A framework for detection of linear gradient filled regions and their reconstruction for vector graphics

Ruchin Kansal
Department of computer science
IIT Delhi
India, New Delhi
rkansal@adobe.com

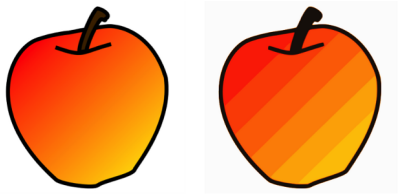
Prof. Subodh Kumar
Department of computer science
IIT Delhi
India, New Delhi
subodh@cse.iitd.ac.in

ABSTRACT

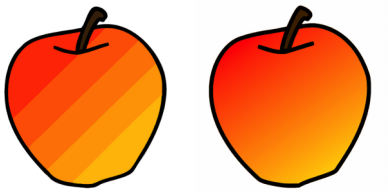
Linear gradients are commonly applied in non-photographic art-work for shading and other artistic effects. It is sometimes necessary to generate a vector graphics form of raster images comprising such art-work with the expectation to obtain a simple output that is further editable, say, using any popular vector editing software. Further, this vectorization process should be automatic with minimal user intervention. We present a simple image vectorization algorithm that meets these requirements by detecting linear gradient filled regions and reconstructing the vector definition using that information. It uses a novel interval gradient optimization scheme to derive large regions of uniform gradient. We also demonstrate the technique on noisy images.

Keywords

Image vectorization Gradient reconstruction Region detection



(a) Original Image (b) Adobe Livetrace output



(c) Inkscape output (d) Our output

Figure 1: Comparison of diferent outputs.

1 INTRODUCTION

The advent of new mobile and touch devices has motivated designers to create and publish their content in vector format. Vector graphics represents 2D images in terms of mathematical elements like curves, contours, straight lines and other shapes, along with their attributes such as fill, stroke, transparency and so on. This is different from raster representation, which stores a color sample at each pixel center. Vector graphics supports rasterization on the fly and therefore it can be viewed at different scales and resolutions without any artifacts.

With the increasing demand for vector content, the need for converting raster images into vectors has also increased. This process is called *Vectorization*. We set out to obtain automatic vectorization with minimal human intervention even on potentially noisy images. Later re-rasterization of our vector representation should produce the appearance of the original image at various scales. Further, for wide-spread usage, this vector representation should be in a standard form and be editable. However, it is not easy for an algorithm to meet all these conditions for every image. In fact, these may be conflicting goals. For example, to match the vector appearance with the original image, an algorithm might generate smaller patches due to which editing becomes difficult. Further, application of color gradients for shading effects in a non-photographic image poses additional challenges to vectorization. In this case, the vectorization algorithm must derive the gradient definition and use it to approximate the color information. Many algorithms, including commercial software [27, 1, 13], are unable to appropriately reconstruct such gradients. For example, they may approximate the gradients with patches of constant fill regions. (See Inkscape [13] and Adobe Illustrator [1] examples in figures 1c and 1b respectively).

A few others, including the one titled ARDECO [17], focus on computing the gradient. For example, ARDECO uses first and higher order gradients to store the color information. However, these techniques either generate too many vector patches or their

 ISBN 978-80-86943-75-6

representations are so complex that their output cannot be easily edited.

Vector graphics can be represented using an open vector format such as *EPS*, *PDF*, or *SVG*[28] or it could be a proprietary format (such as *Adobe Illustrator* or *Corel*). Among open formats, SVG is possibly the most widely used vector format for web and digital media, which we have chosen as our output. Nonetheless, the definition of linear gradient is largely the same in all vector standards modulo occasional minor differences. SVG specifies *linear color gradient* as continuous smooth color transition along a 2D direction from one given color at a known position to another. This direction is called the *Gradient Vector*. The value of each pixel along the gradient vector may be calculated by linearly interpolating the two end colors. The *gradient normal* is orthogonal to the gradient vector. The color of each pixel on the gradient normal remains same. The SVG standard also allows fixing of more than two colors along the gradient vector, to form a smooth multi-color transition. These specific points on the gradient vector with pre-defined color values are called *Gradient Stops* (See figure 2).

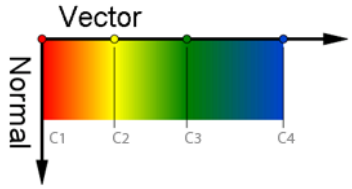


Figure 2: Linear Gradient defined by four gradient stops(C1, C2, C3 and C4). Notice the color along the gradient vector is defined by linear interpolation from one stop to another but the color of the pixels along the gradient normal remains same.

