# Image Super-Resolution by Vectorizing Edges

Chia-Jung Hung Chun-Kai Huang Bing-Yu Chen

National Taiwan University
{ffantasy1999, chinkyell}@cmlab.csie.ntu.edu.tw robin@ntu.edu.tw

**Abstract.** As the resolution of output device increases, the demand of high resolution contents has become more eagerly. Therefore, the image super-resolution algorithms become more important. In digital image, the edges in the image are related to human perception heavily. Because of this, most recent research topics tend to enhance the image edges to achieve better visual quality. In this paper, we propose an edge-preserving image super-resolution algorithm by vectorizing the image edges. We first parameterize the image edges to fit the edges' shapes, and then use these data as the constraint for image super-resolution. However, the color nearby the image edges is usually a combination of two different regions. The matting technique is utilized to solve this problem. Finally, we do the image super-resolution based on the edge shape, position, and nearby color information to compute a digital image with sharp edges.
**Keywords:** super-resolution, vectorization, matting, edge detection, Bézier curve, mean-value coordinate, interpolation.

## 1 Introduction

Image super-resolution is a task that scales a digital image up. The ability of scaling up a digital image is very important for many aspects. For example, there are more and more high-definition display devices but not all contents produced in such high resolution, so we have to scale these contents to fill up the whole display.

Super-resolution is a very ill-posed problem due to its nature. If we want to get a high resolution image with 2x large width and 2x large height, only 1/4 pixels of the target image can be obtained from the original image perfectly. Other 3/4 pixels cannot be determined uniquely. To regularize this problem, we have to make some assumptions.

The most commonly used assumption is that the image is locally smooth. According to this assumption, many interpolation based methods have been proposed. Three well-known interpolation methods are nearest neigobor, bilinear interpolation, and bicubic interpolation. However, these three methods would produce some unwanted artifacts as shown in Fig. 1 (b), such as the result image may be blurry and textureless and the edges in the result image may be jaggy or blocky. Because of these problems, many algorithms have been proposed to solve some parts of them.

In this paper, we focus on the image super-resolution while preserving the image edges, because image edges are strongly related to human perception of image

quality. We consider that blocky artifact would decrease the image quality most seriously. Inspired by image vectorization techniques, we noticed that if we can represent the image edges by some parameterization methods. We can reproduce them at any resolution with the preserved edges. Hence, in this paper, we try to extract the image edges and use a parametric representation to capture them. We can get an enlarged image with these edge data while preserving the edges.

(a) (b)

**Fig. 1.** The result of bicubic interpolation, where (b) is the enlarged red square of (a).

## 2 Related Work

Super-resolution has been an interesting topic for a long time, so there are many different algorithms have been proposed. Since it is an under constraint problem, two typical approaches are usually used to overcome this problem, which are adding data and adding constraint.

To add more data, multiple image super-resolution raised. They use multiple low resolution images of the same scene with sub-pixel displacement as the input to compute a high resolution one. Single image super-resolution includes a wide range of work. As summarized in [14].

In recent researches, [16] proposed a method that builds an over-complete dictionary of low resolution image patches from a large image set, and uses a sparse representation of the image with the dictionary to do the super-resolution. [5] proposed a new image prior using image gradients and used these gradients learned from a bunch of natural images to estimate a high resolution image from a low resolution counterpart. [12] is similar to [5], and based on the statistics about the prior it can produce a natural high resolution image.

[8] and [11] are very similar to the anisotropic diffusion. They scale the image up via an interpolation base method, and try to sharpen the edges. Anisotropic diffusion directly employees the well known image sharpen algorithm “anisotropic diffusion”, while [8] and [11] deblur the scaled blurry image.

Besides, [13] proposed a tensor voting mechanism to do the super-resolution, and [3] proposed a soft edge prior to do it while preserving the edges and keeping the

smoothness of each edge. Generally speaking, algorithms that take image edges into consideration can produce a more satisfactory image.

