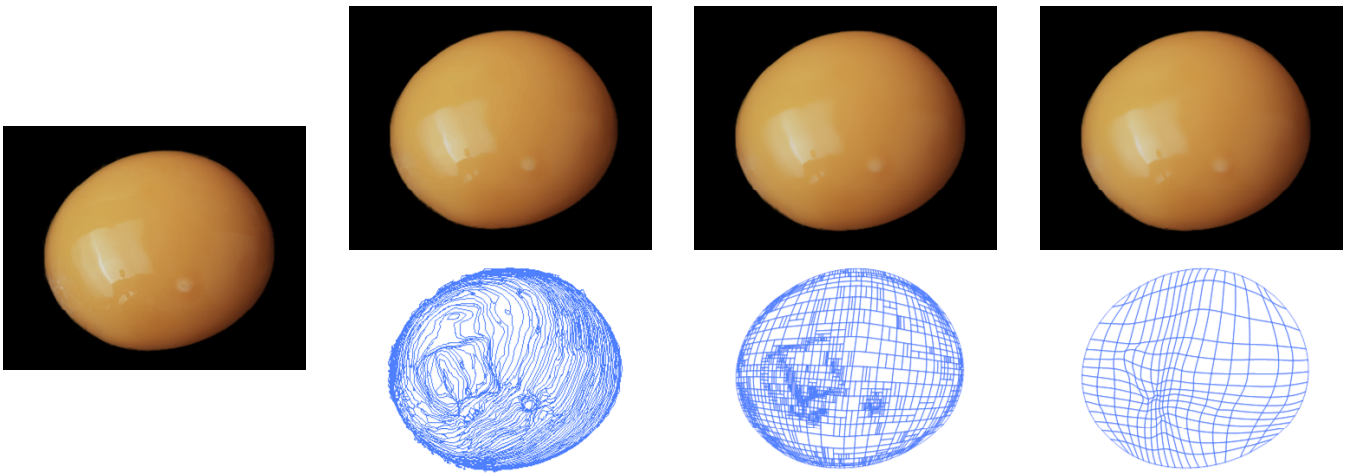
Image Vectorization using Optimized Gradient Meshes

Jian Sun Lin Liang Fang Wen Heung-Yeung Shum

Microsoft Research Asia



(a) input raster image (b) Adobe Live Trace result (c) subdivision mesh (d) optimized gradient mesh

Figure 1: *Three vector representations for a raw yolk. (a) input image. (b) result by Adobe "Live Trace", with 3,619 patches. (c) result by subdivision mesh, with 1,834 patches and 0.61/pixel reconstruction error. (d) gradient mesh optimized by our approach, with a* 18×15 *mesh and 0.6/pixel reconstruction error. The top row shows the reconstructed images. A simple gradient mesh can also faithfully reconstruct the highlights and reflectance on the yolk's surface.*

Abstract

Recently, gradient meshes have been introduced as a powerful vector graphics representation to draw multicolored mesh objects with smooth transitions. Using tools from Abode Illustrator and Corel CorelDraw, a user can manually create gradient meshes even for photo-realistic vector arts, which can be further edited, stylized and animated.

In this paper, we present an easy-to-use interactive tool, called *optimized gradient mesh*, to semi-automatically and quickly create gradient meshes from a raster image. We obtain the optimized gradient mesh by formulating an energy minimization problem. The user can also interactively specify a few vector lines to guide the mesh generation. The resulting optimized gradient mesh is an editable and scalable mesh that otherwise would have taken many hours for a user to manually create.

1 Introduction

The goal of image vectorization is to convert a raster image into a vector graphics that is compact, scalable, editable, and easy to animate. Today, it is more important because of the increasing requirements of vector-based contents (e.g., Flash or SVG) on the Internet, and in vector-based GUIs used in new operating systems such as Windows Vista.

Vector graphics are usually represented by points, lines, curves, polygons or regions since these representations can be manually drawn and editing using tools typical in vector graphics drawing systems. Often, these primitives only allow a gradual global color change to be applied along the paths of curves, or within the fill regions of polygons, in linear or radial forms. However, drawing a complex and multi-colored object with smooth transitions, as shown in Figure 1 (a), is difficult with simple primitives such as curves and polygons. Combining multiple simple primitives to create a coherent multicolored object is complicated and requires considerable artistic skill.

To address this problem, some vector graphics editors like Adobe Illustrator and Corel CorelDraw have recently introduced a drawing tool called "gradient mesh". With gradient meshes, artists can create a single multicolored mesh object on which colors can be assigned and color transitions can be manipulated to create a rich color object. A 2D graphics designer can now use the gradient mesh tool to manually create a vector graphic art with photo-realistic shading/shadow transitions. The results are painterly effects that look as if they had come from image editing tools - yet they are all in vector form. The designer can keep the art completely scalable and editable throughout the design process.

