ISSN 2395-1621

Conversion of Bitmap Image to Vector Image for Image Processing Applications

#1Mrs.Sonal D. Borhade, #2Prof. P. M. Jadhav

1borhadesonal@gmail.com
2 pmjadhav@mit.edu

#1 PG Student, Department of Electronics, Maharashtra Institute of Technology, Pune, India
#2Department of Electronics, Maharashtra Institute of Technology,Pune, India

ABSTRACT

Image vectorization is done to increase visual quality and reducing size of the image. There are various softwares available for this. They scan every pixel of the image and render its vectorized form. Instead, the idea is to perform edge detection and then convert the edge image to vector image. This not only reduces size of the image but also reduces time for execution. This can be used in applications like Rotoscoping (a stage of animation process), erragant gesture correction, correction of bad lighting patch and such other applications. Its application in Rotoscoping is demonstrated. In the near future a video format equivalent to vector format of image is expected to come in use. Also, a codec that fills between the contours is already developed. When these two will work in combination, the visual media will experience a revolution.

ARTICLE INFO

Article History

Received :15th April 2016

Received in revised form :

17th April 2016

Accepted : 19th April 2016

Published online :

23rd April 2016

Keywords— Image Vectorization, Rotoscoping, Vector Image Format, Contours

I. INTRODUCTION

Every digital image falls into one of two categories: a Vector image and a Bitmap image (also called as raster image). A bitmap image, by name itself, represents mapping of bits in the form of pixels in a coordinate system. Every color pixel is stored as a value that lies between 0 to 255. Hence, greater the resolution, greater the number of pixels and greater is the size of image in turn When a bitmap is zoomed or stretched to a high multiple, the edges in the image appear like a staircase and the region with subtle color appear like boxes. This is due to the pixel nature of the bitmap image. This is the problem we usually face when we are trying to use a low resolution logo to fit in a larger space. This results in loss of quality. This greatly reduces file size because only these mathematical entities are stored which needs very less space. Also, the mathematical entities are re-rendered at a greater or smaller scale, providing consistently smooth edges at any size. The idea is that, instead of converting every part of the bitmap image, we will convert only the edges of the objects in the bitmap image. This will take lesser time to compute as will have lesser file size.The vectorized edge image can then be zoomed at any size and then filled with proper colors if required. This type of vector image can prove of great use in logo design and animation artwork.

II. STEPS IN VECTORIZATION

The process of vectorization of image consists of different steps. Every step is selected according to the required nature of final output and the application using it.

A. Converting Input Image to Grayscale

The input image to this step is the full colour bitmap image. Grayscale image carries only intensity information. Images of this sort, also known as black-and-white, are composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest. It is done to reduce computational requirements and also because the final output required is black and white.
The output of this stage is a Grayscale image.

B. Blurring Image

The input to this stage is the Grayscale image generated in the previous stage.Canny edge detection is sensitive to noise. Gaussian blurring of the grayscale image is done to reduce noise. A Gaussian blur effect is typically generated by convolving an image with a kernel of Gaussian values. In practice, it is best to take advantage of the Gaussian blur's separable property by dividing the process into two passes. In the first pass, a one-dimensional kernel is used to blur the image in only the horizontal or vertical direction. In the second pass, another one-dimensional kernel is used to blur in the remaining direction. The resulting effect is the same as convolving with a two-dimensional kernel in a single pass, but requires fewer calculations. When converting the Gaussian's continuous values into the discrete values needed for a kernel, the sum of the values will be different from 1. This will cause a darkening or brightening of the image. To remedy this, the values are normalized by dividing each term in the kernel by the sum of all terms in the kernel.

The output of this stage is a blurred grayscale image.

C. Edge Detection

The input to this stage is Grayscale image with blur.

The purpose of edge detection in general is to significantly reduce the amount of data in an image, while preserving the structural properties to be used for further image processing. Here we have used canny Edge operator. The performance of the Canny Edge Detection algorithm depends heavily on the adjustable parameters, σ, which is the standard deviation for the Gaussian filter, and the threshold values, '*T1*' and '*T2*'. σ also controls the size of the Gaussian filter. The user can tailor the algorithm by adjusting these parameters to adapt to different environments [10].