We have developed an approach that can detect linear gradient filled regions as well as the gradient values. While the contributions of this paper are primarily in effective recovery of regions with uniform gradient, for completeness we do also produce boundary curves and regions with uniform fill color where necessary. We do not target vectorization of photographic quality images, but rather art-design by artists. The distinguishing feature of such images is that they contain relatively large areas of uniform fill and gradients but suffer from noise and other smoothing and post-processing artifacts.

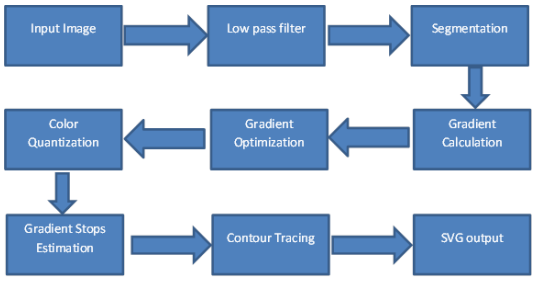


Figure 3: The complete pipeline

2 PREVIOUS WORK

Vectorization generally involves some form of image segmentation followed by a vector approximation for each segment. Significant research in both vectorization and, more generally, image segmentation exists. The research in vectorization is done with various objectives such as fitting smooth curves [15, 24, 2, 9], minimizing the number of colors used [29], preserving editability [4] and matching appearance with input [17, 25, 16, 21]. Image segmentation prior to vectorization is commonly based on edge detection [6], color quantization [8, 11, 5] or a global optimization function [19, 18, 20] to reduce the overall energy.

Early work in this field focused on line drawings and bitonal images [6, 9, 10]. These approaches are mainly based on edge detection [6], thresholding [7, 14, 22, 23], thinning [26] and contour tracing [12]. The extracted line, image contour or region is represented by vector graphics primitives, e.g., curves and paths. More recent algorithms [17, 25, 16, 21, 29, 4, 15] deal with full color images and their goal is to generate high fidelity output.

ARDECO[17], proposed by Lecot and Levy in 2006, tries to decompose the input image into patches. Each patch is approximated by a constant color, linear or higher order gradient in order to minimize the overall energy. The energy function in their approach is determined by a boundary length function and a curve fitting term. The weighting of terms is controlled by user input. Since the energy functional is quite generic, it can handle images with fine details. At the same time it often produces a large number of patches and consequently it is not possible to edit the final vector graphics easily for post-editing. Further, for large gradient fill regions, it often fails to converge to any result. Also, due to linear constraints the segment boundaries produced by them is often not smooth. Finally, the user needs to adjust several parameters by experimentation. Our algorithm is simpler than ARDECO as it considers only first order gradients. Further, our algorithm produces fewer regions so that a user can edit the image easily.

Sun et al [25] introduced a vectorization approach using *Gradient Meshes*. There, a gradient mesh is defined by a grid using topologically planar rectangular

 ISBN 978-80-86943-75-6

Ferguson patches with mesh-lines. Control points of the mesh have three attributes: position, derivative and color. Their approach relies on user assistance for mesh initialization and placement. Recently, Lai, Hu and Martin, 2009 [16] improved that algorithm by generating the gradient mesh automatically. The output of gradient mesh is quite impressive and it can even be applied to photographic images. However, the size of their representation is too unwieldy for further editing. Moreover, Gradient Mesh is not a standard primitive and are hence less portable. They cannot be rendered or edited by standard tools.

Price and Barett [4] proposed an approach for interactive image editing using object based vectorization. They allow the user to select an object and then graph cut is used recursively to form a hierarchical object tree. For each object they define a mesh by locating the corner points and doing recursive subdivision. The resulting mesh can be edited by various tools. However, the approach is designed to be driven by user manually. Also, the algorithm does hot handle gradient reconstruction, it only provides a better means of object construction.

Diffusion curves[21] is a different approach to represent smooth shaded images. A diffusion curve partitions the space through which it is drawn, defining different colors on either side. These colors may vary smoothly along the curve. In addition, the sharpness of the color transition from one side of the curve to the other can be controlled. Due to the limitations with Poisson equations, the color variations in all raster images may not be represented by this system, especially when the image has sparse features in some areas.

Xia et al [29] proposed a vector based representation in which the image is decomposed into non-overlapping triangular patches with curved boundaries. The color variation over each triangular patch is approximated with a thin-plate spline for every color channel. It allows them to approximate raster images with both smooth variations and curvilinear features. Although, the representation is powerful and compact, again editability and portability is a concern.

