Fast, Automatic, Topology-Preserving Gradient Mesh Generation for Image Vectorization

Yu-Kun Lai
Tsinghua University, Beijing

Shi-Min Hu
Tsinghua University, Beijing

Ralph R. Martin
Cardiff University, Wales

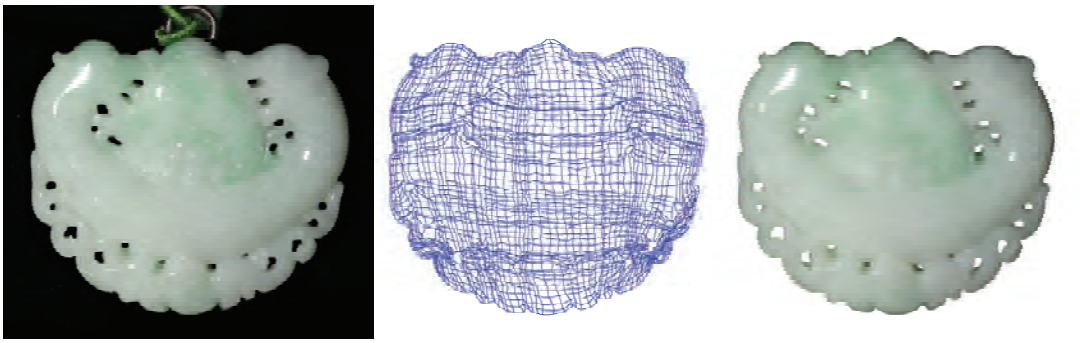

Figure 1: *Vectorization of an amulet with* 21 *holes, using a single topology-preserving gradient mesh.*

Abstract

Gradient mesh vector graphics representation, used in commercial software, is a regular grid with specified position and color, and their gradients, at each grid point. Gradient meshes can compactly represent smoothly changing data, and are typically used for single objects. This paper advances the state of the art for gradient meshes in several significant ways. Firstly, we introduce a *topology-preserving* gradient mesh representation which allows an arbitrary number of *holes*. This is important, as objects in images often have holes, either due to occlusion, or their 3D structure. Secondly, our algorithm uses the concept of image manifolds, adapting surface parameterization and fitting techniques to generate the gradient mesh in a *fully automatic* manner. Existing gradient-mesh algorithms require manual interaction to guide grid construction, and to cut objects with holes into disk-like regions. Our new algorithm is empirically at least 10 times *faster* than previous approaches. Furthermore, image segmentation can be used with our new algorithm to provide automatic gradient mesh generation for a *whole image*. Finally, fitting errors can be simply controlled to balance quality with storage.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation— ; I.3.4 [Computer Graphics]: Graphics Utilities—

Keywords: image vectorization, gradient mesh, image manifold, parameterization

1 Introduction

Vector graphics are well-known to offer various advantages over raster graphics. Many images, ranging from artistic work to captured photos, have relatively uniform or smoothly changing colors, and representing them using vector graphics can lead to significant savings of storage and network bandwidth. Vector graphics provide an additional advantage when images are to be displayed with significantly varying resolutions, on devices such as cell phones or high-definition TVs: they are resolution independent. Artifacts like blurring can appear if raster graphics are used. Because of these and other editability advantages, animation transmitted over the Internet usually uses vector graphics, in forms such as SVG and Shockwave Flash. Computer-aided automatic generation of vector graphics is thus highly desirable.

Recently, both academic and commercial vector graphics representations have been devised. The simplest vector graphics primitives comprise points, lines, curves, and polygons, with associated properties such as color. However, generating a vectorized representation from such simple geometric primitives is a non-trivial task, requiring significant artistic skill. As an alternative, *gradient meshes* are provided by commercial software like Adobe Illustrator and Corel CorelDraw as a way to describe compact vector graphics. Here, image elements are represented by one or more planar quad meshes, each forming a regularly connected grid. The position and color, and gradients of these quantities, are specified at each mesh grid vertex. The image represented by a gradient mesh is computed by bicubic interpolation of this specified information. Gradient meshes are a compact representation and especially suited to objects with smoothly varying colors.

To create a gradient mesh manually requires skill and is labor intensive. Sun et al.[2007] recently described a semi-automatic nonlinear optimization process for fitting a gradient mesh to a given image region. The algorithm requires user assistance both to segment the image into regions, and to construct the mesh—the user should choose the grid corners and usually the spacings of the grid lines for each region. The algorithm treats each image region as a topological rectangle, and cannot directly deal with regions containing holes, a problem the authors themselves recognize, which makes automatic computation difficult. Image inpainting can be used to fill holes, or further segmentation can be used to avoid the holes, but both require artistic skill and intuition for good results.

Here, we introduce a new *topology-preserving* gradient mesh representation allowing an arbitrary number of holes. We also give a novel *fully automatic* algorithm to generate a topology-preserving

gradient mesh. We take as input a raster image which has been matted or segmented into regions; the output is a set of gradient meshes representing an image close to the input. Following the concept of an image manifold [Kimmel et al. 2000], we treat the position and color gradient at each pixel of an input image region as a vector in a high-dimensional space: the image region can be thought of as a surface. We can now adapt well-established geometry processing techniques to gradient mesh generation. We use an adapted parameterization technique to map an arbitrary image region (with or without holes) to a rectangular region. By making the gradient mesh follow the parameterization, we obtain a mesh which is well suited to representing the original image region. Fig. 1 gives an example of representing an image region with 21 holes using a single topology-preserving gradient mesh.

Unlike Sun's highly nonlinear optimization method, the core of our algorithm simply solves sparse linear systems. Non-linear optimization is not only time-consuming, but also sensitive to the user-chosen grid corners and spacings. The main contributions of this paper are:

- A new representation for *topology-preserving* gradient meshes which can represent an arbitrary image region using a single mesh, with or without *holes*.
- A *fully automatic* algorithm for producing such meshes, which is significantly *faster* than previous vectorization methods. It is also easier to implement and more robust as only *linear* operations are performed in the core algorithm, and no user input is required. Fitting errors can be interactively adjusted to trade-off image quality with storage.

