov el al. [7] use a MAPS-like approach to create multiple charts. These charts are recursively subdivided, and newly introduced vertices are expressed using displacements from the previous mesh, mostly as scalar displacements.

In our setting, previous semi-regular remeshing approaches can be viewed as representing a surface as a collection of abutting geometry images. The crux of our contribution is to represent the entire surface as a single geometry image, by cutting the surface and sampling it using a completely regular quad grid. We optimize the creation of the cut to allow for a good parametrization.

## 3 CREATION OF GEOMETRY IMAGES

From a 2-manifold triangle mesh $M$, we create a geometry image consisting of an $n \times n$ array of $[x, y, z]$ data values. If we plan to render using normal mapping, we also create another 2D array of normal values $[n_x, n_y, n_z]$. (See Figure 1.)

Our approach is to cut the mesh $M$ to form a new mesh $M'$ that has the topology of a disk (Figure 2). The cut $\rho$ is specified as a set of edges in $M$. To create $M'$, we split each non-boundary edge in $\rho$ into two boundary edges to form the *opened cut* $\rho'$. This directed loop of edges $\rho'$ is the boundary of $M'$. We say that two edges in $\rho'$ are *mates* if they result from the splitting of an edge in $\rho$.

A vertex $v$ with valence $k$ in $\rho$ is replicated as $k$ vertices in $\rho'$. Vertices in $\rho$ that have valence $k \neq 2$ in the cut are called *cut-nodes*. (We still refer to these as cut-nodes when replicated in $\rho'$.) A *cut-path* is the set of boundary edges and vertices between two ordered cut-nodes in the loop $\rho'$. Each cut-path has a mate defined by the mates of its edges (unless its edges were boundary edges in $\rho$).

Let $D$ be the domain unit square for the geometry image. The parametrization $\phi$ is a piecewise linear map from the unit square $D$ to $M'$, defined by associating domain coordinates $(s, t)$ with each mesh vertex in $M'$. The domain $D$ has a rectilinear $n \times n$ grid, where *grid points* have coordinates $(i/(n-1), j/(n-1))$ with $i, j = 0..n-1$. We evaluate $\phi$ at the grid points to sample the mesh geometry, as well as any other surface attributes (e.g. color, skinning weights, radiance transfer coefficients).

The geometry image samples are used to reconstruct an approximation of $M$. In this work, we use linear basis functions (triangles) to define the reconstruction interpolant for geometry. Our goal is to find a good cut $\rho$ and parametrization $\phi$, such that this reconstruction is a good approximation of $M$ for moderate sampling rates.

**Approach overview** Our strategy for finding a good cut $\rho$ and parametrization $\phi$ is as follows. We first find a topologically sufficient cut, and create an initial parametrization using this cut. We use information from the parametrization to improve the cut, and reparametrize based on the new cut. This process of cutting and reparametrizing is iterated until the parametrization no longer improves. To aid in the exposition, we first describe how a parametrization is found given any cut $\rho$ (Section 3.1). We then describe how the space of cuts is explored (Section 3.2).

## 3.1 Parametrization

For now, assume that we are given a cut $\rho$. To create a parametrization, we first fix a mapping between the opened cut $\rho'$ and the boundary of the unit square $D$. Next, we solve for a map of $M'$ onto $D$ that is consistent with these boundary conditions. We now describe these two steps in more detail.

**Boundary parametrization** In order to avoid cracks in the reconstructed geometry, it is necessary that each cut-node in $\rho'$ be exactly sampled in the remesh. This implies that we must map cut-nodes to grid points on the boundary of $D$. (Other vertices in $\rho'$ are not constrained to lie on grid points.) In addition, cut-path mates must be sampled at identical surface points to avoid cracks, which requires that cut-path mates be allocated the same length on the boundary of $D$. To accomplish this, we allocate for each cut-path an amount of the boundary proportional to its length in $\rho'$. This allocation is then rounded to an integer multiple of $1/(n-1)$. If due to rounding we have over- or under-allocated the boundary, we redistribute the residual to the various cut-paths in units of $1/(n–1)$, making sure to treat cut-path mates identically. Note that an $n \times n$ geometry image can represent a surface with genus at most $n$.

To avoid degeneracies, we must enforce two more constraints. First, no triangle in $M'$ can have all its three vertices mapped to one of the four sides of the square, for it would become parametrically degenerate. If such a triangle arises, we split the triangle by introducing new vertices at the midpoints of its non-boundary edge(s), and split neighboring triangles so as to avoid T-junctions.

