Image Vectorization and Object-based Editing System

Mark Dupplenthaler † ‡ Ken Tsutsuguchi † Akira Kojima †

† The University of British Columbia: 2329 West Mall, Vancouver, BC V6T 1Z4 Canada
‡ NTT Cyber Solutions Laboratories: 1-1, Hikari-No-Oka, Yokosuka, Kanagawa 239-0847 Japan
E-mail: {dupplenthaler.mark, tsutsuguchi.ken, kojima.akira} @lab.ntt.co.jp

Abstract: We describe our image vectorization and editing system. Using a technique based on Xia et al.[8], our vectorization takes a bitmap image and uses a triangular mesh to topologically represent it. The vectorization is fully automatic; in addition, the user can manually select relevant objects which can be individually vectorized. The color inside each patch is represented using thin plate spline surfaces. Each vectorized image is stored as XML and can be loaded on web browsers for displaying and editing. Patches can be tagged with names, moved or stretched, and the interior color can be changed at a more abstract level compared to editing individual pixels.

Keywords: image vectorization, image edit, Bézier triangular patch, thin plate spline, XML

1 Introduction

Vector graphics represent images using mathematical functions and geometrical shapes. In comparison to bitmap images, vector graphics are more resolution independent, smaller in file sizes, and are also easier to be edited geometrically. However, vector graphics have difficulties representing rapidly changing colors and textures because they cannot be fitted precisely by smoothly varying functions. Due to this constraint, vector graphics are mostly used for simple graphics such as fonts and cartoon style graphics which are frequently scaled.

Recently, various methods for vectorization of colored photographic images have been proposed. We have developed “Vectory”, which allows such photographic image vectorization. Also, a web browser based editing system is developed for the resulting vector images. The goal of the application is to create a text-based vector representation from a bitmap image that can be displayed and edited on a web browser. The vectorization algorithm is mostly based on triangular patch based technique by Xia et al.[8], and can be described in several steps. The first step in our algorithm is optional and is a manual image segmentation process done by the user using graph cuts[3] to locate objects in the image (Section 3.1). The next step is to initialize a pixel level triangular mesh for the pixels in the region to be vectorized (Section 3.2). Once initialized, the mesh is coarsened by edge contractions, which results in triangles that group regions of similar colors (Section 3.3). The colors on the inside of each of these patches are fit using thin plate spline surfaces (Section 3.4). The resulting vector image is outputted as an XML text file (Section 3.5) that could be loaded on our HTML page for editing. Editable features include patch tagging with object names, patch moving to geometrically distort regions, and patch color changing (Section 4).

2 Related Works

For photographic image vectorization, algorithms tend to group regions of similar colors using polygons or curves. The colors inside these regions are represented as surfaces that best fit it, or interpolated from the colors at the region’s boundaries.

One vectorization approach that gained popularity recently is the gradient mesh, which is represented as a mesh of rectangular tensor product patches. The colors and color derivatives at the corners of each tensor product patch are used to interpolate its interior colors. Sun et al.[5] proposes the optimized gradient mesh, which requires manual initialization of the region to be vectorized, but automatically fits a structurally simple mesh to it. Although their approach creates large and well-oriented tensor product patches, the vectorization is very sensitive to mesh initialization and also has difficulty for objects with holes in it. Lai et al.[7] solves some of these problems by introducing an automatic gradient mesh vectorization that also allows holes in objects. The image is automatically segmented, and gradient meshes are created for each object with proper orientations to align at object corners.

A similar approach to the gradient mesh is proposed by Price et al.[4] that uses a mesh of rectangular Bézier patches. Automatic object and subobject segmentation is done to create a hierarchy of objects in the image. Each of these objects are vectorized and patches with too much error are subdivided to increase the accuracy.

Instead of rectangular patches, Xia et al.[8] introduces a method that uses a mesh of Bézier triangles. Each Bézier patch is fitted with thin plate spline (TPS) surfaces to represent each color channel. This cubic surface is a more complex representation than interpolation using tensor product patches or Bézier patches. Although TPS requires more coefficients to store color data compared to simpler methods, larger patches can be used which is easier for image editing.

Diffusion curves[6] take a completely different ap-
approach from mesh based techniques. The image is described by curves which align with the color discontinuities of the image. Colors and blur values are attached to either side of this curve and are diffused spatially into the surrounding regions. Although diffusion curves are easy to edit and create images with, it is difficult to accurately represent colors at regions with no edges or at intersections of edges due to the nature of the Poisson diffusion equation.

3 Image Segmentation and Vectorization

This section describes the object initialization and vectorization algorithm for converting a bitmap image into vector format. Our vectorization method closely follows Xia's algorithm[8] with the major difference being that our algorithm prefers to use triangles as vector primitives instead of Bézier patches[2]. This is because the triangular Bézier patches of cubic order that Xia uses takes longer to render compared to triangles. Since our goal is to render vector images on web browsers, the rendering speed is more important than the accuracy increase that Bézier patches provide by precisely aligning to curved color discontinuities. Our algorithm only uses Bézier patches at the boundaries of objects where precise alignment matters the most.