Gradient meshes are able to represent a large variety of image ob-

jects containing both smoothly shaded and rapid changes. Also, as shown in Figure 1, optimized gradient meshes have a much simpler structure and thus are easier for the user to edit than those created by other vectorization methods. However, it is often tedious and time-consuming to delineate an object with photo-realistic gradual transitions from a real image. It can easily take more than an hour by a professional artist using the gradient mesh tool to manually create a simple mesh like Figure 1(d).

In this paper, we provide a tool, called "optimized gradient mesh", to help a user to quickly convert an image object into gradient meshes. It only requires a small amount of user assistance for initialization. Figure 1 (d) shows an optimized gradient mesh created by our approach in less than a minute. Our tool works well with multicolored objects with smooth shading or shadow transitions, such as a face, a body, a toy, a piece of art, or any object with a smooth surface, in real images. The optimized gradient mesh is also able to handle a moderate amount of rapid color/intensity changes. For example, the highlights and complex reflectance on the yolk's surface are faithfully reconstructed in Figure 1 (d).

While the gradient mesh exists in commercial tools, we are the first to introduce it for image vectorization. We also present an effective optimization approach to fit gradient meshes to an image. Gradient meshes produced by our technique have several advantages: 1) *Efficiency of use*; the optimized gradient mesh makes it much faster for users to create gradient meshes from an input image. A skillful artist can save substantial labor and focus on re-creation, and a novice is able to create a single or a library of photo-realistic vector art for re-use. 2) *Easy to edit*; compared with other vectorization tools, the optimized gradient mesh can produce a simpler mesh that the user can further edit and animate. 3) *Scalability*; The gradient meshes can be scaled in size with fewer artifacts, as we will explain in Section 3.1. 4) *Compact representation*. The gradient mesh is an efficient representation for image objects with smooth transitions. For example, Figure 1 (d) shows a 7.7KB mesh, whereas the same quality JPEG image is 37.5KB. This advantage will be increased if the object is scaled in size.

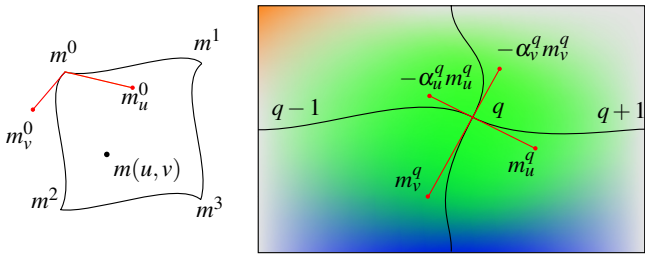
The above advantages arise from the powerful representation ability of the gradient mesh, which introduces geometric and color gradients on each vertex of the mesh and represents an image as a sparse set of points and gradients.

2 Previous Work

For scanned line art, black-white images, maps, and cartoon drawings, the main task of vectorization is to recognize and extract lines, regions, and text [Fan et al. 1995; Dori and Liu 1999; Zou and Yan 2001; Hilaire and Tombre 2006]. These approaches are mainly based on thresholding, thinning, contour tracing, and skeletonization algorithms in image processing. The extracted line, image contour, or region is represented by vector graphics primitives, e.g., curves and paths.

Recent commercial vectorization software, such as Adobe "Live Trace", Corel CorelTrace, and AutoTrace [AutoTrace 2004], are able to process photographic images, but are limited to cartoon-like flat shading images. For real images with smooth shading, these approaches typically output an over-segmented vector graphic consisting of a large number of irregular regions with flat colors. This kind of result is challenging to edit further. Figure 1 (b) shows a Live Trace result which contains thousands of small patches.

In RaveGrid [Swaminarayan and Prasad 2006], a constrained Delaunay triangulation of the edge contour set is performed to yield a polygon based vectorization. Adaptive triangulations [Demaret et al. 2006] approximate the image as a linear spline over an adapted



(a) Ferguson patch (b) gradient mesh

Figure 2: *The gradient mesh on the right consists of four Fergusion patches.*

triangulation. "ARDECO" [Lecot and Levy 2006] fits the image to a set of regions delimited by cubic splines. Each region is filled with a constant color, or a linear or circular gradient. Though these approaches faithfully approximate the image, the vectorization result is too dense to be edited.

The works in [Ramanarayanan et al. 2004; Tumblin and Choudhury 2004] embed vector-based primitives, such as boundaries, lines, or curves into the image to make the image scalable. Adjacent pixels may be separated by these primitives and are not mixed in the rendering. These approaches mainly focus on the sharp edges in the image.

To make the result editable, object-based image editing [Barrett and Cheney 2002] represents the image with an irregular, texture-mapped triangular mesh. The user can easily manipulate the image at an object level. Due to the use of texture mapping, the representation is not a scalable vector graphic.