## 3 System Overview

Our algorithm is derived from bilinear interpolation and image vectorization. Basically, Image edges stand for a large color difference, so to sample the colors from different sides of an edge for interpolation would produce jaggy edge. To do the interpolation without edge crossing, we first vectorize the image edges, and analyze the color compositions to get a more compact representation of the image edges.

Fig. 2 shows the system flowchart. There are several main components in our system, which are:

- Edge Detection and Edge Extraction;
- Matting based Image Color Analysis;
- Sub-pixel Refinement and Edge Shape Fitting;
- Represent the image using a Polygonal Representation;
- Edge-Preserving Super-resolution.

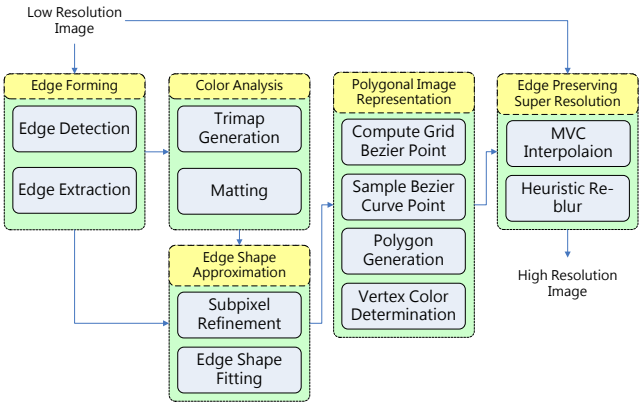


**Fig. 2.** System flowchart.

Since we want to vectorize the image edges, the well known Canny edge detector [1] is used to compute the edge map. The detected edge pixels are linked to form the edges. After extracting the edges from the edge map, we analyze the color information nearby the edges by using a matting algorithm. Then, we use these color information to improve the position of the edge pixels and record these color data as a component of the associated edges.As long as sub-pixel refinement is done, we can vectorize the edge with piecewise smooth cubic Bézier curves. Bézier curve is a parametric curve, we can scale it to any resolution we want without loss its smoothness and any outline deformation.

Finally, to do the interpolation with the edges as the color sampling constrain, we employee the mean value coordinates (MVC) [6] to do the interpolation. MVC is a coordinate with only the function values defined on the polygon vertices. Hence, we make a polygonal representation of the original image with its pixel grids and Bézier curve samples as the vertices. Then, we do a heuristic Gaussian reblurring on the MVC interpolated image.

# 4 Edge Forming

## 4.1 Edge Detection

We use the Canny edge detector [1] to find the edge pixels and employ the MATLAB version of the Canny edge detector, so that we can thin the edge to 1-pixel width successfully. Canny edge detector can only accept a single channel image to compute the edge map. We first convert the image from RGB color domain to YUV color domain, and only use the Y-channel image to compute the edge map. Because the edges detected in the Y-channel image are more intuitive for human.

## 4.2 Edge Extraction

After detecting the edge pixels, we link each pixel with its 8-way neighborhood. The edge map treats as a graph. For each pixel, we record its neighboring amount $N$, which indicates the degree of each pixel after edge extraction. First, we search all pixels with only one neighbor ( $N = 1$ ) as the roots, then traverse the map in a DFS manner. As a pixel has been connected, we decrease its neighbor amount $N$ to reflect how many times the pixel linked.

The graph is traversed from each root until we meet another pixel that has only one neighbor or a pixel with $N = 0$, and this path forms an edge. Because, most edge pixels can have only two neighbors, this process can traverse most edges without any problem. However, if we encounter a pixel has more than two neighbors, we choose the one with smaller spatial distance and color distance as the next pixel. After we traversed all edges from 1-neighbor root, we search all 2-neighbor pixels as the roots again, while the procedure is the same.

# 5 Edge Color Analysis

After edge extraction, we want to interpolate the pixel color without crossing the edges. We need to do the interpolation on a target pixel that is nearby some Bézier curves, we will use the color samples from the Bézier curves and those from the grid points at the same side of the edge to compute its color and preserve the edge at the same time.