A nice discussion of vector primitives related to color gradients is provided by Pascal et al. [3]. They describe various available techniques for construction and rendering of such vector primitives. They mention the current methods of vector creation by hand as well as through vectorization. Some practical challenges and limitations of these approaches are also explained.

Adobe live trace [1] and Vector magic [27] are popular commercial applications, which are used for vectorization and produces standard vector graphics. Inkscape [13] is an open source tool which is also used for vectorization and vector editing. However, none of these packages recognizes linear color gradients in the image and therefore such regions are approximated by rectangular stripes (See figure 1).

3 OUR APPROACH

Figure 3 depicts our pipeline. We start with a raster image. In our experiments all input images are 8-bit per-channel RGB images but the technique is independent of the color format. Like most vectorization approaches, we first segment a filtered version of the image. Color discontinuity imposes segment boundaries. Next, for each segmented region we determine if it can be represented by a linear gradient function. This decision is made by searching for a gradient value that is supported by all pixels of the region. In particular, we calculate the range of gradient values supported by each pixel. A global optimization across pixels of the region then determines the most plausible gradient for the region. Finally, using this optimized gradient direction, we find the gradient stops within the region. In terms of figure 3, the main contributions of our algorithm are in stages 4, 5 and 7.

For regions that cannot be represented using a linear gradient, we employ a color quantization approach to minimize the number of vector elements. Each region is then vectorized using the potrace engine [24] (although any vectorization approach would suffice). Potrace is designed to generate smooth contours of the features which works well with our pipeline. Each stage of the pipeline is described next.

3.1 Image Smoothing

To reduce the effect of noise in the image, we apply a Gaussian blur of radius 3. It reduces the sharpness near edges and produces a relatively smooth image.

3.2 Initial Segmentation

As a preprocessing step, we first perform image segmentation using a standard scheme. We have used mean shift segmentation followed by a flood fill, which gives us good result. although a more specialized segmentation algorithm can also be employed.

3.3 Gradient Approximation

In order to determine the gradient direction m we reconstruct values from the given samples. However, we must also consider that the input color values are imprecise due to noise, smoothing and rasterization. Due to imprecise color values at pixel centers, we resort to an interval color scheme. In particular, if the color at pixel p input to this stage is c, we allow that the actual color is in the range $[c_- : c_+]$, where $c \in [c_- : c_+]$. For example, if c only has rounding error, $c_- = c - 0.5$ and $c_+ = c + 0.5$.

If the gradient slope at pixel p is m, we expect the color at the gradient normal $p + k\frac{1}{m}$ within a segment to be in

 ISBN 978-80-86943-75-6

Algorithm 1 Find the slope range

```
1:  procedure FINDRANGE(p, R) ▷ Calculate range of slopes for pixel p over region R
2:      p.range ← ∅ ▷ Initialize the range of p as empty
3:      S ← Boundary pixels of R ▷ Initialize vector S with boundary pixels of R in such an order that all consecutive pixels in S are neighbors in R
4:      i ← 0
5:      c ← p.color
6:      for i < S.size − 1 do
7:          p1 ← S[i]
8:          p2 ← S[i + 1] ▷ Get two neighbor pixels from S in p1 and p2
9:          if [c− : c+] ⋂ [p1.color, p2.color] ≠ ∅ AND p.region = p1.region = p2.region then
10:             L1 ← line(p1, p) ▷ Find line L1 passing through p1 and p
11:             L2 ← line(p2, p)
12:             p′1 ← intersection(L1, R) ▷ Find intersection of L1 with R
13:             p′2 ← intersection(L2, R)
14:             if [c− : c+] ⊆ [p′1.color, p′2.color] then
15:                 p.range ← [slope(L1), slope(L2)]
16:                 break;
17:             end if
18:         end if
19:         i ← i + 1
20:     end for
21: end procedure
```

the interval $[c_- : c_+]$ for all values of k within a range, if the input color at p is c. However, due to the rasterization in the input image we may not have samples available for any value of k.

We consider a contour around p and locate the normal line passing through p on this contour. For example, this contour can be a rectangle R. Our goal is to locate the range $[c_- : c_+]$ on R. Assuming color interpolation along R, we find two samples p_1 and p_2 on R such that the color c_1 at p_1 and color c_2 at p_2 span the range $[c_- : c_+]$, where p_1 and p_2 are the closest such pixels along the contour. In other words, $[c_- : c_+] \subseteq [c_1 : c_2]$. We conjecture that the normal line intersects the rectangle between p_1 and p_2. As an aside, if one were to search for the exact value c reconstructed from samples near p_1 and p_2, it would yield unreliable estimates for m that are often inconsistent with the estimates of p's neighbors.