In object-based vectorization [Price and Barrett 2006], an image object and its sub-objects are hierarchically segmented by a recursive graph cut algorithm. Each object or sub-object is represented by a regular mesh with Bezier patches. The mesh is further subdivided if the reconstruction error is above a threshold. Figure 1 (c) shows a subdivision mesh produced by this approach. There are many tiny patches in the rapidly changing regions (e.g., highlight and reflectance regions) that make further editing difficult. In this paper, we represent the image object as simple and easy-to-edit gradient meshes created by the proposed optimization approach.

3 Gradient Mesh

We introduce gradient mesh in this section. Note that our understanding of gradient mesh may be different from the actual implementations in Adobe Illustrator and Corel CorelDraw, which have not been made public.

Ferguson patch. A general parametric surface representation has the form $S = \{(x,y,z) : x = X(u,v), y = Y(u,v), z = Z(u,v)\}$, where u and v are the parametric coordinates. Usually, the surface is divided into topologically rectangular patches with mesh-lines. The Coons patch [Coons 1967] is probably the simplest type of parametric surface representation. The bounding mesh-lines of a patch are known and the interior points can be calculated with a blending function. A tensor product patch is defined as

$$m(u,v) = F(u)QF^T(v), \tag{1}$$

where $m(u,v)$ is the position vector of a point (u,v), the F vectors consist of the basis functions, and the Q matrix is a function of the control points of the surface patch. The tensor product representations include Bezier bicubic, rational biquadratic, parametric spline,

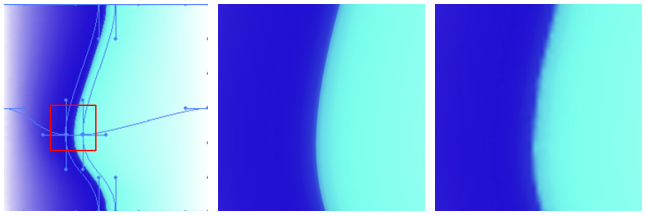


Figure 3: *Scalability. Left: a gradient mesh at original resolution (x 1). Middle: gradient mesh scaling result (x 8). Sharp edges are well preserved. Right: bi-cubic raster scaling result (x 8). Blocky artifacts appear.*

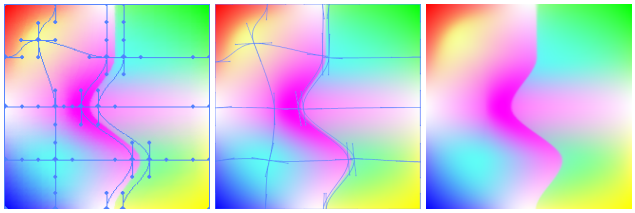


Figure 4: *Optimization. Left: a screen snapshot of a gradient mesh in Adobe Illustrator. We rasterize it as our input image. Middle and Right: optimized gradient mesh. The reconstruction error is 0.7/pixel.*

B-splines, and so on. Most of these representations are suitable for interactive surface design, but not for describing existing surfaces, since they require control points lying outside the surface. The Ferguson patch [Ferguson 1964; Farin 1997], however, is defined with control points lying on the surface. Because of this it is suitable for image vectorization purposes. The Ferguson patch is defined by

$$m(u,v) = F(u)QF^T(v) = UCQC^TV, \tag{2}$$

where

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

$$U = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix}, \text{ and } V = \begin{bmatrix} 1 & v & v^2 & v^3 \end{bmatrix}.$$

The m_u, m_v, m_{uv} are the partial derivatives. In practice, the values of m_{uv} are usually set to zero. Figure 2 (a) shows a Ferguson patch.

3.1 Gradient mesh

A gradient mesh consists of topologically planar rectangular Ferguson patches with mesh-lines. It has four boundaries and each boundary consists of one or more cubic Bezier splines. Figure 2 (b) is a gradient mesh. For each control point q in the mesh, three types of variables can be interactively edited: position, derivatives, and RGB color:

- the 2D position $\{x^q, y^q\}$ of point q.
- the derivatives $\{m^q_u, m^q_u, \alpha^q_u m^q_u, \alpha^q_u m^q_u\}$ are specified by four direction handles, as shown in Figure 2 (red lines). The direction handles can be dragged as in standard path tools. In order to achieve continuity between neighboring patches, m^q_u and $\alpha^q_u m^q_u$ (m^q_v and $\alpha^q_v m^q_v$) have the same direction but can be different in length, where α^q_u (α^q_v) is a scale variable. These four derivatives are shared with four adjacent Ferguson patches.
- the RGB color $c^q = \{c^q(r), c^q(g), c^q(b)\}$ is assigned by sampling a color from an input image or a color palette.