Our algorithm gains its strength from a novel approach to image vectorization based on image manifolds and geometric parameterization. A demonstration is further given that, if combined with a segmentation algorithm, our algorithm is suitable for *whole image* conversion to gradient mesh representation.

In Sec. 2, we review prior image vectorization work. Our topology-preserving gradient mesh representation is discussed in Sec. 3. Our algorithm for generating it is detailed in Sec. 4. Experimental results are given in Sec. 5, and concluding remarks in Sec. 6.

2 Related Work

2.1 Image vectorization

Prior work has considered vectorizing particular kinds of images. For example, Dori et al. [1999] proposed a sparse pixel vectorization algorithm for line drawings. Vectorization of cartoon animations has also been considered [Zou and Yan 2001; Zhang et al. 2009]. Such specialized images usually contain feature lines and relatively uniform color distribution. The corresponding algorithms usually focus on extraction of feature lines and corners to determine the geometric primitives for a vector representation.

Other work has considered vectorizing photographic images. Both commercial (Adobe Live Trace, Corel CorelTrace) and open-source (AutoTrace) tools provide such functionality. However, they work best when vectorizing images with relatively uniform colors; for more complicated images, the resulting vector graphics contain many small pieces, making them unsuited for further editing and costly to store or transmit.

Various representations and algorithms have been proposed in the research community for image vectorization. For smoothly varying photographic images, regular or irregular meshes are widely used.

Triangular meshes with assigned color distributions form one class of representation. Swaminarayan et al. [2006] first extract an edge contour set, then compute a constrained Delaunay triangulation to obtain a triangle-based vectorization. Demaret et al. [2006] use linear splines over an adaptive triangulation to represent images. The optimal linear spline minimizes the mean square color distances to the image. By using a triangulation, such methods can deal with general images. However, many irregular triangles typically result, and so the results are not compact. Lecot et al's 'ARDECO' uses automatic region detection and conversion with centroidal Voronoi regions to approximate the image [Lecot and Levy 2006]. Although the computation is both efficient (if trixels are used) and automatic, again usually too many output polygons result.

Quad meshes may also be used as primitives, and are usually applied in a per-object manner. In object-based vectorization [Price and Barrett 2006], the whole image is first segmented into a few objects. The color of each object is approximated by a regular mesh of Bézier patches. Areas with too large a fitting error are then subdivided, leading to a subdivision mesh representation with controlled fitting error. However, regions with rapidly changing colors lead to many tiny quads. Sun [2007] devised a semi-automatic algorithm to generate gradient meshes as the representation. As positions and pixel colors vary according to the specified gradients, such meshes contain rather fewer quads and have the advantage of being regular. However, his method involves non-linear optimization, which is slow and sensitive to the initial conditions—careful manual setup of the boundary is needed.

Diffusion curves have also been proposed as a vector graphics representation for smoothly-shaded images [Orzan et al. 2008]. Feature edges are automatically or manually selected, with differing color on each side. Images are reconstructed by a diffusion process requiring solution of a Poisson equation. The method is particularly suited to vector graphics with sharp edges. By using more edges, they can also represent certain smoothly varying regions, but that is not really their strength.

Adding feature edges as vector primitives to raster images can provide an augmented representation [Tumblin and Choudhury 2004]; object-based image editing [Barrett and Cheney 2002] represents images with texture-mapped irregular triangle meshes. Such methods do not really produce vector graphics, since the image cannot be directly reconstructed from feature information. Nevertheless, they may provide raster images with some of the advantages of vector graphics such as higher quality resizing, and maybe easier editing.

Our method generalizes traditional gradient meshes, to allow an image region to have *arbitrary topology* (i.e. zero or more holes). Our method can efficiently and automatically convert an image region to such topology-preserving gradient mesh representation, with error control. Thus, our method provides an effective solution to vectorizing gradually changing image objects, allowing efficient storage and transmission, and even whole images if used with a suitable segmentation algorithm.

2.2 Parameterization

Our pipeline relies on parameterization to set up a one-to-one mapping from a manifold surface to a canonical planar domain. For a full review of parameterization techniques, the reader is referred to [Floater and Hormann 2005]. Our method makes use of two particular kinds of parameterization: (quasi-)conformal parameterization, which locally preserves angles, and stretch minimization parameterization, which locally preserves area ratios. For general shapes which may have holes, we modify the original conformal slit maps [Yin et al. 2008] procedure to map the outer boundary of a region to a rectangle, and inner (hole) boundaries to slits. For regions without holes, quasi-conformal mean-value parameterization [Floater 2003] is used instead for better performance. Stretch minimization parameterization [Sander et al. 2001; Yoshizawa et al.

2004] can improve the distribution of parameters given an initial valid parameterization. We use such a parametrization with a carefully chosen metric which takes into account color distribution.

