Manga Vectorization and Manipulation with Procedural Simple Screentone

Chih-Yuan Yao, *Member, IEEE*, Shih-Hsuan Hung, Guo-Wei Li, I-Yu Chen, Reza Adhitya, and Yu-Chi Lai, *Member, IEEE*

Abstract—Manga are a popular artistic form around the world, and artists use simple line drawing and screentone to create all kinds of interesting productions. Vectorization is helpful to digitally reproduce these elements for proper content and intention delivery on electronic devices. Therefore, this study aims at transforming scanned Manga to a vector representation for interactive manipulation and real-time rendering with arbitrary resolution. Our system first decomposes the patch into rough Manga elements including possible borders and shading regions using adaptive binarization and screentone detector. We classify detected screentone into simple and complex patterns: our system extracts simple screentone properties for refining screentone borders, estimating lighting, compensating missing strokes inside screentone regions, and later resolution independently rendering with our procedural shaders. Our system treats the others as complex screentone areas and vectorizes them with our proposed line tracer which aims at locating boundaries of all shading regions and polishing all shading borders with the curve-based Gaussian refiner. A user can lay down simple scribbles to cluster Manga elements intuitively for the formation of semantic components, and our system vectorizes these components into shading meshes along with embedded Bézier curves as a unified foundation for consistent manipulation including pattern manipulation, deformation, and lighting addition. Our system can real-time and resolution independently render the shading regions with our procedural shaders and drawing borders with the curve-based shader. For Manga manipulation, the proposed vector representation can be not only magnified without artifacts but also deformed easily to generate interesting results.

Index Terms—Manga, semantic components, vectorization, screentone, procedural shaders

1 Introduction

Comics and Manga are popular medium for conveying ideas and entertainment. Manga, Japanese comics, have gained worldwide audience from not only Asia but also North America and Europe. With advance in mobile technologies, Manga digitization is important for distribution on these devices. However, scanned conversion still dominates digitization and requires manual intervention to clear up noise, artifacts, and distortions. The process is strenuous and time-consuming, and results are incapable to transform to different resolutions after scanned. Additionally, Manga patches in consecutive frames are similar and a mangaka, a cartoon writer in Japanese, can easily manipulate an existing patch to generate a new one after digitization. Therefore, this study attempts to develop an intuitive and interactive Manga vectorization and manipulation system to simplify digitization and adaptation for different reading devices and generate new interesting contents.

Commercial software such as Adobe illustrator and Corel CoreDraw can vectorize Manga patches into livewires vector representations including vector graphics (SVG) and diffusion curves, but the results contain a large number of small screentone element patches, noise, and artifacts. Past research [8], [14] vectorize Western color comics and line drawing arts. However, these algorithms would generate artifacts in vectorizing Manga because of the following difficulties: first, Manga are drawn in black and white, and shading, color, and material information is expressed with different screentone patterns; second, screentone patterns generally contain small-sized elements of different shapes other than Dots and easily mix with borders; finally, scanned versions contain a large number of noise, artifacts, and distortions even when scanning them in a very high resolution. Additionally, these methods represent the color and stroke information independently which may induce inconsistency during deformation. Our system overcomes these difficulties by decomposing a Manga patch into its simple screentone regions, boarders, and shading regions, and developing unified vector representations for them. Therefore, our element decomposition consists of 5 stages including screentone-preserved binarization, screentone detection, simple screentone classification and property extraction, shading region refinement, and border tracing and smoothing. We design these stages to compensate imperfection of each other. In the first, binarization is a hybrid of Otsu [16] and Gaussian local adaption to relieve the issues in screentone detection and pattern recognition caused by scanned noise such as lighting variation, scanning parameters, and paper textures. Screentone detection uses local oscillating variations [2], [19], [28] to identify possible screentone regions, but the detection may miss some shading region and thin strokes without well-defined binarization. Thus, the combination of two stages would discard noise, artifacts, and distortions by representing a screentone pattern in a procedural format of their extracted properties to refine element decomposition and real-time procedurally

• *The authors are with the CSIE, NTUST, Taipei, Taiwan. E-mail: {cyuan.yao, kn810609, kn810, karls820210, azerdarkblade, cheeryuchi}@gmail.com.*