If p is not on the boundary of its region, a pair (p_1, p_2) implies the existence of another pair (p'_1, p'_2) on the opposite side of the rectangle. For pair (p_1, p_2), we form straight lines by joining p_1 and p, and similarly by joining p_2 and p (Figure 4 explains this construction, see the blue and red lines passing through p). The intersections of these lines with the opposite boundary of rectangle R provides the conjugate pair (p'_1, p'_2). Again, we need not have samples available at p'_1 and p'_2, unless R is symmetric about p. We reconstruct the color, respectively, c'_1 and c'_2 at positions p'_1 and p'_2 from the neighboring samples. If again $[c_- : c_+] \subseteq [c'_1 : c'_2]$, it is evidence of the normal line passing between p'_1 and p'_2. If the slopes of lines $p_1p'_1$ and $p_2p'_2$ are $\frac{1}{m_1}$ and $\frac{1}{m_2}$, respectively, we say that pixel p favors a color gradient in the range $[m_1 : m_2]$ subject to the condition that pairs (p_1, p_2) and (p'_1, p'_2) lie in the same image region. Please note that if the range $[p'_1 : p'_2]$ is not tight and its subset contains the color range $[c_- : c_+]$, that subset is used instead to provide a tighter gradient range. This approach is presented in algorithm 1.

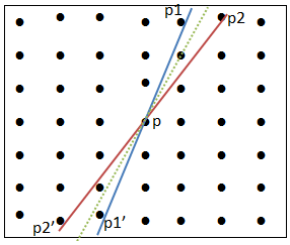


Figure 4: The setup: A rectangular grid of pixels around p is considered. The color interval of pixel p, $[c_- : c_+]$, lies in the colors at pixels p_1 and p_2. This implies the normal direction through p, passes between p_1 and p_2. Consider lines joining p with p_1 and p_2 respectively. These lines intersect at the opposite side of the grid on p'_1 and p'_2. If $[c_- : c_+]$ is spanned by the colors at p'_1 and p'_2, the normal directions is assumed to lie between the two solid lines. Additionally, the green dotted line is formed by fitting a line among all pixels whose colors are similar to that of pixel p. This estimated slope is also stored for each pixel p.

Not the entire range of gradients $[m_1 : m_2]$ is equally favored by p. We also estimate the most favored gradient m' and weight a value $m \in [m_1 : m_2]$ by its difference from m'. To find m', we compute the best fit line to the color values nearest c within R. In particular, we form a set of points S including every pixel within rectangle R with color within $[c_- : c_+]$. The calculation of favored slope is explained in algorithm 2.

If a pixel does not produce a gradient range, either it is not a part of a gradient filled region, or it cannot provide candidate gradients due to noise in the image. On the other hand, it is possible for a pixel to provide multiple gradient candidates due to noise. We include all ranges in the optimization process described next.

Every pixel p_i of a presumed gradient fill region similarly produces its favored gradient m'_i and slope range (m_{i_1}, m_{i_2}). We choose a single gradient value for the region that best satisfies all ranges.

 ISBN 978-80-86943-75-6

Algorithm 2 Find the favored gradient slope

```
1:  procedure FINDGRADSLOPE(p,R)   ▷ Calculate favored gradient slope of pixel p over region R
2:      p.slope ← nil
3:      Q ← all pixels of R   ▷ Initialize vector Q with all pixels of R
4:      i ← 0
5:      S ← ∅
6:      for i ≠ Q.size do   ▷ Loop on all pixels in Q
7:          q ← Q[i]
8:          if p.color ∈ [q.color_− : q.color_+] AND p.region = q.region then
9:              S ← S⋃{q}
10:         end if
11:         i ← i+1
12:     end for
13:     if S ≠ ∅ then
14:         L ← FitStraightLine(S)
15:         p.slope ← slope(L)
16:     end if
17: end procedure
```

Choice of this region R is important as it should be sufficiently large to have enough samples to reproduce the reliable gradient range and value. The size of region R may also vary on each pixel depending on the noise in the image. Therefore, we choose rectangles of dynamically varying sizes whose dimensions are decided on each pixel. We start with a size of 3x3, and keep on increasing this region until the line fit error is below a certain threshold ε. Because of this dynamically sized region, our approach can handle different kinds of noisy images sucessfully.