### 5.1 Trimap Generation

To utilize the image matting technology, a trimap is necessary. In our system, we generate it automatically by assigning one side of the edge as the foreground region and another side as the background region. However, the edge can reside anywhere in the image, and it may only separate a small area of the image into two regions. Therefore, to generate a trimap associated to an edge, we have to crop a patch of the image nearby the edge first, and then solve the colors by image matting.

### 5.2 Matting


**Fig. 3.** Result of trimap generation and matting algorithm.

After generating the trimap of each edge, we are ready to solve the image matting problem. Though the generated trimap is not so perfect, the closed-form matting [9] can generate quiet adequate solution for most cases. As shown in Fig. 3, (a) is the cropped images for each edge extracted by our algorithm; (b) is the automatically generated trimaps (the white, black, and gray colors indicate the foreground pixels, background pixels, and unknown region, respectively); (c) is the alpha map solved by closed-form matting [9]; (d) and (e) are the solved background and foreground images, respectively.

## 6 Edge Shape Approximation

### 6.1 Sub-pixel Refinement

An ideal edge should reside in between the gradient local maximum and local minimum. The Canny edge detector only has pixel-level precision. Furthermore, an edge in a color image should relate to all color channels. If we detect each color channel separately, how to merge them will be a problem due to the inconsistency. To overcome this problem, we use an alpha map generated by the matting algorithm. As [13] depict, the alpha values are adequate to do the edge pixel enhancement.

Rather than simply utilizing the sub-pixel refinement method in [13], we compute the sub-pixel position of an edge pixel by a method similar to Harris corner detector

[7]. We think that the ideal position of an edge pixel should between the foreground and background that means an edge pixel should have $\alpha = 0.5$. To find such position, we slice the alpha value along the gradient of the point as an 1D function, and search a position of $\alpha = 0.5$ approximately. First, we approximate the 1D function using the Taylor expansion:

$$f(x) \approx f(0) + f'(0)x + \frac{f''(0)}{2}x^2 .$$

Then, we can solve $f(x) = 0.5$ by the above formula and moving the edge pixel to $x$.

### 6.2 Edge Shape Fitting

After extracting the edges from the edge map and the sub-pixel refinement, we can fit the edge shape by a piecewise smooth cubic Bézier curve. For an edge with point $P_0$, $P_1$,..., $P_n$, we want to find a piecewise Bézier curve $Q(t,V)$ that fits $P_0$, $P_1$,..., $P_n$. Assume that the curve $Q(t,V)$ passes through $P_0$ and $P_n$, it could be defined as:

$$Q(t,V) = \sum_{k=0}^{3} B_k(t) V_k ,$$

where $0 \le t \le 1$, $V_0 = P_0$, $V_3 = P_3$, $V = (V_1, V_2)$, and

$$B_k(t) = \binom{3}{k} t^k (1-t)^{3-t} .$$

Then, we can try to find the curve $Q(t,V)$ by minimizing

$$D(t,V) = \sum_{i=1}^{n-1} d_i = \sum_{i=1}^{n-1} [Q(t_i,V) - P_i]^T \cdot [Q(t_i,V) - P_i] ,$$

where $t = (t_1,...,t_{n-1})$.

We employ the algorithm in [2] to do the curve fitting. When the average fitting error of a curve exceeds a threshold set by the user (in our experiments, we set it to 0.5), we split the curve into two curves at a point with the largest fitting error. We do not force smoothness in the conjunction point of the edge split; because of keep it unsmooth can preserve its shape better.

## 7 Polygonal Image Representation

Because the edges' neighborhood may be overlapped and we need a global pixel value when we calculate the target image. To do the interpolation without edge crossing, we use MVC (Mean Value Coordinate) [6] to interpolate the pixel values by using the original image grids and the Bézier curve points as the MVC polygons' vertices.