Second, as we lay out $\rho'$ along the boundary of $D$ we must break any edge that spans one of the four corners of $D$. Otherwise a single boundary edge in $M'$ would map to an "L" shape in $D$. The edge is broken by introducing a vertex at the domain corner, thus splitting its adjacent triangle into two. To enforce topological consistency across the cut, the same procedure is applied to its mate edge.

Finally, we find that placing a valence-1 cut-node at a corner of $D$ results in poor geometric behavior, so if this occurs we rotate the boundary parametrization.

**Interior parametrization** Having fixed the boundary of the parametrization, we now solve for its interior. When creating a parametrization, there are numerous metrics that can be used to measure its quality, e.g. [3, 6, 8, 13, 16]. For our application, an ideal metric would be some measure of surface accuracy after sampling and reconstruction. As shown by the analysis in [17], the $L^2$ geometric-stretch metric introduced in [18] is in fact an approximation of this ideal measure.

Geometric stretch measures the amount of spacing that occurs on the surface when the parameter domain is uniformly sampled. Thus, minimizing geometric stretch tends to uniformly distribute samples on the surface. In [17], the stretch metric is shown to be related to signal-approximation error (SAE) — the difference between a signal defined on the surface and its reconstruction from a discrete grid sampling. Specifically, the stretch metric corresponds to the first-order Taylor expansion of SAE under the assumption of locally constant reconstruction. In our context, the signal is the geometry itself, and therefore geometric stretch can be seen as a predictor of geometric reconstruction error. In Section 5, we show the advantage of using a geometric-stretch parametrization over the Floater "shape-preserving" parametrization.

We compute a geometric-stretch parametrization using the hierarchical optimization algorithm described in [17]. First, the interior of $M'$ is simplified to form a progressive mesh representation [9]. The few interior vertices in the resulting base mesh are optimized within $D$ by brute-force. Then, we apply vertex splits from the progressive mesh to successively refine the mesh. For each inserted vertex, we optimize the parametrization of its neighborhood to minimize stretch using a local, non-linear optimization algorithm.

## 3.2 Cutting

We now describe how we automatically find a good cut $\rho$ for $M$. Starting with a surface of arbitrary genus, we first find an initial cut that opens $M$ into a disk. Given the resulting topological disk, we use a novel algorithm to augment the cut in order to improve the subsequent parametrization and reconstruction quality.

**Initial cut** It is well known that any closed surface can be opened into a topological disk (called a polygonal schema) by cutting along an appropriate set of edges [15]. Such a cut was used in [5] as part of a geometric modeling system for creating smooth surfaces. Piponi and Borshukov [16] describe an interactive system allowing a user to manually cut a genus-zero manifold into a single chart using a tree of edge cuts.

The computational complexity of optimally cutting a mesh of arbitrary genus into a disk is studied in [4]. Algorithms for finding special kinds of cuts (those that form reduced and canonically reduced polygonal schemata) are described in [2, 11, 21].

Our algorithm, which is most similar to that of [2], works as follows. If the mesh has boundaries, let $B$ be the set of original boundary edges. This set remains frozen throughout the algorithm, and is always a subset of the final cut $\rho$. After removing a single seed triangle from the mesh, we apply two phases.

In the first phase we repeatedly identify an edge $e \notin B$ adjacent to exactly one triangle, and remove both the edge and the triangle. Note that the two remaining edges of the triangle are left in the simplicial complex, even if they are dangling. In order to obtain a result of "minimal radius", we order triangle removals according to their geodesic distance from the seed triangle. When this first phase terminates, we have removed a topological disk that includes all of the faces of the mesh. Thus, the remaining vertices (which is in fact all of them), and the remaining edges must form a topological cut $\rho$ of $M$. At this point, $\rho$ consists of a set of connected loops along with some unnecessary trees of edges (and is similar to the construction of [20]).

In a second phase, we repeatedly identify a vertex adjacent to exactly one edge (i.e. a dangling edge), and remove both the vertex and the edge. This second phase terminates when all the edge trees have been trimmed away, leaving just the connected loops. Since the resulting cut $\rho$ may be serrated (it is not made up of shortest paths), we straighten each cut-path in $\rho$ by computing a constrained shortest path that connects its two adjacent cut-nodes and stays within a neighborhood of the original cut-path.

For the case of a closed mesh of genus 0, the resulting $\rho$ will consist of a single vertex, since it has no loops. Because our parametrization requires that we map $\rho'$ onto a square, we add back to $\rho$ two adjacent mesh edges.