3.4 Gradient Optimization

After computing the favored gradients for each pixel p_i, m'_i and the range $[m_{i_1} : m_{i_2}]$, the final gradient m_r for the region should ideally lie in this range and as close to m'_i as possible. We compute m_r by optimizing across all pixels of the region. This optimization can be easily posed and solved using a simple geometrical construction.

We select a function which maximizes its potential if the selected gradient $m_r = m'_i$. This potential monotonically decreases as m_r grows apart from m'_i. One can select a Gaussian or radial basis function as the weight, but a cosine-linear weight function provides the best results. Given two line slopes m' and m'', the dot product of the vectors in their respective directions gives a projection of a vector on another. By definition, the magnitude of the dot product of two unit vectors decreases as the angle between them grows. We define our objective function to maximize the sum of these dot products. To do so, the value of objective function $f(x)$ is computed by finding all pixels p_i which have the range $[m_{i_1} : m_{i_2}]$ containing x and performing a summation over the dot products with their favoured slope m'_i. i.e. $f(x) = \sum_{i=0}^{n} |g(x,i)|$ where $g(x,i) = \hat{x}.\hat{m'_i} | m_{i_1} \leq x \leq m_{i_2}$ and $\hat{x}$ and $\hat{m'_i}$ are two unit vectors in the direction of x and m'_i respectively.

The linear optimization can be performed using any standard technique like the dynamic system based global optimization [19].

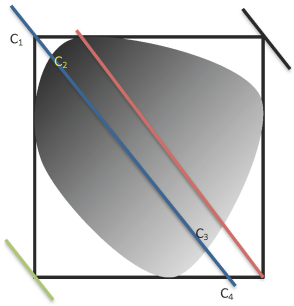


Figure 5: Gradient Stops Estimation: The shaded area is a gradient filled region while its bounding box is marked as black rectangle.We draw lines from four corners of the bounding box parallel to gradient axis (shown in different colors), since the line passing from top-left corner (marked in blue) overlaps the maximum pixels of the region, it is used for gradient stops estimation. Two stops are generated where line hits the bounding box(C1 and C4) while two stops are generated where line intersects the regions (C2 and C3). Also, note that value of C2 and C3 is determined by using the pixel color at the respective location of image while value of C1 and C4 is computed by extrapolation of C2 and C3 along the gradient axis.

3.5 Gradient Stops Estimation

We use a heuristic approach to find the gradient stops. If we draw lines parallel to the computed gradient vector m_r from each corner of the bounding box of the region as shown in Figure 5, the one with the largest overlap with the region may be selected for stops estimation. We generate four color stops on the gradient vector, two on each end points on the bounding box and two on each intersection of this line with the region (See Figure 5). If the points at C1 and C4 do not lie in the region, their colors may be estimated using extrapolation from C2 and C3 as shown.

3.6 Color Quantization

Segmentation for the remaining colored regions of the image is performed using color quantization [11]. A color palette of the given number of colors is first prepared, and then each pixel is assigned the index of color palette that it best matches. For our experiments we

 ISBN 978-80-86943-75-6

have used an octree based color quantization approach [8, 5].

3.7 Contour Tracing

Tracing is the process of fitting curves that bound each image regions. After tracing, we obtain a set of curves that represent the image geometry. We employ the potrace engine developed by Peter Schilinger [24] for this outline tracing. The same engine is also used by open source vector drawing package Inkscape [13].

3.8 Final Vector Output

Once we obtain the curves outlining each image segment, we apply fills to these curves and generate the final vector representation (in the SVG format). The region color, as noted before, may be either a solid fill color obtained through quantization or a linear gradient produced by the optimization algorithm.

4 RESULTS AND VALIDATIONS

We analyzed the vector output of our algorithm from various prespectives like appearance, editability, accuracy and error per pixel. Results are given below.

4.1 Appearance

We applied our approach to different non-photorealistic raster images and the results are presented in Figure 6.

4.2 Comparison with ARDECO

The implementation of ARDECO is publicly available on the authors webpage. In figure 9 ARDECO produced more than 1200 paths while our approach outputs 4 paths only. This is because we perform an early segmentation and then apply gradient detection on the various segments. We are also able to handle noisy images, as shown in figure 9 where the input image contains random RGB noise.

4.3 Editability

The output SVG can be easily edited using any standard vector graphics tool like Inkscape. Examples are shown in figure 7.

4.4 Accuracy

To measure the accuracy, we applied our algorithm to images whose gradient direction and magnitude was already known. The results are shown in figure 8.