### 7.1 Computing Bézier Grid Points

To get the Bézier grid points on the original image grids means that we want to find the following set:

$$S = \{t \mid Q(t,V).x \in Z \wedge Q(t,V).y \in Z\}.$$

That means we have to solving $t$ by given $Q(t,V)$ and an integer $n$. Basically, it is a root-finding problem; however, since our equation is in the Bézier form, it can be solved by a more efficient method called Bezier clipping [10].

### 7.2 Sampling Bézier Curve Points

After all Bézier grid points of one Bézier curve are calculated, the set $S$ is sorted for further usage. When we get all grid samples from an edge, we uniformly sample the points between two consequent grid points by simply interpolating the parameter between those points.

### 7.3 Polygonal Image Representation

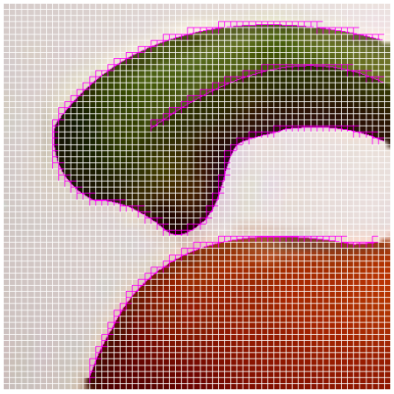

**Fig. 4.** An example of polygonal representation of an image.

Because there can be multiple polygons in an original image grid and the MVC needs polygon vertices in counter clockwise, we build an association list that uses the original image grids as the indices, and record which Bézier grid point belong to the image grid point's neighboring. According to this association list, we can form a point list by traversing each grid point's list counter clockwise. Then, we build a polygon list from the point list.

Note that the Bézier point in between the two Bézier grid points can reside outside of the current image grid, so before connecting them, some checks must be done. Because we use every point inside the list as a starting point of a polygon, we have to check the polygon before inserting it into the polygon list. Finally, we can get the polygon list of a grid.

### 7.4 Vertex Color Determination

Because vertex color can be affected by the edges, we have to use the edge position as a hint to determine the color of each vertex. Hence, we scale the edges first, and then we can compute the vertex color. For each edge, we have assigned each side of it with different regions in the trimap generating step. To determine the color of each vertex, we have to scale the edges to the target resolution, and determine where the foreground and background regions are in the target resolution.

However, we have sampled each Bézier curve into a sequence of points in the polygon generating step. Therefore, we scale each sample point, and then connect the consequent points with a single line. At last, we use an identical foreground and background assignment of the trimap generating step by similar rules. In the following section, we call the scaled foreground and background map as FBMap.

For all Bézier curve points, we assign its color as blend and record its associated Bézier curve, and determine its color until we do the interpolation. For a vertex of the original image grid, it can be covered by FBMap. Here we say "cover" means that the vertex resides in the unknown region of the trimap of the associated edge. If there is no FBMap covers it, we will use the original image pixel color as its color. There is only one FBMap covers it. If it belongs to the foreground or background region, we assign it the color of the foreground or the background. If it belongs to the blend region, we have to determine its color until we do the interpolation,. If there are more than one FBMap cover it, we can calculate the pixel color within each FBMap by fore mentioned method, and calculate an associated confidence value defined by [15]:

$$R_d(F,B) = \frac{\left\| C - (\alpha F + (1-\alpha)B) \right\|}{\left\| F - B \right\|},$$

where $C$ is the original pixel color, $\alpha$ is the alpha value, and $F$ and $B$ are foreground and background colors, respectively. We choose the pixel color with the lowest confidence value as its color.

## 8 Edge Preserving Super Resolution

### 8.1 Mean Value Coordinate

While taking the edges as the constraint, we use the MVC to interpolate the pixel colors. After we determine the polygonal representation of the image, we can use the MVC to interpolate the pixel colors inside each polygon smoothly.

### 8.2 Image Interpolation using MVC

For each scaled grid of the image, we first count how many polygons inside the grid. If there is only one polygon inside, we can directly apply the MVC with the vertices' pixel colors as the function values to do the interpolation. However, we use

the bilinear interpolation to accelerate the process. If there is more than one polygon inside, we need to determine the target pixel belongs to which polygon and then use the polygon to do the MVC interpolation.