**Iterated cut augmentation** Through experiment, we have found that to obtain efficient geometry images, it is important for $\rho$ to pass through the various "extrema" of $M$. For example, in the hand model a good cut should pass through its five fingers (see Figure 2). Therefore our goal is to find these extrema and augment the cut so that it passes through them. A similar subproblem is investigated by Sheffer [19], who classifies extrema as vertices with high (discrete) curvature. Unfortunately, this type of local method will not be able to find protrusions with widely distributed curvature.

Our approach to finding extrema is to search for mesh regions that behave poorly (have large geometric stretch) under a parametrization using the current cut. Specifically, we map the vertices of $\rho'$ to the unit circle $C$, spaced according to their edge lengths over the surface. (We use the unit circle at this point instead of the unit square in order to avoid boundary constraints.) The cut mesh $M'$ is parametrized into the interior of $C$ using the shape-preserving parametrization of Floater [6]. Given the resulting map we identify the triangle with maximum geometric stretch, and pick one of its vertices as an extremal vertex.

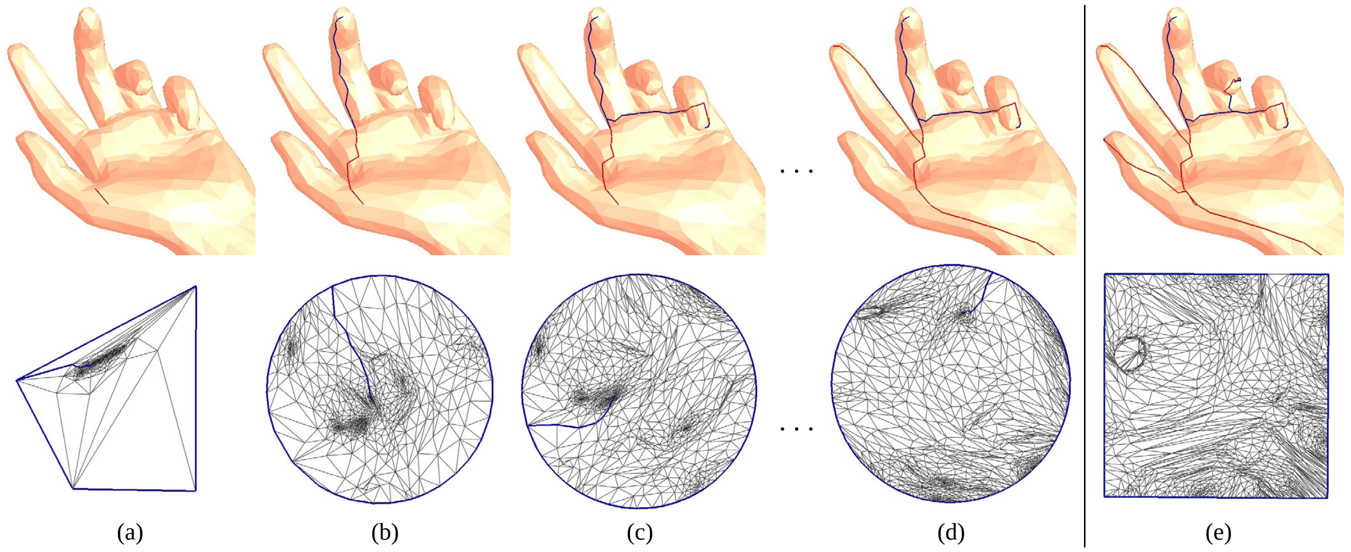


Figure 2: Columns (a–d) show iterations of the cut improvement algorithm. Upper images show the mesh $M$ with the current cut $\rho$ (blue except red where occluded). Bottom images show the Floater parametrization (over circle) of the corresponding $M'$, together with the shortest path to an extremal point, which will be added to $\rho$. Column (e) shows the final cut $\rho$ and the geometric-stretch parametrization (over square).

The intuition for this method is that any protrusion of the mesh experiences high geometric stretch under a Floater parametrization. For instance, it can be shown that when parametrizing a tube closed at its top and open at its base, a triangle at a height $h$ from the base has geometric stretch exponential in $h$, reaching a maximum at the tube apex. It is important to use the Floater parametrization for protrusion detection, since the geometric-stretch parametrization would evenly distribute stretch, thus hiding the extrema.

Having identified an extremal point, we find the shortest path from it to the current boundary of $M'$ (measuring distance on the mesh), and add this path to $\rho$. This maintains the invariant that $\rho$ is a valid cut of $M$.

We repeatedly apply this augmentation process, as shown in Figure 2. To determine when to stop, we run our geometric-stretch parametrization algorithm (Section 3.1) after each cut, and stop if the geometric stretch increases.