3.1 Image Segmentation

In this optional step, the image can be manually segmented to select objects and its subobjects. Objects selected this way can be vectorized by itself, or the entire image could be vectorized and the patches that reside in each object could be labelled. To tag objects in the image, the user must place foreground and background seeds for use with graph cuts[3] to separate the object with the rest of the image. The weights of the edges used in graph cuts are based on the object selection algorithm used by Price et al.[4] and are as follows:

\[
\{p, T_f\} = \begin{cases} 
\infty, & \text{if } p \text{ is at a background seed} \\
0, & \text{if otherwise}
\end{cases}
\]

\[
\{p, q\} = \frac{1}{\|C(p) - C(q)\|^2 + 1},
\]

with \(p\) and \(q\) being neighboring pixels in the image, \(T_f\) being a foreground terminal, \(T_b\) being a background terminal, and \(C(p)\) being the RGB color of the pixel.

3.2 Mesh Initialization

The vectorization algorithm begins with creating a pixel level triangular mesh of the 2D image plane in the same manner as Xia et al.[8]. Image segmentation results are used as a mask to select relevant pixels of the image. The mesh structure is stored as an interconnected graph having vertices, edges, and faces. Each vertex holds the RGB color value at the corresponding pixel and the faces form planes that intersect at color values of its three corner vertices. Note that there are actually three meshes (one for each color channel) using the same 2D domain.

3.3 Mesh Coarsening

Once the pixel level triangular mesh is initialized, it is coarsened in the same manner as Xia et al.[8] by a mesh simplification algorithm based on vertex pair contractions[1]. The mesh coarsening algorithm groups similar color regions of the image, creating many small triangles for textured regions and larger triangles for regions with similar color. Even if the image is not properly segmented, mesh coarsening separates most faces at color discontinuities.

The mesh coarsening algorithm repeatedly selects the least cost edge and attempts to contract it if possible. The algorithm iterates these edge contractions until a specified number of faces are remaining. When an edge contracts, one of its endpoint vertices collapses into the other, and the faces adjacent to this contracted edge will be deleted as well as any overlapping edges after the contraction.

After mesh coarsening, our algorithm converts the faces containing Bézier curves (the object boundaries) into Bézier patches. The control points of the straight

---

Figure 1: Vectorization of 512x512 Lena using 2610 triangular patches: far left: original bitmap image, mid left: zoom up of pixel level triangle mesh, mid right: coarsened triangle mesh, right: reconstructed vector image
edges are placed uniformly between the endpoints so only the edges that were already curved stay curved. For the interior control point of the cubic Bézier patch, we just take the average of the midpoints of the three edges of the Bézier patch.

3.4 TPS Color Fitting

Thin plate splines (TPS) are cubic surfaces that are used to represent the interior colors of faces in the coarsened mesh. Since mesh coarsening creates faces bounding regions of similar color, each color channel is represented accurately by fitting a smooth cubic surface to it. The center coordinates of TPS bases are sampled uniformly in the parametric domain in the same way as Xia et al.[8]. To increase the accuracy of TPS fits, our algorithm allows an increase in the number of TPS basis functions or a four way subdivision of faces that have too much error. The TPS surface is described as follows:

\[ f(u, v) = \sum_{i=1}^{N} \alpha_i \phi(||(u_i, v_i) - (u, v)||) + b_0 + b_1 u + b_2 v \]  

Where \( N \) being the number of TPS basis functions, \( \sum_{i=1}^{N} \alpha_i = 0 \), \( \sum_{i=1}^{N} \alpha_i u_i = 0 \), \( \sum_{i=1}^{N} \alpha_i v_i = 0 \), and \( (u_i, v_i) \) being the center coordinates of TPS basis functions. The TPS radial basis function is \( \phi(s) = s^2 \log(s) \).

3.5 XML Output

The output of the vectorization algorithm is a XML file. The data written are the vertices and control points, as well as the TPS coefficients of every face. The faces that form the interior of objects selected during image segmentation are tagged correspondingly. Currently the file sizes are quite large as values are written in XML format with no compression (i.e. 512x512 image with 2610 faces is 1.2Mb).

4 Object Based Editing

The XML file outputted by our vectorization algorithm can be loaded and edited from web browsers (e.g. Google Chrome, Apple Safari). The browsers must have HTML5 Canvas support and preferably have fast JavaScript engines. The editing platform uses two images: a simple vector image and a complete vector image. The simple vector image fills each face with its average color and is used to update the user instantaneously. The complete vector image calculates colors from TPS surfaces and takes some time to render.