3 Topology-Preserving Gradient Mesh Representation

We next briefly review the traditional gradient mesh representation and then give our generalization allowing holes. Following [Sun et al. 2007], we treat each quad of a gradient mesh as a Ferguson patch [Ferguson 1964] which stores the position and color, and their derivatives (with respect to mesh parameters u, v), at each vertex. The gradient mesh can be evaluated using $U = \begin{pmatrix}1 & u & u^2 & u^3\end{pmatrix}$, $V = \begin{pmatrix}1 & v & v^2 & v^3\end{pmatrix}$, $0 \le u \le 1, 0 \le v \le 1$, and:

$$F(u,v) = UCQ^f C^T V^T,$$

$$Q^f = \begin{bmatrix} m^0 & m^2 & m^0_v & m^2_v \\ m^1 & m^3 & m^1_v & m^3_v \\ m^0_u & m^2_u & m^0_{uv} & m^2_{uv} \\ m^1_u & m^3_u & m^1_{uv} & m^3_{uv} \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix}.$$

Superscripts $0, 1, 2, 3$ correspond to the four patch vertices (see Fig. 2(left)). m represents a component to be evaluated, either a spatial coordinate x or y, or color component r, g, or b. Second-order derivatives (m^*_{uv}) are usually ignored and assumed to be zero. Given the values and derivatives at the patch corners, positions and colors can be readily evaluated inside the patch using bicubic interpolation. A traditional gradient mesh is a *regular* grid of Ferguson patches. As vertices are shared by adjacent patches, patch curve boundaries have G^1 continuity and colors have C^1 continuity.

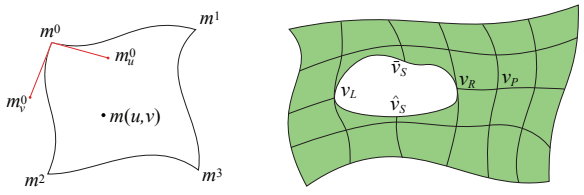

Figure 2: *Ferguson patch and topology-preserving gradient mesh.*

To allow holes (see Fig. 2(right)), we use the following construction, carefully designed to allow automatic gradient mesh generation by algorithm. To model a hole, we slice apart several consecutive horizontal edges within the grid. Vertices v_S within the cut sequence are split into two new vertices $\bar{v}_S$ and $\hat{v}_S$ and treated separately, except for the left and right end vertices of the cut, v_L and v_R respectively. Mesh vertices must meet the continuity requirements discussed above. v_L has a further constraint. The two Ferguson patches adjacent to the hole and sharing v_L have different u derivatives, $\partial\bar{m}_L/\partial u$ and $\partial\hat{m}_L/\partial u$. To ensure smoothness, these must satisfy $\partial\bar{m}_L/\partial u = -\partial\hat{m}_L/\partial u$. v_R must meet similar constraints. Multiple cuts can be introduced for image regions with multiple holes. Such *topology-preserving gradient meshes* allow an image region of arbitrary topology to be represented by a single mesh.

While this representation generalizes traditional gradient meshes, it is still compatible with current commercial software: a mesh in our representation with b holes can be converted into $b + 1$ traditional meshes without holes by cutting along the horizontal grid lines passing through slits. However, our representation is preferable, as it is more compact, and allows easier cut-and-paste operations, for example.

4 Algorithm

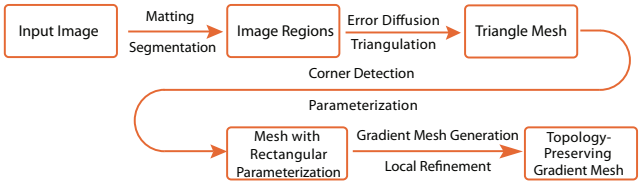

Figure 3: *Algorithm pipeline.*

We now give our algorithm for automatically generating topology-preserving gradient meshes, summarized in Fig. 3. The input is an image, and the goal is to output a set of gradient meshes which approximate it. If the whole image contains a single object as foreground, the object is first matted (e.g. using [Levin et al. 2008]) and then approximated directly by a gradient mesh. Otherwise, a segmentation algorithm is used to decompose the image into different pieces (objects). Each object in the image is treated as a 2-manifold surface patch embedded in some high-dimensional space, or *image manifold* [Kimmel et al. 2000], allowing us to utilize geometric tools. Each surface patch may contain one or more boundary loops. We map the outer loop to a rectangular planar domain in the boundary setup stage. We then map the patch interior into the domain, while reducing the inner loops to parallel slits. This parameterization serves as the basis for further stretch minimization reparameterization, with a metric carefully chosen to ensure regions with rapidly changing color gradient are naturally allowed a larger area. This allows a gradient mesh to be constructed using a *regular* grid in the parameter domain, while the final grid has spacing adapted to the rate of change of color variation in the image. If fitting errors need to be controlled, an iterative process may be used. Our method has certain similarities with [Lai et al. 2006], where feature sensitive parameterization is used for surface fitting.

4.1 Boundary setup

We now explain each step for a single image region; we will later consider decomposing an input image into regions. Every input image region has an outer loop, and may also contain one or more inner loops. The boundary setup stage only deals with the outer loop. The aim is to determine which four pixels on the outer loop should be mapped to the four corners of a rectangular domain. While corner pixels may best be chosen by a user, we give a method which determines them automatically, with satisfactory results.

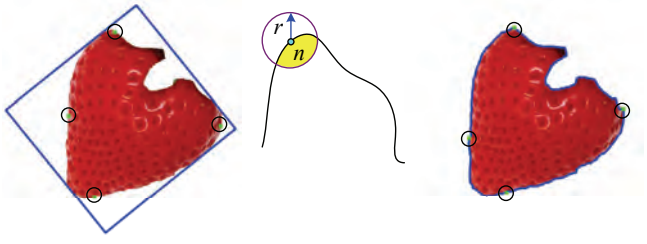

Figure 4: *Corner detection (a) bounding box and closest points to corners, (b) integral invariant computation, (c) detected corners.*

To determine the corner pixels, we first perform a principal component analysis of the region bounded by the outer loop, and rotate the image region to align its x and y axes with the principal directions, the shortest being x. An axis-aligned bounding box is constructed

around the image region, and the four pixels c_i ($i = 1 \ldots 4$) closest to the four corners of the bounding box are selected (see Fig. 4(a)). To determine a suitable pixel to be the i^{th} corner pixel, we consider boundary pixels within a distance d pixels from c_i, and select one having a denoised angle close to $\pi/2$. Setting d to $1/20$ of the region's perimeter works well in practice. As the boundary of an image region may be rather noisy, we use an integral invariant to estimate the angle at each boundary pixel [Manay et al. 2004]. At each such pixel, we place a disk of size r, and count the number n of pixels within the disk segment bounded by the outer loop (see Fig. 4(b)). For a perfect rectangular corner, $n \approx \pi r^2/4$. Using λ to balance the final corner pixel $\bar{c}_i$'s closeness to c_i and the desire for a rectangular corner, we set

$$\bar{c}_i = \arg\min_{\hat{c}_i} \left| n(\hat{c}_i) - \frac{1}{4}\pi r^2 \right| + \lambda ||\hat{c}_i - c_i||, \qquad (1)$$

In practice, we set $\lambda = 0.1$ and $r = 5$. The final corners tend to snap to some local corner-like points, as illustrated in Fig. 4(c).