As a further improvement in the case of genus-zero meshes, when we find the *first* extremal point, we discard the original cut, which was based on an arbitrary random seed point, and replace the cut with a pair of adjacent edges at this extremum.

**Cutting summary** This pseudocode summarizes our algorithm:

```
function Cut_and_parametrize(mesh $M$)
    Remove seed triangle.
    while there remains an edge $e$ adjacent to only one triangle $t$
        Remove $e$ and $t$.
    while there remains a vertex $v$ adjacent to only one edge $e$
        Remove $v$ and $e$.
    Cut $\rho$ := remaining edges and vertices.
    if only a single vertex remains in $\rho$ then
        Add back two adjacent edges to $\rho$.
    Straighten each cut-path in $\rho$.
    Param $\phi$ := geometric-stretch parametrization using $\rho$.
    repeat
        $f$ := Floater parametrization using $\rho$.
        $t$ := triangle with maximal stretch under $f$.
        $s$ := shortest path on $M$ from $t$ to $\rho$.
        $\rho'$ := $\rho + s$.
        $\phi'$ := geometric-stretch parametrization using $\rho'$.
        if stretch($\phi'$) $>$ stretch($\phi$) break.
        $\rho := \rho'$;   $\phi := \phi'$.
    Report cut $\rho$ and parametrization $\phi$.
```

## 3.3 Topological sideband

A geometry image is a parametric sampling of the topological disk $M'$. Its reconstruction looks like $M$ because its boundary vertices coincide geometrically. For some applications though, it is important to be able to "fuse" the boundary of $D$ so that it has the original topology of $\rho$. This fusing could be achieved by searching for geometric correspondences on the image boundary, but this process might be error-prone, particularly if the geometry image undergoes lossy compression.

Since the necessary topological cut information is extremely compact, we record it into a sideband signal as follows. We associate a pair of labels e.g. $\{a, \overline{a}\}$ to each cut-path and its mate. We then store the string of labels corresponding to the sequence of cut-paths on the boundary of $M'$, e.g. $ab\overline{a}\overline{b}c\overline{c}$. From this string, we can recover the topology of the cut, i.e. the valence $k$ of each cut-node in $\rho$ and the ordering of the cut-nodes along $\rho'$. We also store for each cut-path $a$ its discretized length on the boundary of domain $D$, and we store the starting boundary location of the first cut-path. From this topological and parametric information, we can later establish the correspondence of all boundary grid vertices.

The size of this sideband information is $O(q \log n)$ bits, where $q$ is the number of cut-paths and $n$ is the sampling rate over $D$. For our models, $q$ ranges between 3 and 10, and the sideband is approximately 12 bytes long.

## 4 APPLICATIONS

**Rendering** To render geometry images on current hardware, we span each $2 \times 2$ quad of grid points using two triangles, by splitting along the shorter of the two diagonals.

Level-of-detail rendering is implemented by mip-mapping the geometry image, as shown in Figure 3. In order to avoid cracks at multiple levels of details, we use geometry images of size $(2^j + 1) \times (2^j + 1)$, and minify using simple sub-sampling. Also, the boundary mapping $\phi$ of Section 3.1 is constructed to place cut-nodes to grid-points of the lowest intended resolution ($65 \times 65$ for all of our examples). Unlike [16], our boundary samples coincide exactly across the cut so we need no special boundary treatment, even for mip-mapping.

For hardware that implements normal mapping, we also create a normal map using the exact same parametrization $\phi$. Usually, we sample the normals into an image of higher resolution than the