Canny's edge detection algorithm is computationally more expensive compared to Sobel, Prewitt and Robert's operator. However, the Canny's edge detection algorithm performs better than all these operators under almost all scenarios [11]. Evaluation of the images showed that under noisy conditions, Canny, LoG, Sobel, Prewitt, Roberts's exhibit better performance, respectively. Hence, the best of them i.e. canny edge operator was selected for use.

The output of this stage is an edge image with a specified threshold.

D. Tracing Algorithm

The input to this stage is an edge image in Bitmap format.

A comparative study of different tracing algorithms revealed that the Potrace algorithm is very efficient; it produces visibly better outputs than other comparable algorithms [5]. In addition to its superior graphical output, Potrace also compares favorably in terms of speed and file size. Potrace mainly works on concepts of paths, nodes and graphs, those are easier to computationally implement. Potrace also use the linear Bezier Curves for smoothing. They are easily available in Opencv. Other comparable algorithms have mathematically extensive computations.

So, Potrace Algorithm was selected for tracing. The output of this stage is vector image ready to display in vector image format.

E. Display Output in Vector Image Format

Tracing algorithm is implemented in Opencv and Opencv software does not support vector image format. So there emerged a need of software which will support such a format and display the output.

There were several choices from which two of them are used, Inkscape and Filemonitor. The output of Potrace algorithm is directly exported to Inkscape and filemonitor.

This is the final Vector Image output which can be used in several applications.

An easy way to comply with the conference paper formatting requirements is to use this document as a template and simply type your text into it.

III. Edge Detection of image

The edge detection is first carried out on the input image the purpose of which is to significantly reduce the amount of data in an image, while preserving the structural properties to be used for further image processing. John F. Canny (JFC) in 1986 has developed one such algorithm 'Canny Edge Detection Algorithm' which will be used for this purpose [1].

The algorithm runs in 5 separate steps:

1. Smoothing: Blurring of the image to remove noise.

2. Finding gradients: The edges should be marked where the gradients of the image has large magnitudes.

3. Non-maximum suppression: Only local maxima should be marked as edges.

4. Double thresholding: Potential edges are determined by thresholding.

5. Edge tracking by hysteresis: Final edges are determined by suppressing all edges that are not connected to a very certain (strong) edge.

Images after edge detection by canny edge detection algorithm are obtained using these steps and the image obtained is given in fig.4.

IV. Potrace Algorithm

The equations are an exception to the prescribed specifications of this template. You will need to determine whether or not your equation should be typed using either the Times New Roman or the Symbol font (please no other font). To create multileveled equations, it may be necessary to treat the equation as a graphic and insert it into the text after your paper is styled. A colon is inserted before an equation is presented, but there is no punctuation following the equation. All equations are numbered and referred to in the text solely by a number enclosed in a round bracket (i.e., (3) reads as "equation 3"). Ensure that any miscellaneous numbering system you use in your paper cannot be confused with a reference [4] or an equation (3) designation.The Potrace algorithm transforms the Edge Detected Image into a vector outline in several steps. In the first step, the bitmap is decomposed into a number of paths, which form the boundaries between black and white areas. In the second step, each path is approximated by an optimal polygon. In the third step, each polygon is transformed into a smooth outline. In an optional fourth step, the resulting curve is

optimized by joining consecutive Bezier curve segments together where this is possible. Finally, the output is generated in the required format.
Execution of every step is carried out in the following stages.

Stage 1: Paths
1. Path runs through vertices. So find the vertices.
2. Construct a directed graph consisting of closed paths.
3. Perform despeckling by dropping all paths whose interior consists of fewer than 't' pixels.

Stage 2: Polygons
1. Construct polygons from closed paths.
2. Optimize polygon to have fewer segments.

Stage 3: From polygons to vector outlines
1. Adjust the position of the vertices of the polygon to fit the original bitmap as best as possible.
2. Calculate corner and curves based on the lengths of adjacent line segments and the angles between them.

Stage 4: Curve optimization
1. Attempt to join adjacent curve segments.
2. Join adjacent curve segments that agree in convexity.
3. Join adjacent curve segments if total change in direction is less than 179 degrees.