It might improve the result if we considered corner locations and parameterization in an interlinked, iterative process; however, this would significantly increase the computational cost. This scheme works well in most cases, as our later examples show. A plausible alternative approach would be to locate all corner-like points, and then select four of them taking into account their spacing and corner angles. However, this may fail if the boundary is either too smooth or too rough, or give undesirable results in approximately symmetric cases. The image processing literature contains many corner detectors, and could be used to guide further investigation of this issue.

The four corners separate the outer loop into four segments. Let the segment between $\bar{c}_i$ and $\bar{c}_{i+1}$ be s_i (addition is assumed to be taken $\bmod 4$), and the number of pixels in this segment be $|s_i|$. The width and height of the rectangle in the parameter domain are set to (p_x, p_y) where $p_x = \max\{|s_1|, |s_3|\}$ and $p_y = \max\{|s_2|, |s_4|\}$ where corners are numbered anticlockwise from the lower left (this choice of x and y is arbitrary, and could be swapped).

4.2 Triangle mesh generation

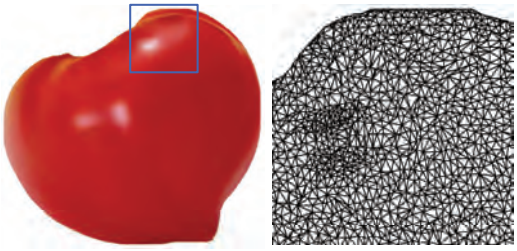

Figure 6: *An image region and detail of part of its triangulation.*

We now use a problem-specific image manifold which treats the image region as a surface. We first convert the input image region to a triangular mesh. This could be done simply by treating each pixel as a triangle vertex, joining four-connected neighbor, and dividing the resulting quads into two triangles. However, if the input image has relatively high resolution, such a mesh will be too large. Instead, inspired by the *trixel* concept in ARDECO [Lecot and Levy 2006], we compute a mesh according to local color consistency. For trixels, emphasis is placed on triangle shape quality, but this is less important in our setting as the triangles are just used as a basis for further processing. Instead, we first compute a saliency map using a simple compass filter with Sobel weights as in [Matusik et al. 2005], and then distribute a certain number of samples inside the region using error diffusion [Jarvis et al. 1976]. Saliency is used to weight sample placement, giving salient regions denser sampling. The total number of samples is set to be a fixed fraction (e.g. 10%) of the number of pixels. We then make a constrained Delaunay Triangulation (CDT) of the samples to obtain the triangular mesh M (see Fig. 6). The color at each vertex and its derivatives can be estimated from the underlying image. While the Sobel filter used to improve pixel sampling during mesh construction may not always capture low contrast features, for smooth regions, pixels are sampled more-or-less uniformly, which is sufficient for our purpose. The later parameterization step considers the rates of change of colors, and the sampling rate in this stage does not significantly affect the results.

During subsequent processing, the concept of a *metric* is fundamental to the method. In this discrete setting, the metric is simply an assignment of a length to each triangle edge. We use different metrics for different purposes, as explained later.

4.3 Topology-related mapping

We now consider how to construct a one-to-one mapping between the triangle mesh representation of the input image region and the 2D rectangular domain. The input image region may or may not contain holes. Slit map parameterization is an effective approach for a genus-zero open surface with multiple holes. The original slit map method takes the outer loop and one inner loop as a pair, with isolines between them. However, as shown in Fig. 5, we require isolines that go from s_1 to s_3 and from s_2 to s_4. In our approach, using the detected corners, we first construct a topological cylinder whose inner loops are hole loops of the input image region, and in which s_1 and s_3 are the end loops of the cylinder (see Fig. 5). This construction naturally leads to a *rectangular* parameterization domain which is appropriate for the gradient mesh. We first uniformly resample boundary s_2 or s_4 as necessary, so that both now contain p_y vertices. Since at this stage, we are only interested in topology, we choose a metric which assigns unit length to all triangle edges. This certainly satisfies the triangle inequality and is thus a valid metric. This simple metric is sufficient to build a topology-based mapping. Vertices on s_2 are merged with corresponding vertices on s_4, to form a topological cylinder M' containing holes. The modified shape M' contains $(2 + b)$ boundary loops: b inner loops b_i, and both s_1 and s_3. We compute a parallel slit map on this geometry to build a one-to-one mapping with the following steps. We construct a holomorphic one-form $\omega = \eta + \sqrt{-1}(*\eta)$ such that the imaginary part of its integral satisfies

$$\mathrm{Im}\left(\int_{b_i} \omega\right) = \int_{b_i} \eta = 0, \qquad \mathrm{Im}\left(\int_{s_1} \omega\right) = \int_{s_1} \eta = -2\pi,$$

where η is a harmonic one-form, $*$ is the Hodge star operator (which amounts to rotating the one-form by $\pi/2$) and Im takes the imaginary part of a complex number. This implies that

$$\mathrm{Im}\left(\int_{s_3} \omega\right) = \int_{s_3} \eta = 2\pi.$$

We can determine ω by taking a linear combination of the corresponding harmonic bases. The latter can be determined by solving $2(b+1)$ Laplacian equations with appropriate boundary conditions, which entails the solution of sparse linear systems. Finding the appropriate linear combination of the harmonic bases which satisfies the above equations just requires the solution of a small linear system (see [Yin et al. 2008] for further details). This gives a one-form ω defined on M', i.e. a value on each edge of M'. We may treat ω as being defined on the original mesh M instead of M' since edges between M and M' are in one-to-one correspondence. Given the