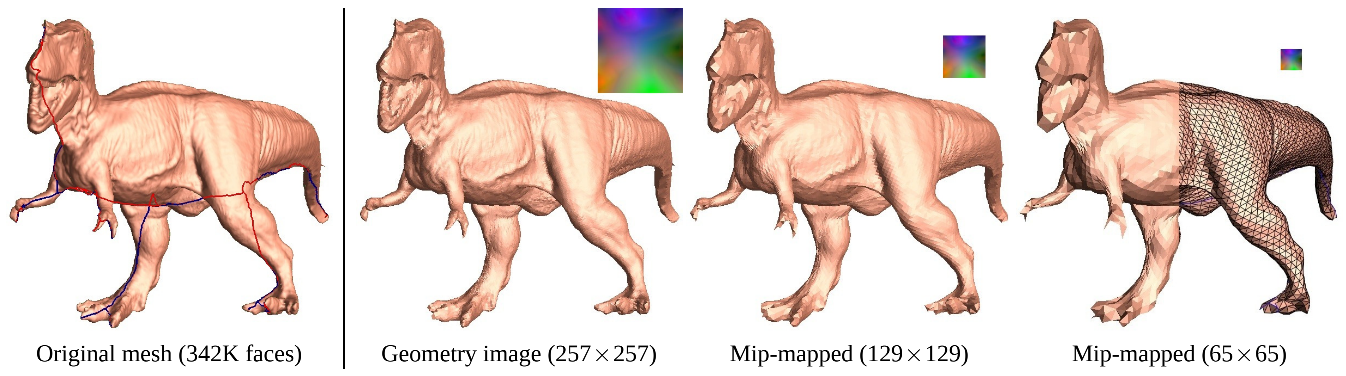


Figure 3: Mip-mapping a geometry image. As in all examples, the boundary parametrization is constructed for a $65 \times 65$ domain grid.

geometry since the normal-map signal tends to be more detailed. During rendering, the normal-map signal is rasterized over the triangles by hardware texture-mapping, using bilinear reconstruction of each quad in the normal map. (Texture coordinates at the vertices are assigned the range $[(0.5)/n', \ldots, (n'-0.5)/n']$ where $n'$ is the texture resolution, for correspondence with the texture samples.)

Because geometry images have the same regular structure as texture images, one can envision hardware that would use bilinear (or even bicubic) basis functions to reconstruct the geometry. Moreover, the rendering process should be inherently simpler than with traditional texture mapping. The attribute samples can be accessed in *scan order* rather than backward-mapped through random-access texture coordinates. Also, the attribute samples have a *regular correspondence* with the geometry samples, and therefore do not require general tri-linear interpolation lookup.

Both view-frustum and backface culling could be implemented in a unified setting by constructing hierarchies on the geometry image and the normal image respectively.

**Compression and Decompression** For compression we use the image-compression coder provided by Davis [1]. For decompression, we decode the wavelet coefficients to recreate an $n \times n$ grid of $[x, y, z]$ values. Our wavelet decoder produces floating-point coordinate values as output. Quantizing these values to 12-bit integers provides sufficient resolution for our models.

Since this wavelet coding is lossy, cut-path mates may be reconstructed differently, leading to cracks in the mesh (see Figure 1d). To address this problem, we also record and losslessly compress the topological sideband (Section 3.3). During decompression, we use this topological information to geometrically fuse the cut. We first determine the equivalence classes of boundary grid points. Most boundary grid point are paired up with a single other grid point, while grid points that sample a cut-node are grouped with $k-1$ other grid points, where $k$ is the valence of the cut-node in $\rho$. We average together the $[x, y, z]$ values of equivalent grid points, and replace their data with this common average. We record the vector displacement added due to this averaging for later error diffusion.

This simple averaging scheme gives rise to a continuous surface, but can lead to unsightly steps in the reconstructed geometry near the cut. In order to smooth these steps, we apply a simple error diffusion technique, spreading the displacements towards the center of the square. The result of this fusing process is shown in Figure 1e.

## 5 RESULTS

We have run our system on a number of high-resolution models, with and without boundaries. Uncompressed examples are shown in Figure 7. These required about an hour to convert offline. The conversion bottleneck is the sequence of parametrizations in the iterated cut augmentation process. Currently, we set the geometry image resolution $n$ manually (most often $n = 257$), but this parameter could be set automatically to achieve a desired accuracy.

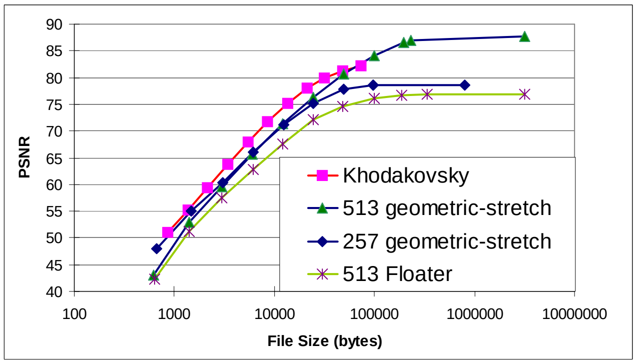


Figure 4: Rate distortion for geometric reconstruction from compressed geometry images of the bunny (at $257 \times 257$ and $513 \times 513$ resolutions, and using a Floater-parametrization), compared to [10].