To render each colored Ferguson patch, color derivatives $\{c^q_u, c^q_v\}$ for each channel are also necessary. In the gradient mesh tool, these color derivatives are automatically estimated. Without knowing the implementation details in Adobe Illustrator or Corel CorelDraw, we have compared a number of methods to estimate the color derivatives from given colors on mesh points. We have found that a monotonic (1D) cubic spline interpolation [Wolberg and Alfy 1999] of colors along the mesh lines produces numerically similar rendering results compared with the rendering results by Adobe Illustrator. For the example in Figure 2 (b), two cubic splines are fitted along the mesh line $\{q-1, q, q+1\}$ using the monotonic cubic spline interpolation algorithm [Wolberg and Alfy 1999]. Then the color derivatives c^q_u, c^{q-1}_u and c^{q+1}_u are analytically calculated from the cubic splines. Each color channel is processed independently.

After getting the information (positions, geometric derivatives, colors, and color derivatives) of all control points, each Ferguson patch in the gradient mesh is rendered according to Equation (2):

$$f(u,v) = UCQ^fC^TV, \tag{3}$$

where Q^f contains color control variables. In Figure 2 (b), a smooth, multicolored image is generated by assigning the top-left point as orange-red, the middle-center point as green, the bottom-center point as blue, and the other points as light gray.

By representing an image object by a sparse set of points and gradients, gradient meshes are compact, scalable, and editable.

Scalability. Gradient meshes can be scaled with fewer artifacts than images. Sharp edges whin an image object can be preserved by placing two closely-spaced mesh lines on either side of the edge. Although a Ferguson patch is continuous in parametric coordinate space, the derivatives of image coordinates w.r.t. parametric coordinates (e.g., dx/du) can change from positive to negative or vice-versa. Therefore, when we render the image by incrementally sampling the parametric coordinates, we may visit the same image pixel twice and assign different colors, resulting in color discontinuities. Gradient mesh users often take advantage of this feature to draw an object with sharp edges (as shown in Figure 3).

Gradient mesh editing. Gradient meshes are easier to manipulate than images. The user can add, delete or edit mesh points and mesh lines. For each mesh point, its derivatives are manipulated by dragging four direction handles. With a selected mesh point, the user can choose a color from the color palette to define the color for that point. Each mesh point's direction handles and paths define how this point's color blends with other colors from other mesh points.

4 Optimized Gradient Mesh

In this section, we present our vector graphics creation tool: the optimized gradient mesh. We denote the gradient mesh as $M = \{Q_p, Q^f_p\}_{p=1}^P$, which consists of P patches. Given a smooth raster image object I, we expect to create or optimize a simple gradient mesh M with small reconstruction error. It is straightforward to minimize the following energy function:

$$E(M) = \sum_{p=1}^{P} \sum_{u,v} ||I_p(m(u,v)) - f_p(u,v)||^2 \tag{4}$$

where u, v are discrete parametric coordinates in each patch p and I_p is the image region in the patch p. This energy term is the reconstruction residual between the input image and the color graphics

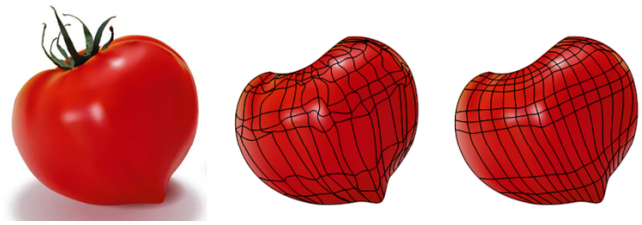


Figure 5: *Smoothness constraint. Left: input image. Middle and Right: optimized results without and with the smoothness constraint. The reconstruction errors are 1.8/pixel and 0.9/pixel.*

rendered by the gradient mesh using Equation (3). This fitting problem is similar to geometric modeling work such as [Schmitt et al. 1986; Krishnamurthy and Levoy 1996]. In this work, we fit a set of connected Ferguson patches to an image.

4.1 Optimization

Notice that minimizing $E(M)$ is a non-linear least squares (NLLS) problem, i.e., the energy can be written in the form $E = \sum a_k ||\mathbf{f}_k(\mathbf{z})||^2$, where $\mathbf{z}$ is the vector form of unknowns in M. For each control point, the associated unknowns are $\{x, y, m_u, m_v, \alpha_u, \alpha_v, c_u, c_v\}$. To enforce the differentiability of the rendered image, the scale variables α_u, α_v are shared with the color derivatives in all color channels. The color value c is a function of (x, y) and is sampled from the underlying image, i.e., $c = I(x, y)$. For object boundaries, the color is sampled from the estimated foreground colors by the coherent matting method [Shum et al. 2004] in a narrow band along the boundaries.