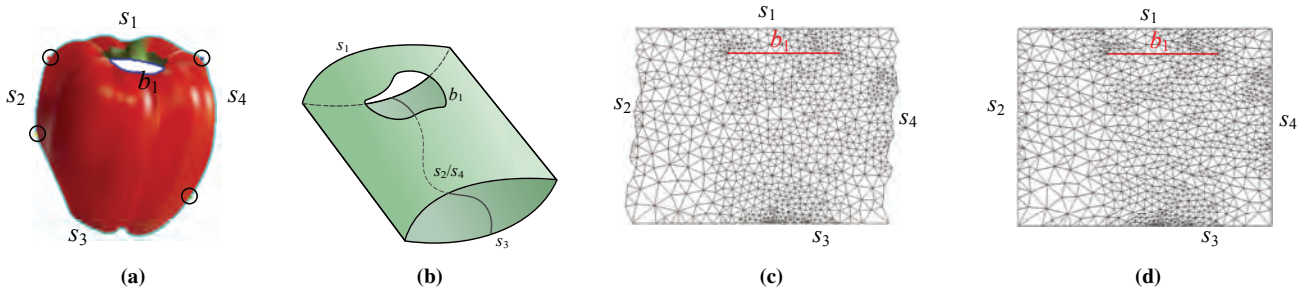

Figure 5: *Topology-related mapping for an image region with a hole: (a) input image region with hole, (b) corresponding topological cylinder with hole, (c) quasi-rectangle in the parameter domain, with slit, (d) corrected rectangle with slit in the parameter domain. The triangulations in (c) and (d) have been reduced to make them clearer.*

parameter of any vertex v_0, the parameter $p(v)$ for any other vertex v can be computed by computing $\int_{\gamma(v)} \omega$ where $\gamma(v)$ is any path connecting v_0 and v.

The constructed parameter domain is a quasi-rectangle (see Fig. 5(c)), whose left and right boundaries differ only by a translation. It can easily be corrected to form a rectangle (see Fig. 5(d)), and this process is certainly one-to-one. We finally uniformly scale the rectangle so that its aspect ratio is restored to p_x/p_y. Note that, after this mapping, each hole is mapped to a slit parallel to the x coordinate in the parameter domain, as desired for gradient mesh generation.

4.4 Feature-related mapping

After the above mapping, the mesh M corresponding to an image region has been mapped to a rectangular domain. However, the parameter for an individual vertex of M does not take into account local color information. Our aim is to use the gradient mesh to approximate the input image. It is desirable that after parameterization, a region which is difficult to represent by a single Ferguson patch should take up more area (so that the patch itself is smaller).

A Ferguson patch is ideal for representing a region with slowly varying color gradient. We thus perform reparameterization using a metric that takes into account how rapidly color gradient changes. Assume that $\mathbf{c}(x, y) = (r(x, y), g(x, y), b(x, y))$ are the red, green and blue components of the pixel at (x, y). We assign an 8-dimensional vector $\mathbf{f}(x, y) = (x, y, wd\mathbf{c}/dx, wd\mathbf{c}/dy)$ as the feature coordinate of each vertex, where w is a specified weight. Position is included in this definition as well as color gradient, because uniformly distributed grids are preferred in places where color gradients are similar. w is used to balance between lower fitting error (larger w) and smoother grid lines (smaller w). Experiments show that w can be varied over a wide range and yet still produce good results. We have used $w = 300$ (assuming color values in the range of 0–1) for most examples in the paper. For an adjacent vertex pair corresponding to two nearby pixels (x_1, y_1) and (x_2, y_2), distance is defined as $||\mathbf{f}(x_1, y_1) - \mathbf{f}(x_2, y_2)||_2$. Stretch minimization reparameterization [Sander et al. 2001; Yoshizawa et al. 2004] is now carried out using this feature-preserving metric. The fast algorithm given in [Yoshizawa et al. 2004] produces sufficiently accurate results for our purpose by solving several sparse linear systems. It takes as input a correct one-to-one mapping, which we have already obtained, and updates the parameters of the inner vertices to reduce stretch with respect to the above metric.

4.5 Gradient mesh generation

As this parameterization favors more rapidly varying regions of the input patch, we may construct the gradient mesh using a *regular* grid in the parameter domain: we have made the parameterization uniform in the sense of difficulty of representing the input by patches. To make the mesh we simply choose sets of u and v parameters. To ensure that the end points of slits and sharp corners on the boundary are accurately captured, we choose u and v values to include u and v parameters for all the slit end points, as well as the corresponding parameter values of the sharp corners on the boundary. We then further subdivide the u and v parameters as needed, in as uniform a manner as possible, to give a coarse grid.

We may further improve the result at low cost, using the observation that changing a control point in a gradient mesh only *locally* affects the output. We sort the vertices in descending order of average local fitting error, and successively adjust control points with relatively large fitting error by locally altering their positions within a small neighborhood. This local operation costs little, as the parameterization already provides a reasonable distribution. The greedy approach here always reduces the fitting error and is thus robust in practice. The geometric positions and derivatives of the control points of the gradient mesh can easily be obtained from the parameterization. The color at each control point can be calculated by interpolating colors at appropriate positions in the input image. The color derivatives can be estimated using monotonic cubic interpolation [Wolberg and Alfy 1999] and do not need to be stored. We avoid using a spatially adaptive grid here for two reasons: (i) even with holes, our method is compatible with the gradient mesh representation supported by major commercial software, (see Sec. 3), and (ii) simplicity; we wish to focus on the core ideas to make the algorithm and description easier to follow. This alternative could be explored in future.