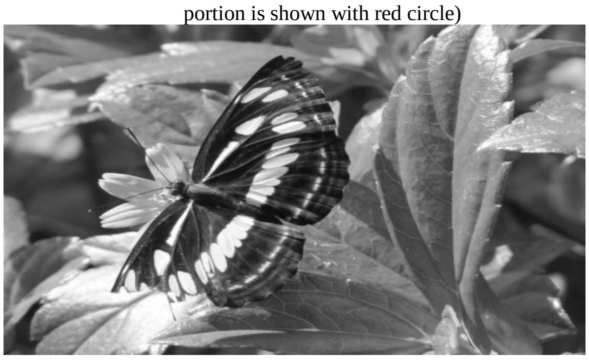
Stage 5: Output Generation
1. Perform scaling and rotation of the image by linear transformation.
2. Perform redundancy coding in the postscript backend because it takes advantage of the macro abilities of postscript language.
3. Quantize the final coordinates, which are real numbers. (Round them to closest 1/10 of pixel). The coordinates of the points are then output as integers.

The Coding for this algorithm is performed in Open CV Software. We face a major limitation of the use of this software tthat OpenCV does not support any vector image format to display its output. Hence, we exported the output in File Monitor , which supports .svg format (format for vector image) and displayed the results.

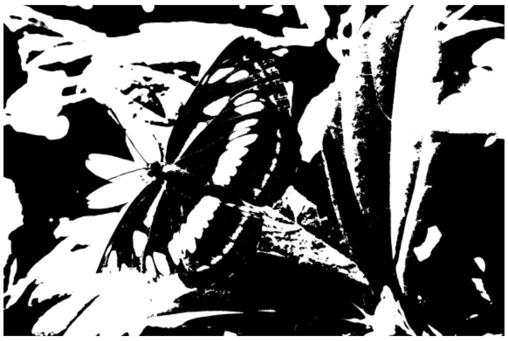
V. Results

High Resolution, Clear, Good Contrast, Well Illuminated Input

1. Original image captured with *Canon* PowerShot SX610 HS (target portion is shown with red circle)

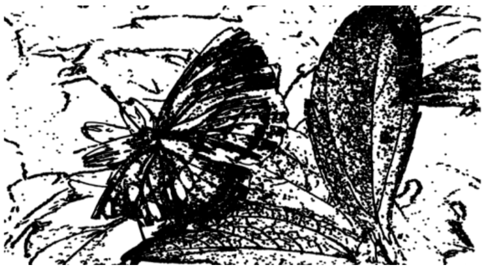


2. Gray scale image

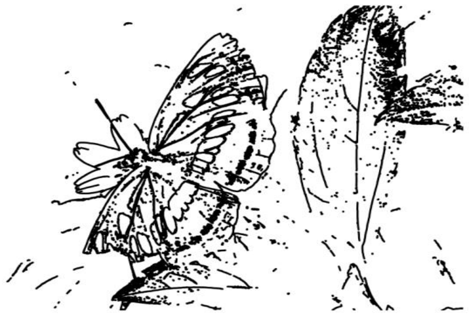


3. Binary image

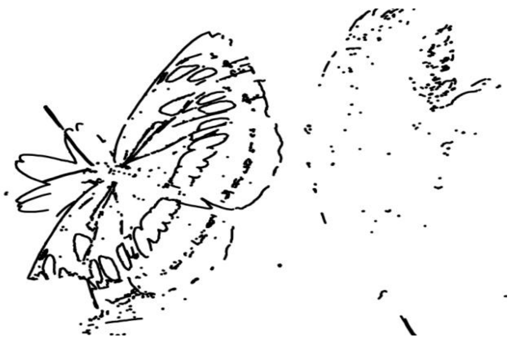
4. Target portion Zoomed of Binary image