The Levenberg-Marquardt (LM) algorithm [Levenberg 1944] has proven to be the most successful solver for NLLS due to its use of an effective damping strategy that lends it the ability to converge promptly from a wide range of initial guesses. For more details of the LM algorithm, please refer to [Nocedal and Wright 1999]. LM requires the computation of a Jacobian $(\mathbf{J}_k)i = \frac{\partial \mathbf{f}_k(z)}{\partial z_i}$. Since $\mathbf{f}_k(\mathbf{z})$ contains an image observation, the derivative of the image is required and computed by convolution with a derivative of Gaussian filter. Notice that the Jacobian matrix is block-sparse because each $\mathbf{J}_k$ is only dependent on its neighboring control points. We use a standard LM implementation with sparse matrix support. In order to avoid local minima and make the solver robust, we build a Gaussian pyramid from the input image and apply a coarse-to-fine optimization for LM. The optimization time is about 10-30 seconds for a 10×10 mesh with our current unoptimized implementation. More sophisticated LM variants or GPU acceleration could be used for further speedup.

Figure 4 shows an example of an optimized gradient mesh. We first manually create a gradient mesh in Adobe Illustrator. It is then rasterized to an image as our input. The mesh is initialized by a 4×4 evenly divided mesh. Figure 4 (b) is the optimized gradient mesh after 40 LM iterations. The optimized mesh-lines are similar to the manually created mesh-lines, and the rendering results are virtually identical. Notice that both smooth regions and sharp edges are faithfully reconstructed.

Smoothness constraint. However, for complex examples or real images with noise, the optimization of energy function (4) often becomes stuck in a local minimum. The middle column of Figure 5 shows such an example. The user also expects a smooth optimized mesh since the manually created gradient meshes are usually

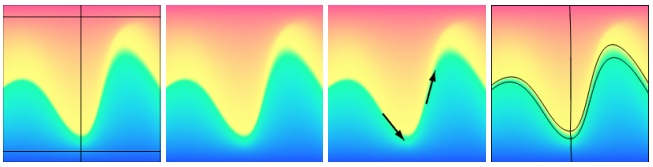


Figure 6: *From left to right: input image and initial mesh, directly optimized gradient mesh with reconstruction error of 2.5/pixel, two vector lines on u direction are specified by the user, optimized gradient mesh with vector lines as soft constraints with reconstruction error of 0.5/pixel.*

smooth. Hence, we add a smoothness term into the energy:

$$E'(M) = E(M) + \lambda \sum_{p=1}^{P} \sum_{s,t} \Big[||m(s-\Delta s,t) - 2m(s,t) + m(s+\Delta s,t)||^2 + ||m(s,t-\Delta t) - 2m(s,t) + m(s,t-\Delta t)||^2 \Big], \quad (5)$$

where s and t are discrete parametric coordinates. And Δs and Δt are sampling intervals which are set as 5 times the sampling intervals of u and v, respectively. The parameter λ balances the influence of the two terms, and the default value of λ is 50. Note that smoothness is also enforced between neighboring patches, i.e., $m_p(-\Delta s, t) = m_{p-1}(1-\Delta s, t)$, where p and $p-1$ are two adjacent patches in the u direction. The new energy term is the smoothness of the gradient mesh which minimizes the second-order finite difference. The last column of Figure 5 shows the optimized gradient mesh with the smoothness constraint. Smaller reconstruction errors are obtained by arriving at a better local minimum.

Boundary constraint. The boundary of a gradient mesh consists of four segments. Each segment is one or more cubic Bezier splines. We will describe the boundary initialization in section 4.3. In the optimization, we want to enforce a boundary constraint in which the control points on the boundary only move along the splines. For example, for the control point q on the spline S in the u direction, the associated unknowns are reduced to $\{t, m_v, c_u, c_v\}$, where t is the parametric coordinate of the control point on the spline. The scale α_v should be one, and variables m_u and α_u are a function of t. Supposing that the two adjacent control points of q are $q-1$ and $q+1$, we have $m_u = \frac{\partial S}{\partial t}(t_{q+1} - t_q)$ and $\alpha_u = (t_q - t_{q-1})/(t_{q+1} - t_q)$, where $\frac{\partial S}{\partial t}$ is the derivative of the spline S. The boundary constraint in the v direction is similarly enforced.

4.2 Vector line guided optimized gradient mesh

In most cases (all examples shown in the paper), our optimization can generate satisfactory results automatically. We also allow the user to draw a few vector lines in the image to control the mesh generation, e.g., the dominant directions of mesh-lines. Formally, if vector fields V_u and V_v along the u and v directions are specified, we optimize an energy $E''(M)$ by adding a new term:

$$E''(M) = E'(M) + \beta \sum_{p=1}^{P} \sum_{u,v} \Big[w_u(m(u,v)) \langle \frac{\partial m(u,v)}{\partial u}, \perp V_u(m(u,v)) \rangle^2 + w_v(m(u,v)) \langle \frac{\partial m(u,v)}{\partial v}, \perp V_v(m(u,v)) \rangle^2 \Big], \quad (6)$$

where $\langle \cdot \rangle$ is the dot-product operator, $\frac{\partial m(u,v)}{\partial u}$ is the derivative in the u direction, and $\perp V_u(m(u,v))$ is the unit normal vector of the V_u at location $m(u,v)$. $\frac{\partial m(u,v)}{\partial v}$ and $\perp V_v(m(u,v))$ are similarly defined.