When we do the interpolation, there are still some vertices' colors have not been determined, so as we find which polygon the target pixel belongs to, we have to examine the color of each vertex. If the vertex color has been assigned blend, then we first determine the polygon belongs to which side of the associated Bézier curve. If the polygon belongs to the foreground, then each vertex with the blend color should be assigned the foreground color of the edge and vice versa.

### 8.3 Image Reblurring

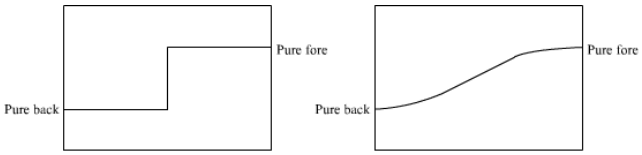


**Fig. 5.** Function along edge gradient.

When we sample the color that affected by the edge, we use the foreground and background colors directly. This procedure makes the gradient along the edge of our system becomes a step function that contains only the pure foreground and background colors as shown in

(a) (b)

**Fig. 5** (a). For a nature image, the gradient along the edge should be a smooth function. As [4] depict, this phenomena will make the image unnatural, so reblurring is needed for a more natural image.

## 9 Result

In this section, we show some results of our method. In each of our experiment, we scale the original image to 8x size. Fig. 6 shows the results. Table 1 lists the performance of each step of our system. We tested our system on a desktop PC with an Intel Core2Quad 2.4GHz CPU with 3.0GB RAM without any optimization. The performance depends on the edge extraction and the input image size.

**Table 1.** Performance.

| | Case1 | Case2 |
|---|---|---|
| Width (px) | 511 | 535 |
| Height (px) | 397 | 500 |

| | | |
|---|---|---|
| Edge Number | 325 | 435 |
| Edge Detection (sec.) | 0.75 | 0.83 |
| Edge Generation (sec.) | 0.01 | 0.03 |
| Trimap Generation (sec.) | 32.08 | 73.81 |
| Mat Solving (sec.) | 16.05 | 20.97 |
| Sub-pixel Refinement (sec.) | 0.14 | 0.13 |
| Edge Shape Fitting (sec.) | 10.06 | 22.16 |
| Béizer Curve Sampling (sec.) | 1.00 | 1.56 |
| FBMap Generation (sec.) | 12.14 | 33.77 |
| Vertex Color Determination (sec.) | 38.05 | 62.36 |
| Edge Preserving Interpolation (sec.) | 278.8 | 502.2 |
| Total Running Time (sec.) | 389 | 717 |