To render or manually select patches, pixel coordinates must be converted to parametric coordinates. To do this, bounding box tests are first performed for each face to quickly filter out distant pixels. For remaining pixels, point in face tests are performed. If the face is a triangle, barycentric coordinates are simply computed from pixel coordinates, which also suffices as a point in triangle test. For Bézier patches, we use the same method as Xia et al.[8] to calculate parametric coordinates of interior pixels in a patch. Bézier patches are recursively subdivided into four until all subdivided patches are smaller than a pixel. When the patches are this small, it is reasonable to approximate each subdivided patch as a triangle. We then calculate the barycentric coordinates of any pixels that are inside of these approximated triangles and the resulting coordinates are propagated back to the original Bézier patch. Upon rendering, the parametric coordinates obtained for pixels inside each face are then used to calculate colors from TPS surfaces.

Once patches are selected they can be edited (Figure 2). The editable features include patch names, control point locations, and patch colors. By allowing patches to be tagged with names, a reference for quick access to this image region is established. In addition to the regions already tagged from image segmentation during vectorization, object tags could be attached to newly selected patches. These tag names are saved as a part of the XML file, resulting in object data to be saved together with image colors in one file. Regions could also be distorted by disconnecting patches from the mesh or stretching and contracting patch control points. This allows editing of image features at a more abstract level compared to editing individual pixels, and local deformations of regions can be done with only a few mouse clicks. Additionally, we allow color changes to selected patches by changing TPS coefficients.

5 Results and Discussions

Our algorithm can vectorize most 512x512 resolution bitmaps in less than a minute on an Intel Core i7 2.93Ghz processor. If higher accuracy is required, the vectorization time and rendering time increase (Table 1).

As mentioned previously, we mainly use triangles as vector primitives rather than Bézier patches. Bézier patches can represent larger areas by proper alignment to region boundaries because of curved edges, and an equal level representation using straight edged triangles requires more faces. However, Bézier patches take much
longer to render compared to triangles (Table 1). Compared to a triangles of similar size, we have seen that Bézier patches take roughly 10 times longer to render. The cause of such long rendering time comes from the cubic nature of Bézier patches. While triangles can simply calculate a pixel’s barycentric coordinates relative to itself, Bézier patches need a root finding method to convert to parametric coordinates. Xia et al. [8] uses a GPU based algorithm to instantaneously render Bézier patch meshes, but our objective is to allow users to easily manipulate vector images on web browsers. For vector images to update interactively upon each edit, we sacrifice the accuracy that Bézier patches provide.

Our object level editing system is easy and intuitive. The user simply mouse drags over image regions to select patches. These patches can also be moved by mouse drags, resulting in interesting transformations caused by deformed TPS surfaces. However, the desired changes are not always possible as individual pixel colors cannot be changed. This also affects color changes because the entire patch is affected, sometimes resulting in unwanted regions in the patch to change color.

We could improve our algorithm in many ways. Currently we do not see clear magnifications as seen in Xia et al. [8] due to our triangle patches not aligning perfectly to color discontinuities. At regions with high color variations, many color defects can be seen upon zooming in. However, rendering speed is usually slower when accuracy is increased. Techniques that allow both accuracy and rendering speed to increase should be explored. Another area we can improve on is image segmentation. Currently, the user must manually select foreground and background seeds for graph cuts. If the relevant objects can automatically be determined, it will be much easier for the user to perform object level editing. The current file size of our output xml files are also too large and different file formats may need to be considered.

<table>
<thead>
<tr>
<th>Lena 512x512</th>
<th>Faces</th>
<th>Vectorization(sec)</th>
<th>Rendering(sec)</th>
<th>Mean Error</th>
<th>Mean Intensity Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Quality</td>
<td>1043</td>
<td>24</td>
<td>1</td>
<td>8.2</td>
<td>4.4</td>
</tr>
<tr>
<td>Medium Quality</td>
<td>2610</td>
<td>32</td>
<td>1</td>
<td>5.9</td>
<td>3.2</td>
</tr>
<tr>
<td>High Quality</td>
<td>5221</td>
<td>46</td>
<td>2</td>
<td>4.7</td>
<td>2.5</td>
</tr>
<tr>
<td>Only Bézier patches</td>
<td>2610</td>
<td>58</td>
<td>38</td>
<td>6.7</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Table 1: Table of render times with image quality on an Intel Core i7 2.93 Ghz processor. The mean error is $\sqrt{dR^2 + dG^2 + dB^2}$, and the mean intensity error is $\frac{|dR| + |dG| + |dB|}{3}$, averaged over all pixels.

6 Conclusions

We have introduced “Vector” which is an image vectorization software that is able to automatically vectorize photographic images. Manual segmentation allows vectorization and tagging of selected objects. The vectorization is done using a mesh of triangles, and Bézier patches are used at object boundaries for curved alignment to its boundaries. Additionally we have a web browser based editing system that can attach object names to patches, move patch control points, and change colors of patches. The editing is done by changing triangular faces, which lets users perform geometric deformations much easier compared to editing at a pixel level. Although we still have many areas we can improve on such as magnification accuracy and rendering speed, we manage to vectorize images with qualities almost identical to its original bitmap and perform simple edits on them easily.

References