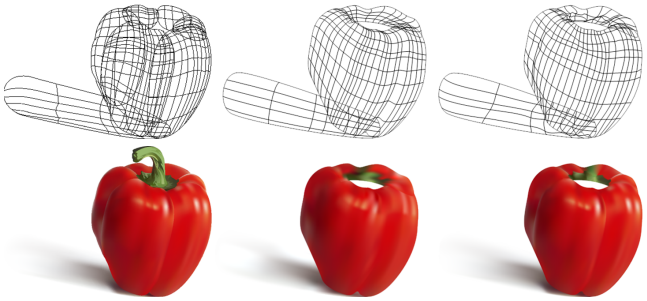


Figure 7: *Red pepper. Left: gradient meshes by an artist. Middle: initial gradient meshes. Right: optimized gradient meshes. After the optimization, the highlight and shadow regions are faithfully reconstructed.*

The default value of β is 20. The new energy term encourages consistency between optimized mesh-lines and specified vector fields.

The vector field is only computed in a narrow band along the vector line. The width of the narrow band is set as one fifth of the length of the vector line. The weight $w_u(m(u,v))$ in Equation (6) is the Gaussian falloff factor: $w_u(m(u,v)) = G(d|0,\sigma_v^2)$ where d is the distance from the location $m(u,v)$ to the nearest vector line. We set the standard deviation σ_v to one third of the width of the narrow band. Figure 6 compares the optimized gradient meshes with and without the vector lines as soft constraints.

4.3 Mesh initialization

As drawn by artists, a complex object usually consists of several semantic parts. We first decompose the input image object into several sub-objects using an interactive image cutout tool [Li et al. 2004] or a free lasso tool. Then, the boundary of each sub-object is manually divided into four segments. To obtain a good result, it is better to have a division so that the segments follow the major and minor axes of the object. Each segment is fitted by one or more cubic Bezier splines. Finally, the mesh-lines are initialized in two ways: evenly distributed or interactive placement.

Each sub-object with four segments is treated as a Coons patch, which supports multiple splines on one segment. To create an evenly distributed mesh, we simply divide the Coons patch evenly in parametric u and v coordinates. To interactively initialize a mesh, the user clicks a point p in the Coons patch, then we compute the parametric coordinates of p - (u_p, v_p) using subdivision. A mesh-line in the u direction can be added by fixing the v coordinate to v_p. Adding a mesh-line in the v direction or two mesh-lines in both directions can be done in a similar way.

The user can quickly create an initial mesh by simple mouse clicks. Placing more mesh-lines in rapidly changing regions and fewer mesh-lines in smooth regions usually produces a better result. But accurate mesh-line placement and editing are not required. The mesh initialization is very quick — 1-2 minutes on average. The initial mesh rendering result without optimization is also shown on the screen for instant feedback. The initial meshes of grapes (the third row) and face (the fourth row) are shown in Figure 8.

5 Experimental Results

We first show an optimized result from a vector object created by the gradient mesh tool in Adobe Illustrator, as shown in Figure 7. An artist drew this object using six meshes with 354 patches, and our optimized meshes consist of three meshes with 276 patches.

	Boundary Init. (mins)	Mesh Init. (mins)	Optimization (mins)
Fig.1 - egg	1	0.2	2.2
Fig.7 - pepper	3	1.5	2.8
Fig.8 - cup	2	0.5	1.9
Fig.8 - flower	3	1.5	2.4
Fig.8 - grapes	4	2.0	11.3
Fig.8 - sculpture	3	1.5	9.3
Fig.8 - face	4	2.5	10.3

Table 1: *Timings of examples in Figure 1, 7, and 8.*

To fit the shadow region on the ground, the occluded region is inpainted. In the rendering, supersampling is applied along the object boundaries to reduce alisasing artifacts. Then, we compare our approach with subdivision meshes [Price and Barrett 2006] in the top row of Figure 8. The subdivision meshes contain 1,973 small patches while our optimized gradient meshes consist of only 177 patches without sacrificing visual quality.