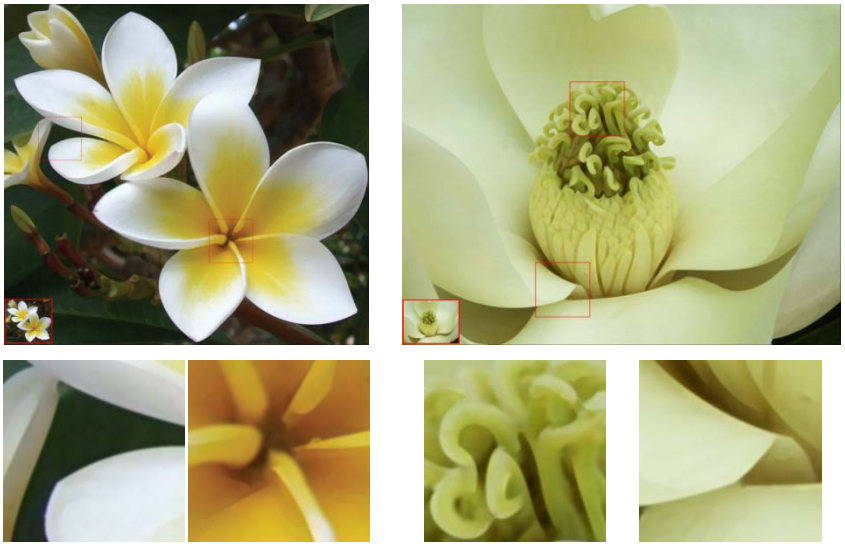

**Fig. 6.** Result images.

## 10 Conclusion and Future Work

In this paper, we proposed a new method to do the edge-preserving image super-resolution. We represent the image edges with an explicit parametric form, and use the image matting as a tool for color analysis. Our system can produce acceptable result even in a very large scale factor. However, since our algorithm is based on the Canny edge detector, there may be some failure cases due to it. To further enhance it is one of our future work. Besides the image quality, the performance is also a typical issue. There are some components may be accelerated by using SIMD instructions or GPGPU technologies.

## 11 Acknowledgement

This paper was partially supported by the National Science Council of Taiwan under 98-2622-E-002-001-CC2 and also by the Excellent Research Projects of the National Taiwan University under NTU98R0062-04.

## 12 References

1. Canny, J.: A Computational Approach To Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 6, pp. 679--698. (1986)
2. Chang, H., Yan, H.: Vectorization of Hand-Drawn Image using Piecewise Cubic Bézier Curves Fitting. Pattern Recognition, vol. 31, no. 11, pp. 1747--1755. (1998)
3. Dai, S., Han, M., Xu, W., Wu Y., Gong, Y.: Soft Edge Smoothness Prior for Alpha Channel Super Resolution. In: IEEE Conference on Computer Vision and Pattern Recognition. (2007)
4. Elder, J.H.: Are Edges Incomplete? International Journal of Computer Vision, vol. 34, no. 2-3, pp. 97--122. (1999)
5. Fattal, R.: Image Upsampling via Imposed Edges Statistic. ACM Transactions on Graphics, vol. 26, no. 3, Article no. 95. (2007)
6. Farbman, Z., Hoffer, G., Lipman, Y., Cohen-Or, D., Lischinski, D.: Coordinates for Instant Image Cloning. ACM Transactions on Graphics, vol. 28, no. 3, Article no. 67. (2009)
7. Harris, C., Stephens, M.J.: A Combined Corner and Edge Detector. In: Alvey Vision Conference, pp. 147--152. (1988)
8. Joshi, N., Szeliski R., Kriegman, D.J.: PSF Estimation using Sharp Edge Prediction. In: IEEE Conference on Computer Vision and Pattern Recognition. (2008)
9. Levin, A., Lischinski, D., Weiss, Y.: A Closed Form Solution to Natural Image Matting. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 2, pp. 228--242. (2008)
10. Sederberg T.W., Nishita T.: Curve Intersection using Bézier Clipping. Computer-Aided Design, vol. 22, no. 9, pp. 538--549. (1990)
11. Shan, Q., Li, Z., Jia J., Tang, C.K.: Fast Image/Video Upsampling. ACM Transactions on Graphics, vol. 27, no. 5, Article no. 153. (2008)
12. Sun, J., Sun, J., Xu Z., Shum, H.Y.: Image Super-Resolution using Gradient Profile Prior. In: IEEE Conference on Computer Vision and Pattern Recognition. (2008)
13. Tai, Y.W., Tong, W.S., Tang, C.K.: Perceptually-Inspired and Edge-Directed Color Image Super-Resolution. In: IEEE Conference on Computer Vision and Pattern Recognition, vol. 2, pp. 1948--1955. (2006)
14. van Quwerkerk, J.D.: Image Super-Resolution Survey. Image and Vision Computing, vol. 24, no. 10, pp. 1039--1052. (2006)
15. Wang J., Cohen, M.F.: Optimized Color Sampling for Robust Matting. In: IEEE Conference on Computer Vision and Pattern Recognition. (2007)
16. Yang, J., Wright, J., Ma Y., Huang, T.: Image Super-Resolution as Sparse Representation of Raw Image Patches. In: IEEE Conference on Computer Vision and Pattern Recognition. (2008)