The fitting error for this initial grid is evaluated. We then split that interval of either u or v which leads to greatest reduction in fitting error to give a finer grid. This process is repeated until the fitting error is below a user-specified threshold. Alternatively, a hierarchical sequence of grids of different fitting error may be computed, allowing the user to choose a trade-off between image quality and compact representation: a preview image, the fitting error and the file size can be interactively presented in a user-friendly interface.

4.6 Matting and segmentation

If the input to our algorithm contains only one foreground object of interest, we may extract the object using some efficient matting method [Levin et al. 2008]. Matting processes also give a reasonable estimation of foreground color when the object is merged with

the background. This is useful to ensure accurate sample colors are estimated for boundary control points.

If the input to our algorithm is a whole image, rather than a single object, it should be first segmented. Since each gradient mesh can handle color variations quite well, it is preferable to segment the input image into large, semantic pieces. Experimentally, we have found using large thresholds with a graph-based image segmentation method [Felzenszwalb and Huttenlocher 2004] produces an adequate coarse segmentation, which we then refine with GrabCut [Rother et al. 2004] to provide smoother boundary curves. Region growing is performed to fill in gaps between different segments, based on color similarity. Feeding each segment into the previously described algorithm leads to a gradient mesh representation of the input image. Since the boundary of each image region can only be approximately reconstructed, the reconstructed image may contain some gaps (usually of no more than one-pixel width). We may enlarge each region a little to avoid such gaps, or use partition-of-unity methods [Ohtake et al. 2003] to interpolate between samples from neighboring reconstructed regions.

5 Experimental Results

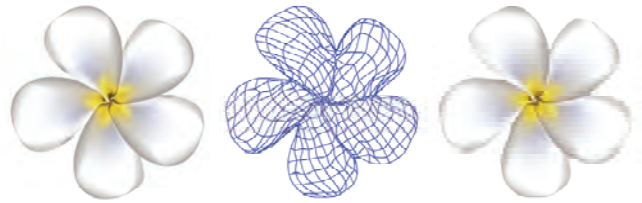

Figure 7: *An object without holes: original object, automatically generated single gradient mesh, and reconstructed object.*

Fig. 7 shows an example of generating a traditional gradient mesh using our method. The flower can be represented by a single gradient mesh, which is hard to model because of its complex geometry. Sun et al. rely on manually segmenting a similar example into 5 regions (their method appears to require regions of relatively simple shape). They require inpainting as well as user provided initialization. We obtain a good result fully *automatically*, using a *single* mesh.

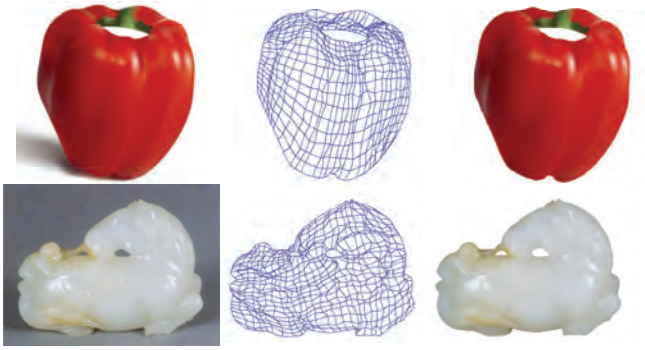

Figure 8: *Topology-preserving automatically generated gradient meshes for objects containing one or more holes.*

Figs. 1 and 8 show examples containing holes. The amulet in Fig. 1 contains 21 holes! It would be extremely difficult for a user to manually decompose and initialize such a shape, as would be required in Sun's approach. As it does not contain many feature edges, it may not be very suitable for representation using diffusion curves [Orzan et al. 2008]. We can fully automatically vectorize this object in about 2.5 minutes. The time taken for topology-based mapping is proportional to the number of holes, so this example takes significantly longer than other examples.

As a mostly linear approach, our method is significantly faster than Sun's non-linear optimization algorithm. Comparative experiments show that our method is empirically at least 10 times faster. Our experiments were carried out on an Intel Core2Duo 2.0GHz laptop with 2GB memory. Detailed performance is presented in Table 1. The timings for key steps including triangulation, parameterization and gradient mesh generation, as well as average fitting errors, are presented.

Table 1: *Experimental results: T: triangulation time, P: parametrization time, M: mesh generation time, E: average color error per pixel.*

Image (Figure)	T	P	M	E
Flower (Fig. 7)	8.1s	1.3s	5.8s	2.78
Pepper (Fig. 8)	8.2s	2.6s	6.2s	1.25
Jade (Fig. 8)	8.5s	2.8s	6.3s	2.39
Plumerias (Fig. 9 (c))	9.6s	8.2s	7.9s	2.83
Amulet (Fig. 1)	10.8s	125.0s	18.6s	2.76

Since Sun's method exactly optimizes the fitting error of pixels in the L_2 norm given sufficiently good initialization, our method may need more control points to achieve the same fitting error (as in the Pepper example, Fig. 8, which Sun however must manually cut into pieces). Much more significant, however, is that our method is much *faster*, and fully *automatic*. We can produce a good representation for any image regions with slowly varying detail (like the Pepper), whether or not containing holes. In cases where maximum data reduction is paramount, our method could be used to provide excellent automatic initialization for Sun's approach: better initialization generally leads to better convergence (both in time and probability of finding a global optimum). Used as a front-end to Sun's method, our algorithm would add very little extra computational effort, and also provide both topology preservation and automation. We also note that it is widely accepted (by e.g. the image compression community) that the L_2 norm does not adequately capture human perception of differences between images. Using perceptual measures of similarity [Wang et al. 2004] could improve results.