	Our Error		*Inkscape Error*	
Input	***Gradient***	***Solid***	***Gradient***	***Solid***
6e	7.7	16.25	17.49	24.40
6k	11.4	13.2	18.64	21.31
6c	11.63	18.67	14.76	26.55
6i	15.64	25.14	26.20	44.8
6a	25.51	34.26	46.92	37.40

Table 1: We calculated the per pixel error for gradient and solid colored regions separately in our output and then compared with the corresponding region error in inkscape.

4.5 Per Pixel Error

To analyze the per pixel error in our output, we rasterized our vector output and then compared it with the original image to compute root mean squared error per pixel. Table 1 compares the error in our gradient and solid colored regions with the corresponding regions in Inkscape output. Both Inkscape's and our approach used a quantization palette size of 16 colors.

Table 1 shows that even the per pixel error for solid colored regions is low with our approach as compared to Inkscape. This is due to the fact that our approach excludes the gradient region while performing quantization, therefore with the same size of color palette, more accurate colors are represented.

The high per pixel error in output can be explained due to the several factors:

1. Vector and raster spaces are not equivalent. The pixel at location (x, y) in input image may not be present at the same exact location in the vector space.

2. Input image may contain small pixel level features, which are merged in larger regions during vectorization.

3. Vector output is optimized to be represented with fewer colors using some method of color minimization.

5 LIMITATIONS AND FUTURE WORK

We have proposed an algorithm to find gradient in images that optimizes the gradient values across noisy pixels. It mainly targets reconstruction of simple art drawings that can then be further edited or stylized. The proposed algorithm works well when the linear gradient in input image is specified by two color stops. Otherwise the the gradient region may be split into multiple smaller regions. This limitation can be easily handled by modifying the gradient stops estimation step to account for multiple color stops. Our approach may also not produce good results when the linear gradient is applied on small width regions, like linear gradient

 ISBN 978-80-86943-75-6

on a single pixel wide curve, for it needs to find segments with a few neighbors around its pixels. We believe the algorithm can be easily adapted to handle non-linear gradients – for example a radial gradient. Our algorithm is designed to operate on each pixel independently, therefore it can parallelize well. Future work should also include deriving vector graphics for videos and using the level of optimization in a feedback loop to refine the segmentation potentially producing even fewer patches.

6 REFERENCES

[1] Inc. Adobe Systems. Adobe illustrator cs5, 2010.

[2] Autotrace. An open-source program for converting bitmap to vector graphics, 2004.

[3] Pascal Barla and Adrien Bousseau. Gradient art: Creation and vectorization. In Paul Rosin and John Collomosse, editors, *Image and Video-Based Artistic Stylisation*, volume 42 of *Computational Imaging and Vision*, pages 149–166. Springer London, 2013.

[4] William A. Barrett and Alan S. Cheney. Object-based image editing. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 777–784, New York, NY, USA, 2002. ACM.

[5] Dan S Bloomberg. Color quantization using octrees. 2008.

[6] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.

[7] Jung-Shiong Chang, Hong-Yuan Mark Liao, Maw-Kae Hor, Jun-Wei Hsieh, and Ming-Yang Chern. New automatic multi-level thresholding technique for segmentation of thermal images. *Image and Vision Computing*, 15(1):23 – 34, 1997.

[8] D. Clark. Color quantization using octrees. *Dr. Dobb's Journal*, pages 54–57, Jan 1996.

[9] Dov Dori, Senior Member, and Wenyin Liu. Sparse pixel vectorization: An algorithm and its performance evaluation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 21:202–215, 1999.

[10] Kuo-Chin Fan, Den-Fong Chen, and Ming-Gang Wen. A new vectorization-based approach to the skeletonization of binary images. In *ICDAR*, pages 627–630. IEEE Computer Society, 1995.

[11] Michael Gervautz and Werner Purgathofer. A simple method for color quantization: Octree quantization. In *New Trends in Computer Graphics*. Springer Verlag, Berlin, 1988.

[12] O. Hori and S. Tanigawa. Raster-to-vector conversion by line fitting based on contours and skeletons. In *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on*, pages 353 –358, oct 1993.

[13] Inkscape. An open source linux/windows scalable vector graphics editor, 2010.

[14] Ralf Kohler. A segmentation system based on thresholding. *Computer Graphics and Image Processing*, 15(4):319 – 338, 1981.

[15] Johannes Kopf and Dani Lischinski. Depixelizing pixel art. In *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, pages 99:1–99:8, New York, NY, USA, 2011. ACM.

[16] Yu-Kun Lai, Shi-Min Hu, and Ralph R. Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Trans. Graph.*, 28(3):85:1–85:8, July 2009.