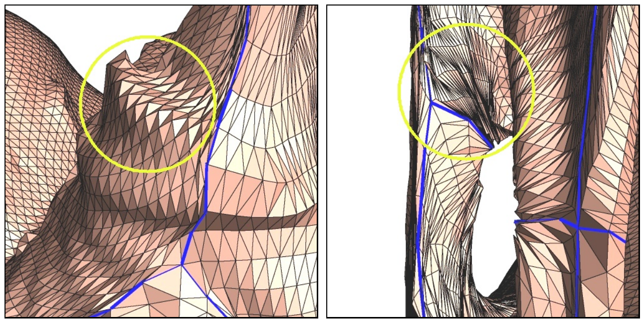


Figure 5: Example artifacts in the Buddha geometry image: aliasing (jaggedness) near sharp features, and regions of high anisotropy.

Geometry images tend to be relatively smooth, and therefore provide opportunity for compression. Even simple image compressors will define basis functions that span the whole surface, and therefore allow high compression ratios. Figure 4 shows rate-distortion curves when using the image wavelet-coder of [1]. These curves measure the reconstruction accuracy for various compression rates applied to the geometry image. Error is measured as Peak Signal to Noise Ratio PSNR = $20 \log_{10}(\text{peak}/d)$, where peak is the bounding box diagonal and $d$ is the symmetric rms Hausdorff error (geometric distance) between the original mesh and the reconstructed geometry. The blue curves show results for wavelet-compressed geometry image created using a geometric-stretch parametrization and two different sampling rates. The green curve corresponds to a geometry image formed using the same cut, but with a Floater parametrization, and is noticeably less efficient. For comparison, the red curve is the result of the compression scheme described in [10], which is more efficient by about 3dB. Reconstructions from compressed geometry images are shown in Figure 6.

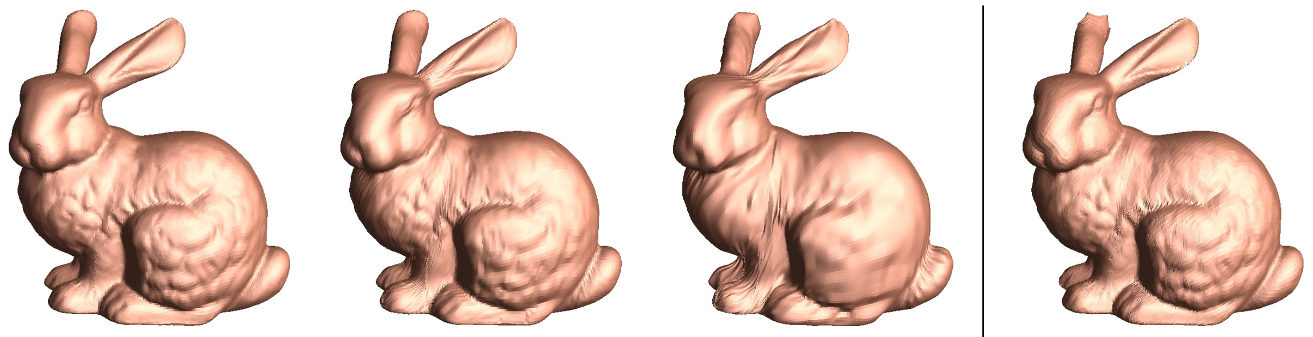


Figure 6: (a–c) Surfaces reconstructed from a $257 \times 257$ geometry image under increasing levels of wavelet compression. (d) Reconstructed from a $257 \times 257$ Floater-parametrized geometry image. All models are flat-shaded.

## 6 SUMMARY AND DISCUSSION

We have introduced geometry images, a completely regular representation for approximating the geometry of an irregular mesh. Geometry images can be easily rendered and compressed using current hardware and software. Due to their simplicity, we envision that geometry images may inspire new hardware rendering approaches.

We have found that we can create efficient geometry images on a wide variety of models. However, models of high genus can be problematic. Such models may require long cuts to open up all the topological handles. In that case, much of the surface lies near the cut boundary, making it difficult to create a parametrization without significant geometric stretch and poor resampling. Figure 5 shows examples of trouble areas in the remeshing of the Buddha model. Our genus-6 Buddha model was obtained by filtering out tiny topological handles from a genus-104 scanned model [23]; working directly on the genus-104 surface would have been impossible.

In general, remeshing techniques can have difficulty capturing sharp surface features accurately at low sampling rates. In semi-regular remeshing, one technique to improve accuracy is to make the chart boundaries correspond with the most significant features, so that the subdivided domain edges follow these features [13]. Another technique is feature-sensitive remeshing [22], which warps the parametrization as a post-process to align the remesh edges with the sharp surface features. When creating our geometry images, adding a pass of feature-sensitive remeshing could improve reconstruction results for meshes with sharp geometry.