Figure 8 also shows the vectorization results for a flower, grapes, a sculpture, and a human face. In the flower example, five gradient meshes are individually optimized for the five petals which are segmented with overlap. The user spent 3 mins to specify boundaries and 1.5 min to initialize the meshes. The optimization takes 2.5 mins in total. The last two columns show editing results. We locally change the colors of a number of mesh points and deform a few mesh-lines to obtain naturally smooth transitions. In the next example, we separated the grapes into eight pieces. In the face example, the result consists of seven meshes - face(1), eyes(2), neck(1), right ear(1), and others(2). Note that the rendering results of the eyebrows, eyelashes, and hairs are smooth because these highly textured regions are beyond the capability of the gradient mesh. In the last example, the body of a sculpture is fitted with three overlapped gradient meshes. Table 1 shows the timings of boundary initialization, mesh initialization, and mesh optimization for seven examples in the paper.

With gradient meshes, the user can manipulate or control the smooth transitions of colors. Moreover, animation effects can be achieved by linear interpolation of meshes. Figure 9 shows three edited keyframes for an animation effect.

6 Conclusions

In this paper, we have introduced gradient mesh as an image representation and present an optimized gradient mesh tool for image vectorization. The output gradient meshes are simple, quick to create, and easy for editing and animation. It is a new application of geometric modeling tools in the image domain.

Limitations remains in our approach. A simple gradient mesh is insufficient to capture the fine image details and highly textured regions, as shown in Figure 10. Another difficult case is when the boundaries of the object are too complicated, or the object has complicated topologies, very thin structures, or many small holes. Despite these limitations, our optimized gradient mesh extends the range of "editable" vectorization to a variety of images.

Acknowledgements

We thank the anonymous reviewers for helping us to improve this paper, and Stephen Lin for his help in video production and proofreading. This work is inspired by Alex Hsu's talk at Microsoft Research Asia.