[17] Gregory Lecot and Bruno Levy. Ardeco: Automatic region detection and conversion. In *Eurographics Symposium on Rendering*, 2006.

[18] Musa Mammadov, Alexander Rubinov, and John Yearwood. Dynamical systems described by relational elasticities with applications. *Continuous Optimization*, pages 365–385, 2005.

[19] Musa A Mammadov. A new global optimization algorithm based on dynamical systems approach. In *Proceedings of the 6th International Conference on Optimization: Techniques and Applications (ICOTA' 04)*. Ballarat, Australia, 2004.

[20] University of Ballarat. Ganso library for optimization functions.

[21] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: a vector representation for smooth-shaded images. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 92:1–92:8, New York, NY, USA, 2008. ACM.

[22] Arnulfo Perez and Rafael C. Gonzalez. An iterative thresholding algorithm for image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-9(6):742 –751, nov. 1987.

[23] N. Ramesh, J.-H. Yoo, and I.K. Sethi. Thresholding based on histogram approximation. *Vision, Image and Signal Processing, IEE Proceedings -*, 142(5):271 –279, oct 1995.

[24] Peter Selinger. Potrace: a polygon-based tracing algorithm, 2003.

[25] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[26] H. Tamura. A comparison of line thinning al-

 ISBN 978-80-86943-75-6

gorithms from digital geometry viewpoint. In *Proceedings of the Fourth Int'l Joint Conf Pattern Recognition*. Kyoto, Japan, 1978.

[27] Inc. Vector Magic. Vector magic, 2010.

[28] SVG working group. Svg format for vector graphics.

[29] Tian Xia, Binbin Liao, and Yizhou Yu. Patch-based image vectorization with automatic curvilinear feature alignment. In *ACM SIGGRAPH Asia 2009 papers*, SIGGRAPH Asia '09, pages 115:1–115:10, New York, NY, USA, 2009. ACM.

 ISBN 978-80-86943-75-6

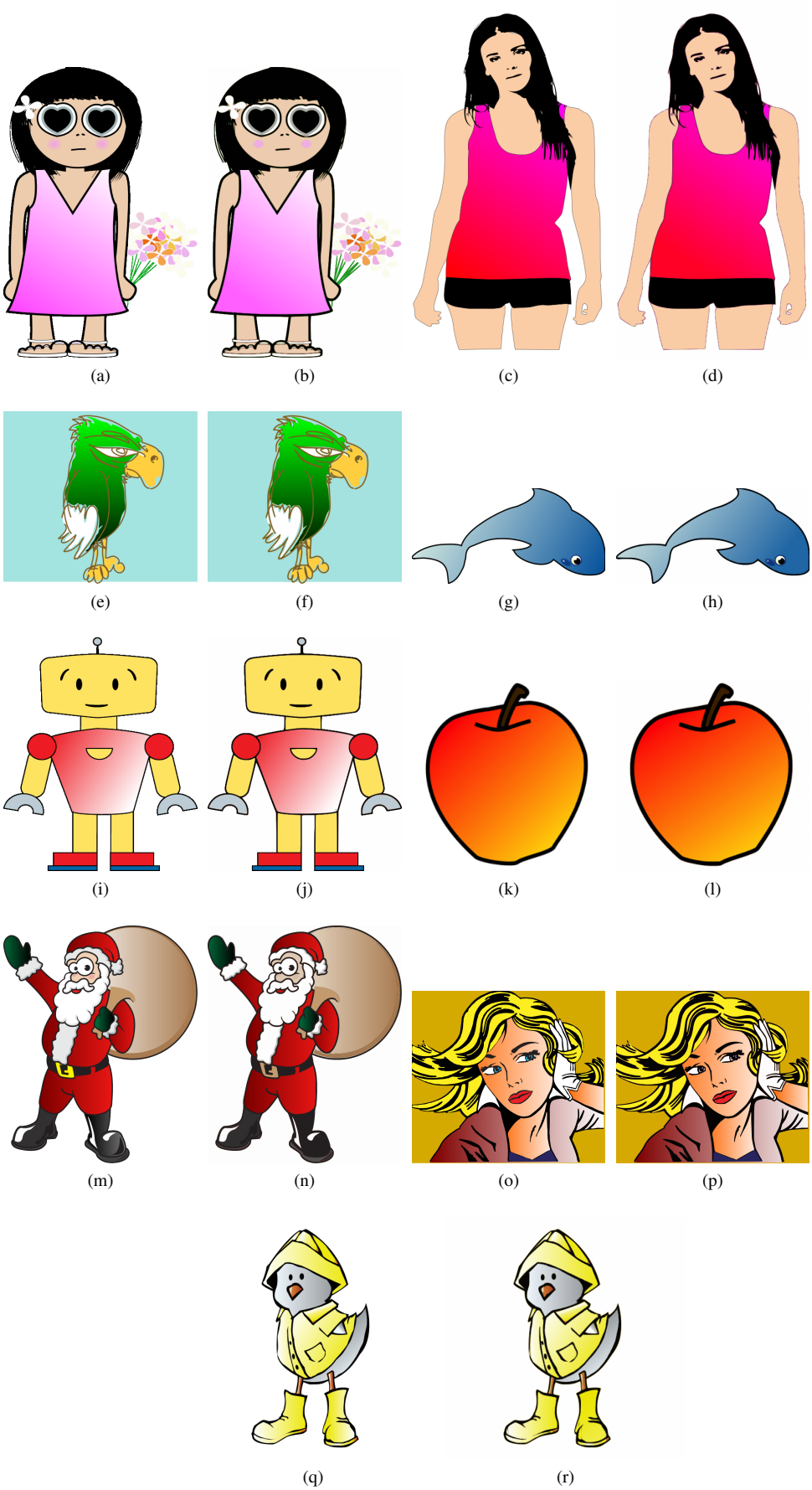
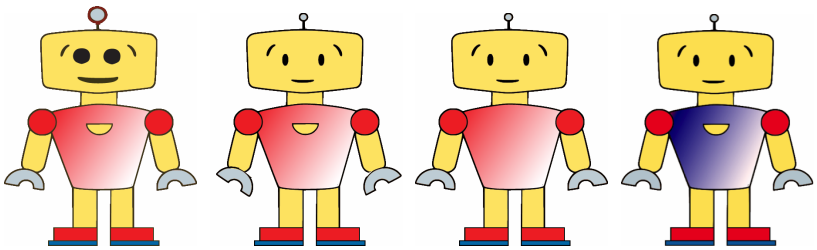