Since we used off-the-shelf compression code, we did not explore the extra savings that could be obtained using local-frame detail representation [10]. Adding this to our system may improve compression efficiencies.

## ACKNOWLEDGEMENTS

We gratefully thank Zoë Wood for providing the topologically simplified dragon and Buddha, Pedro Sander for the parametrization code, and Stanford University and Cyberware for models.

## REFERENCES

[1] DAVIS, G. Wavelet Image Compression Construction Kit. *http://www.geoffdavis.net/dartmouth/wavelet/wavelet.html*.

[2] DEY, T. K., AND SCHIPPER, H. A new technique to compute polygonal schema for 2-manifolds with application to null-homotopy detection. *Discrete and Computational Geometry 14* (1995), 93–110.

[3] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. In *SIGGRAPH 95*, pp. 173–182.

[4] ERICKSON, J., AND HAR-PELED, S. Cutting a surface into a disk. *ACM SoCG 2002*.

[5] FERGUSON, H., ROCKWOOD, A., AND COX, J. Topological design of sculptured surfaces. In *SIGGRAPH 92*, pp. 149–156.

[6] FLOATER, M. Parametrization and smooth approximation of surface triangulations. *CAGD 14*, 3 (1997), 231–250.

[7] GUSKOV, I., VIDIMCE, K., SWELDENS, W., AND SCHRÖDER, P. Normal Meshes. In *SIGGRAPH 2000*, pp. 95–102.

[8] HAKER, S., ANGENENT, S., TANNENBAUM, A., KIKINIS, R., SAPIRO, G., AND HALLE, M. Conformal Surface Parameterization for Texture Mapping. *IEEE TVCG 6*, 2 (2000), 181–189.

[9] HOPPE, H. Progressive Meshes. In *SIGGRAPH 96*, pp. 99–108.

[10] KHODAKOVSKY, A., SCHRÖDER, P., AND SWELDENS, W. Progressive Geometry Compression. In *SIGGRAPH 2000*, pp. 271–278.

[11] LAZARUS, F., POCCHIOLA, M., VEGTER, G., AND VERROUST, A. Computing a Canonical Polygonal Schema of an Orientable Triangulated Surface. In *ACM SoCG 2001*, pp. 80–89.

[12] LEE, A., MORETON, H., AND HOPPE, H. Displaced Subdivision Surfaces. In *SIGGRAPH 2000*, pp. 85–94.

[13] LEE, A., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *SIGGRAPH 98*, pp. 95–104.

[14] LOUNSBERY, M., DEROSE, T., AND WARREN, J. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *ACM TOG 16*, 1 (January 1997), 34–73.

[15] MUNKRES, J. *Topology*. Prentice Hall, 2000.

[16] PIPONI, D., AND BORSHUKOV, G. D. Seamless Texture Mapping of Subdivision Surfaces by Model Pelting and Texture Blending. In *SIGGRAPH 2000*, pp. 471–478.

[17] SANDER, P., GORTLER, S., SNYDER, J., AND HOPPE, H. Signal-Specialized Parametrization. *Microsoft Research MSR-TR-2002-27* (January 2002).

[18] SANDER, P., SNYDER, J., GORTLER, S., AND HOPPE, H. Texture Mapping Progressive Meshes. In *SIGGRAPH 2001*, pp. 409–416.

[19] SHEFFER, A. Spanning Tree Seams for Reducing Parameterization Distortion of Triangulated Surfaces. *Shape Modelling International* (2002).

[20] TAUBIN, G., AND ROSSIGNAC, J. Geometric compression through topological surgery. *ACM TOG 17*, 2 (1998), 84–115.

[21] VEGTER, G., AND YAP, C. K. Computational complexity of combinatorial surfaces. In *ACM SoCG 1990*, pp. 102–111.

[22] VORSATZ, J., RÖSSL, C., KOBBELT, L., AND SEIDEL, H.-P. Feature Sensitive Remeshing. *Computer Graphics Forum 20*, 3 (2001), 393–401.

[23] WOOD, Z., HOPPE, H., DESBRUN, M., AND SCHRÖDER, P. Isosurface topology simplification. *Microsoft Research MSR-TR-2002-28* (January 2002).

[24] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interactive multiresolution mesh editing. In *SIGGRAPH 97*, pp. 259–268.