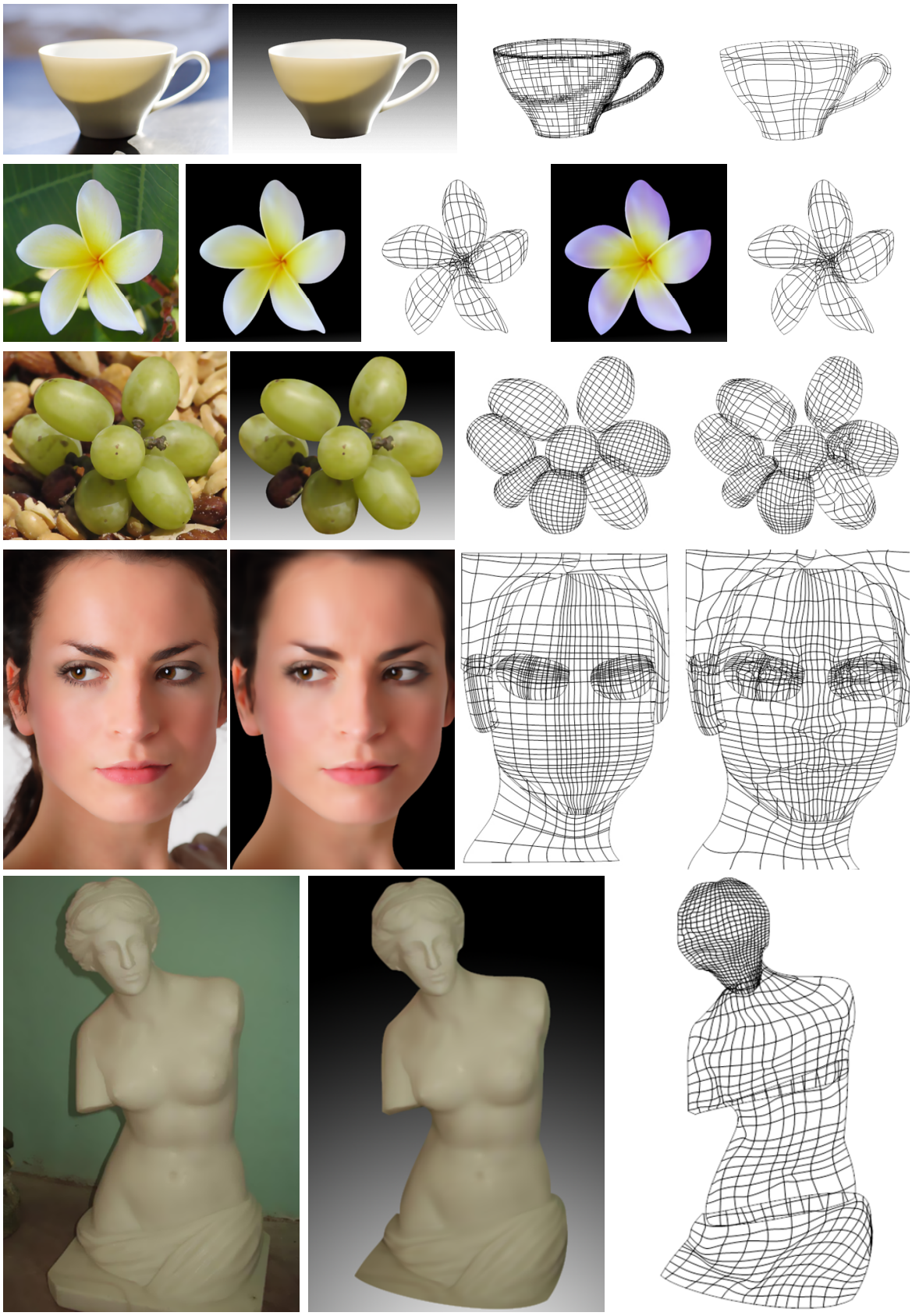


Figure 8: *More examples on real images. From left to right: input image, reconstruction result, and optimized gradient meshes. In the top row, the third column shows the subdivision mesh result. In the second row, the last two columns are the edited rendering result and mesh. In the third row, the third column is the initial mesh that is evenly distributed. In the fourth row, the third column is the interactively initialized mesh. From top to bottom: teacup (2 meshes, 177 patches), flower (5 meshes), grapes (8 meshes), face (7 meshes), and sculpture (4 meshes). The reconstruction errors of all examples are below 1.0/pixel.*

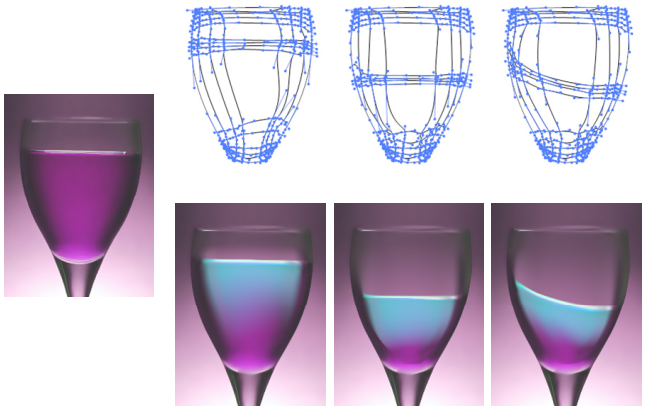


Figure 9: *Editing result. Left: input image. Right: three of four edited gradient meshes (keyframes) for a 8-sec animation. The colors of several control points are changed to create smooth transition effects.*

Figure 10: *From left to right: input image, reconstructed image, and optimized gradient meshes. Simple meshes can represent the low-frequency parts and large scale structures in the image, but fine details are lost .*

References

AUTOTRACE. 2004. http://autotrace.sourceforge.net/.

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

COONS, S. A. 1967. Surfaces for computer-aided design of space form. Tech. Rep. MAC-TR-41, Massachusetts Institute of Technology.

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. on PAMI 21*, 3, 202–215.

FAN, K.-C., CHEN, D.-F., AND WEN, M.-G. 1995. A new vectorization-based approach to the skeletonization of binary images. In *ICDAR*, 627–632.

FARIN, G. 1997. *Curves and surfaces for computer aided geometric design: a practical guide*. Academic Press, New York.

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

HILAIRE, X., AND TOMBRE, K. 2006. Robust and accurate vectorization of line drawings. *IEEE Trans. on PAMI 28*, 6, 890–904.

KRISHNAMURTHY, V., AND LEVOY, M. 1996. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH*, 313–324.

LECOT, G., AND LEVY, B. 2006. ARDECO: Automatic region detection and conversion. In *Eurographics Symposium on Rendering*, 1604–1616.

LEVENBERG, K. 1944. A method for the solution of certain problems in least squares. *Quart. Appl. Math.*, 2, 164–168.

LI, Y., SUN, J., TANG, C.-K., AND SHUM, H.-Y. 2004. Lazy snapping. *ACM Trans. Graph. 23*, 3, 303–308.

NOCEDAL, J., AND WRIGHT, S. J. 1999. *Numerical Optimization.* Springer.

PRICE, B., AND BARRETT, W. 2006. Object-based vectorization for interactive image editing. In *Visual Computer (Proceedings of Pacific Graphics)*, vol. 22, 661–670.

RAMANARAYANAN, G., BALA, K., AND WALTER, B. 2004. Feature-based textures. In *Eurographics Symposium on Rendering*, 186–196.

SCHMITT, F. J. M., BARSKY, B. A., AND HUI DU, W. 1986. An adaptive subdivision method for surface-fitting from sampled data. In *SIGGRAPH*, 179–188.

SHUM, H. Y., SUN, J., YAMAZAKI, S., LI, Y., AND TANG, C. K. 2004. Pop-up light field: An interactive image-based modeling and rendering system. *ACM Transaction of Graphics 23*, 2, 143–162.

SWAMINARAYAN, S., AND PRASAD, L. 2006. Rapid automated polygonal image decomposition. In *35th Applied Imagery and Patt. Reco. Workshop*, 28–33.

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

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

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