Figure 6: The results with our approach. Original image is on the left and the final vector image is shown on right.

 ISBN 978-80-86943-75-6



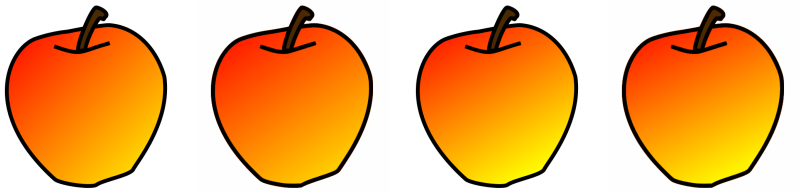
(a) Editing the output vector: Scaled the body parts.

(b) Editing the output vector: Rotated the arm levers.

(c) Editing the output vector: Removed a path.

(d) Editing the output vector: The original linear gradient color stops (as shown in figure 6) were towards red to white. Using Inkscape, we edited the output so that the gradient stops are changed to blue and white.

Figure 7: Editing the final output.

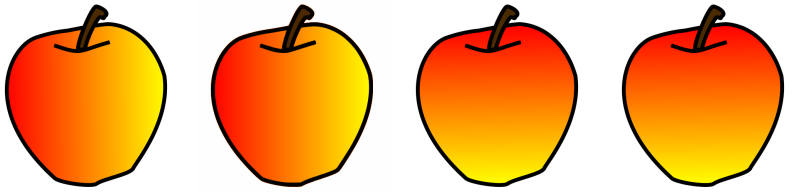


(a) Original gradient direction: 1.0, color varying from (255, 0, 0) to (255, 255, 0).

(b) Computed gradient vary-direction: 1.2, color varying from (255, 4, 0) to (255, 248, 0).

(c) Original gradient colordirection:1.75 , color varying from (255, 0, 0) to (255, 255, 0).

(d) Computed gradient colordirection:1.85 , color varying from (255, 8, 0) to (255, 242, 0).



(e) Original gradient direction: 0, color varying from (255, 0, 0) to (255, 255, 0).

(f) Computed gradient direction: 0, color varying from (255, 2, 0) to (255, 252, 0).

(g) Original gradient direction: ∞, color varying from (255, 0, 0) to (255, 255, 0).

(h) Computed gradient direction: ∞, color varying from (255, 2, 0) to (255, 251, 0).

Figure 8: Comparison of computed gradient with original known gradient in image.

(a) Input noisy Image.

(b) Ardeco output:1200 small patches.

(c) Our output: Only four patches.

Figure 9: Noisy images and comparison with ARDECO.

 ISBN 978-80-86943-75-6