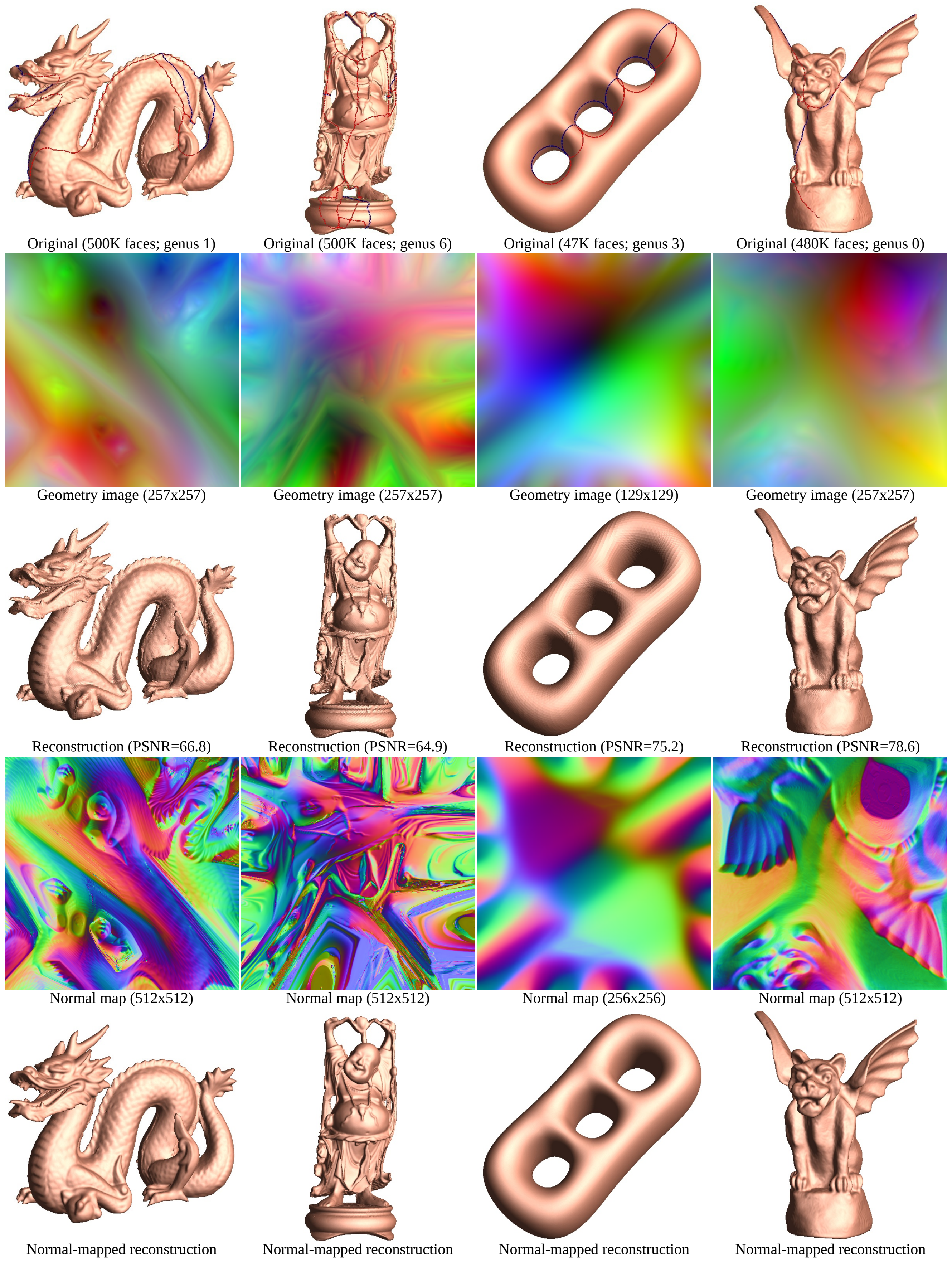

Original (500K faces; genus 1)
Original (500K faces; genus 6)
Original (47K faces; genus 3)
Original (480K faces; genus 0)
Geometry image (257x257)
Geometry image (257x257)
Geometry image (129x129)
Geometry image (257x257)
Reconstruction (PSNR=66.8)
Reconstruction (PSNR=64.9)
Reconstruction (PSNR=75.2)
Reconstruction (PSNR=78.6)
Normal map (512x512)
Normal map (512x512)
Normal map (256x256)
Normal map (512x512)
Normal-mapped reconstruction
Normal-mapped reconstruction
Normal-mapped reconstruction
Normal-mapped reconstruction

Figure 7: Examples: original meshes with cut, geometry images and their reconstructions, and use of normal-mapping.