Fig. 9 presents an example of vectorizing the foreground flowers from a photograph. The two flowers overlap with a rather smooth transition, and reliably segmenting them into separate flowers is difficult. Our method generates an acceptable vectorization for the whole foreground (with a hole), using a relatively coarse grid. However, on close inspection of Fig. 9(c) it has some visible artifacts. Our method allows the user to interactively adjusting the fitting error using further subdivision (taking another 4s). By using a denser grid (going from 30×35 to 30×55), the fitting error is reduced to 2.15 per pixel and the reconstruction is much improved as shown in Fig. 9(d).

The complicated boundaries and / or holes in such examples make it challenging to even manually initialize such cases when using Sun's method. Our method, however, is fully automatic, and does not require any user assistance. Note that the grids of generated gradient meshes may be distorted, as illustrated by examples in the paper. This is mostly because those examples are relatively difficult to model (automatically) with a single gradient mesh: they have complicated boundaries and feature distribution. For a gradient mesh to be effective, it should follow the boundaries and / or

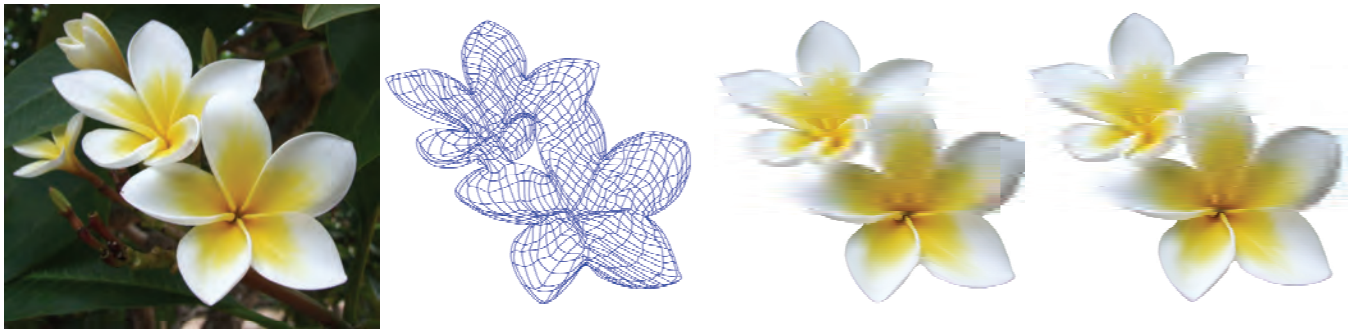

Figure 9: *Vectorization of plumerias with varying error control: (a) original, (b) automatically generated single coarse gradient mesh, (c) reconstruction from coarse mesh, (d) reconstruction from refined gradient mesh.*

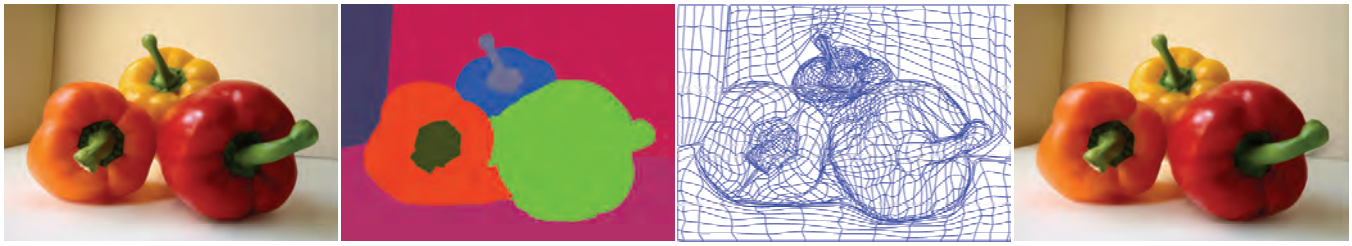

Figure 10: *Gradient meshes for a whole image: original image, automatically generated segmentation image—note how one region has a hole, automatically generated gradient meshes, final reconstructed image.*

features, making the grid less uniform.

Fig. 10 gives an example of automatically converting an *entire image* to topology-preserving gradient mesh representation. In some regions segmentation is not perfect but acceptable reconstruction is still obtained. The whole image is automatically segmented into 8 pieces, some containing holes. The local density is reasonably balanced between different segments, using error-based subdivision. The overall time for this example was about 2 minutes, and the average color error is 2.13 units per pixel. The gradient mesh representation takes up 9.4KB of storage after zip compression while a JPEG image of comparable quality requires 20KB. This indicates that our method is a plausible compression method for at least some kinds of images.

We have shown that our method can deal with image regions of arbitrary topology automatically and efficiently. One limitation for our gradient mesh generation (and probably others) is it may not be suitable to represent image regions with fine and rapidly changing details. For such cases, the fitting error will be relatively large, as illustrated in the durian example in Fig. 11.

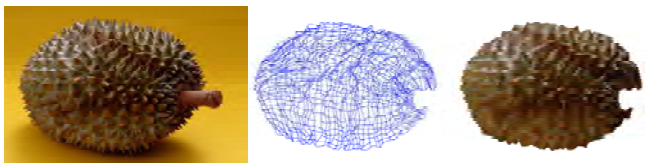

Figure 11: *A durian: original object, generated gradient mesh, and reconstructed object. Larger fitting error results for such objects with many fine details.*

6 Conclusions

We have introduced a new *topology-preserving* gradient mesh representation, which can handle objects with complex geometry and holes, and a novel *automatic* and *efficient* algorithm for generating such meshes from an input image. The results are general, compact, and compatible with commercial software

Treating an input image region as a particular type of image manifold allows geometric parameterization techniques to be applied to this image processing problem. Our method is more efficient than prior, less general gradient mesh techniques, mainly depending on the solution of a few sparse linear systems, instead of a non-linear system. No user guided initialization is required.