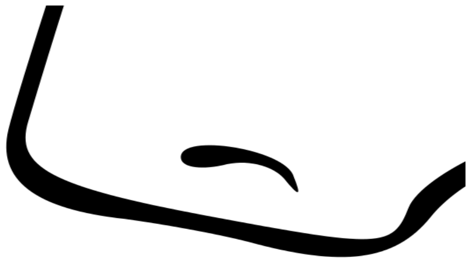


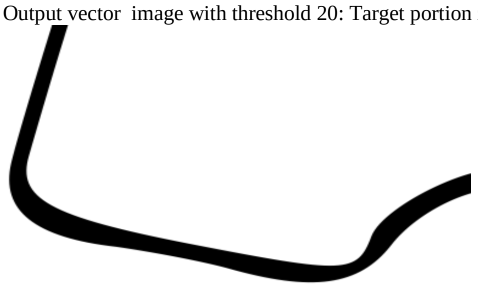
5. Output Vector image with Threshold 20



6. Output Vector image with Threshold 50


7. Output Vector image with Threshold 100


8. Output vector image with threshold 20: Target portion zoomed


9. Output vector image threshold 50 :Target portion zoomed

- **Fig.1.** Image is captured by *Canon* PowerShot SX20 HS. Size 2.4 Mb. The Image consists of a butterfly with a cut wing. The target portion on the cut wing is highlighted by a red circle.
- **Fig.2.** Image is converted to grayscale image before application of Canny Edge operator resulting in **Image 2**.
- **Fig.3.** Image is converted to binary image
- **Fig.4.** Image is the zoomed target portion of the binary image. The pixellation and staircase effect is clearly visible.
- Then conversion to vector image is done with Different Threshold Values set for the Canny Edge Detection operator.
- **Fig.5.** Image is output vector image with threshold value 20. The output has more number of edges.
- **Fig.6.** Image is output vector image with threshold value 50. The output has lesser number of edges as compared to Image 5.
- **Fig.7.** Image is output vector image with threshold value 100. The output has only very prominent edges remaining. Threshold value greater than this may remove many important edges. So, using value greater than 100 for images having greater details will not be a good choice.
- **Fig.8.** Image is the target portion of output vector image (with threshold 20) zoomed. Comparing this image with image 4 (binary image), the edges are continuous and crisp.
- **Fig.9.** Image is the target portion of output vector image (with threshold 100) zoomed. One smaller edge in the inner area is not visible.

Some peculiar advantages of this execution is that outputs with different thresholds can be achieved which cannot be done with other existing vectorization softwares. Size of output image is smaller as specified below.

Original Image (specimen): 2.96 Mb

Project Output Image: 62.26 kB

Inkscape Output Image: 1.04 Mb

Also accuracy can be further increased by setting values of parameters in Potrace Algorithm.

VI. CONCLUSION

A vectorization method to convert Bitmap image to vector image is developed in which edges in the images are extracted first and then converted to vector image. This gives advantage of further reduction in size of the image.

.

REFRENCES

1. “Potrace: a polygon-based tracing algorithm”, Peter Selinger,September 20, 2003.
2. Canny Edge Detection, 09gr820, March 23, 2009.
3. Building Simulation programs in Visual Studio (.NET 2003 and 6.0).
4. OpenCV Reference Manual v2.2, December 2010.
5. “Depixelizing Pixel Art”, Johannes Kopf, Microsoft Research, Dani Lischinski , The Hebrew University, May 2011.
6. “Vector Based Codec”, Prof. Phil Willis, University of Bath,Uk, 12 December 2012
7. "Some Experiments in Image Vectorization", J. Jimenez and J.L. Navalon, , IBM J. Res. Develop. 26, pp 724-734, 1982.
8. "Computer Processing of Line Images: A Survey", R.W. Smith, Pattern Recognition, 20(1), pp7-15, 1987
9. “Sparse Pixel Vectorization: An Algorithm and Its Performance Evaluation”, Dov Dori and Liu Wenyin, , 2002.
10. “Study and Comparison of Various Image Edge Detection Techniques”, Raman Maini & Dr. Himanshu Aggarwal, International Journal of Image Processing (IJIP), Volume (3): Issue (1)
11. “A Comparison of various Edge Detection Techniques used in Image Processing”,G.T.Shrivakshan1, Dr.C. Chandrasekar, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 5, No 1, September 2012, ISSN (Online): 1694-0814