Because our algorithm has the advantages of being automatic and allowing holes, it can be used in conjunction with segmentation to automatically vectorize an entire image. Global control over fitting error is provided, allowing interactively choice between fitting quality and storage requirements.

Utilizing geometric parameterization in image processing is in itself an interesting novel tool, and we anticipate further potential applications to a wide range of image processing applications.

Acknowledgements

The authors would like to thank anonymous reviewers for their valuable comments. We would like to thank Kun-Peng Wang, Tao Chen, Ming-Ming Chen and Meng Ding for their kind help. This work was supported by the National Basic Research Project of China (Project Number 2006CB303106), the Natural Science Foundation of China (Project Number 60673004, U0735001), the Specialized Research Fund for the Doctoral Program of Higher

Education (Project Number 20060003057) and an EPSRC Travel Grant.

References

BARRETT, W., AND CHENEY, A. S. 2002. Object-based image editing. In *Proc. ACM SIGGRAPH*, 777–784.

DEMARET, L., DYN, N., AND ISKE, A. 2006. Image compression by linear splines over adaptive triangulations. *Signal Processing 86*, 7, 1604–1616.

DORI, D., AND LIU, W. 1999. Sparse pixel vectorization: an algorithm and its performance evaluation. *IEEE Trans. Pattern Analysis and Machine Intelligence 21*, 3, 202–215.

FELZENSZWALB, P. F., AND HUTTENLOCHER, D. P. 2004. Efficient graph-based image segmentation. *International Journal of Computer Vision 59*, 2, 167–181.

FERGUSON, J. 1964. Multivariable curve interpolation. *Journal of the ACM 11*, 2, 221–228.

FLOATER, M. S., AND HORMANN, K. 2005. Surface parameterization: a tutorial and survey. In *Advances in Multiresolution for Geometric Modeling*, IV: 157–186.

FLOATER, M. 2003. Mean value coordinates. *Computer Aided Geometric Design 20*, 1, 19–27.

JARVIS, J. F., JUDICE, C. N., AND NINKE, W. H. 1976. A survey of techniques for the display of continuous tone pictures on bilevel displays. *Computer Graphics and Image Processing 5*, 1, 13–40.

KIMMEL, R., MALLADI, R., AND SOCHEN, N. 2000. Image as embedded maps and minimal surfaces: movies, color, texture and volumetric medical images. *International Journal of Computer Vision 39*, 2, 111–129.

LAI, Y.-K., HU, S.-M., AND POTTMANN, H. 2006. Surface fitting based on a feature sensitive parameterization. *Computer-Aided Design 38*, 4, 800–807.

LECOT, G., AND LEVY, B. 2006. ARDECO: Automatic region detection and conversion. In *Proc. Eurographics Symposium on Rendering*, 349–360.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2008. A closed-form solution to natural image matting. *IEEE Trans. Pattern Analysis and Machine Intelligence 30*, 2, 228–242.

MANAY, S., HONG, B.-W., AND YEZZI, A. J. 2004. Integral invariant signatures. In *Proc. European Conference on Computer Vision*, 87–99.

MATUSIK, W., ZWICKER, M., AND DURAND, F. 2005. Texture design using a simplicial complex of morphable textures. In *Proc. ACM SIGGRAPH*, 787–794.

OHTAKE, Y., BELYAEV, A., ALEXA, M., TURK, G., AND SEIDEL, H.-P. 2003. Multi-level partition-of-unity implicits. In *Proc. ACM SIGGRAPH*, 463–470.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: a vector representaiton for smooth-shaded images. In *Proc. ACM SIGGRAPH*, 92:1–8.

PRICE, B., AND BARRETT, W. 2006. Object-based vectorization for interactive image editing. *The Visual Computer 22*, 9–11, 661–670.

ROTHER, C., KOLMOGOROV, V., AND BLAKE, A. 2004. "GrabCut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graphics 23*, 3, 309–314.

SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. 2001. Texture mapping progressive meshes. In *Proc. ACM SIGGRAPH*, 409–416.

SUN, J., LIANG, L., WEN, F., AND SHUM, H.-Y. 2007. Image vectorization using optimized gradient meshes. In *Proc. ACM SIGGRAPH*, 11:1–7.

SWAMINARAYAN, S., AND PRASAD, L. 2006. Rapid automated polygonal image decomposition. In *Proc. Applied Imagery and Pattern Recognition Workshop*, 28–33.

TUMBLIN, J., AND CHOUDHURY, P. 2004. Bixels: Picture samples with sharp embedded boundaries. In *Proc. Eurographics Symposium on Rendering*, 186–196.

WANG, Z., BOVIK, A. C., SHEIKH, H. R., AND SIMONCELLI, E. P. 2004. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Processing 13*, 4, 600–612.

WOLBERG, G., AND ALFY, I. 1999. Monotonic cubic spline interpolation. In *Proc. Computer Graphics International*, 188–195.

YIN, X., DAI, J., YAU, S.-T., AND GU, X. 2008. Slit map: conformal parameterization for multiply connected surfaces. In *Proc. Geometric Modeling and Processing*, 410–422.

YOSHIZAWA, S., BELYAEV, A., AND SEIDEL, H.-P. 2004. A fast and simple stretch-minimizing mesh parameterization. In *Proc. Shape Modeling International*, 200–208.

ZHANG, S.-H., CHEN, T., ZHANG, Y.-F., HU, S.-M., AND MARTIN, R. R. 2009. Vectorizing cartoon animations. *IEEE Trans. Visualization and Computer Graphics*, IEEE computer Society Digital Library, doi:10.1109/TVCG.2009.9.

ZOU, J. J., AND YAN, H. 2001. Cartoon image vectorization based on shape subdivision. In *Proc. Computer Graphics International*, 225–231.