Manuscript received 3 Oct. 2015; revised 3 Jan. 2016; accepted 20 Jan. 2016. Date of publication 4 Feb. 2016; date of current version 4 Jan. 2017. Recommended for acceptance by Y. Yu. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TVCG.2016.2525774

render these elements with arbitrary resolution and compactness. Currently, we evaluate this concept only on procedurally formulating simple screentone patterns including regularly distributed dots, stripes, and grids. Then, our system applies the Local Binary Pattern (LBP) to classify the detected screentone patterns into one of two categories: simple and complex. Our system procedurally handles simple screentone patterns but treats screentone elements of complex patterns as solid shading regions. Furthermore, we also correctly locate white, solid, and screentone shading borders and recover the lighting information as a refinement stage with those procedural properties.

Additionally, manipulation is another goal of digitization but fundamental elements are generally not semantic components such as eyes, hairs, and hands which can be good manipulation components. However, locating contextual components generally requires high-level human recognition, and our system lets users draw scribbles along with intuitive iconized candidates for the formation of desired contextual components. Finally, our system vectorizes each component into a mesh along with embedded Bézier borders for resolution independency and intuitive manipulation. The curve-embedded representation provides a unified mechanic for real-time and consistent manipulation including screentone replacement, shape manipulation, and lighting addition. Our system shades the deformed meshes with a procedural screentone shader and a curve-based shader based on extracted shading information for real-time rendering with arbitrary resolution. We have done the vectorization process in the preprocessing step using several seconds, but done the manipulation and rendering in real time.

To summarize our contributions, this work proposes the effective Manga vectorization and manipulation pipeline of element decomposition, semantic component construction, and vectorization for procedurally shading regularly distributed simple screentone patterns.

- Element decomposition uses a hybrid binarization, screentone detector, screentone identifier, element refiner, line and border tracer, and curve-space Gaussian refiner to robustly decompose raster patches into Manga elements of strokes, solid areas, and screentone shading regions; We develop a screentone-preserved binarization algorithm to make this process more robust and effective against scanned noise and artifacts. Our system separates detected screentone regions into simple and complex patterns according to our collected screentone database using Local Binary Pattern and estimates procedural properties of simple screentone patterns for procedural shading and element refinement.
- We provides an intuitive user interface to decompose elements into semantic components by semi-automatically connecting corners and edges based on user sketches. Then, our system treats these elements or semantic components as unified curve-embedded meshes for resolution independently stroke and screentone rendering and consistent manipulations, such as deformation and virtual lighting effect.

As demonstrated in the results, the vectorized Manga patches are resolution independent and intuitively manipulated to generate interesting deformation and add lighting effects.

2 Related Work

Manga vectorization may involve several different research fields, and the following gives a short discussion of them.

Computational cartoon and comics: There are several research focusing on comics and cartoon digitization. Manga colorization proposed by Qu et al. [19] is considered as a state-of-the-art colorization. They use level-set segmentation and Gabor wavelet encoding along with user scribbles to locate shading regions and select proper colors based on their average intensities. However, their method does not vectorize the screentone pattern for antialiasing, and the segmentation results may bleed out of the borders. Additionally, the algorithm is really slow due to per-pixel-based feature estimation, and the whole process requires manual work for segmentation and color assignment. Qu et al. [18] transform a color image to a bitonal background of screentone patterns for preservation of the tone similarity, texture similarity, and chromatic distinguishability. Although the results can properly map colors to screentone patterns, the algorithm does not aim at vectorizing the screentone patterns for viewing at different devices.

Zhang et al. [30] vectorize 2D raster cartoon animations by segmenting the color images into proper regions based on the detection of boundary and decorative curves. Kopf and Lischinski [8] vectorize Western color comics into black regions and CMY color grids from a colored halftone comics page. Although the results of both methods are satisfactory, their algorithms are not suitable for our case because Manga only comprise of black and white drawings, and their method cannot explicitly extract those screentone regions. Furthermore, although Kopf and Lischinski [8] detect the color grids by their spacing and orientation, their method could not work for multiple dots regions and for other screentone patterns. Moreover, they represent black and color components separately, and the representation induces inconsistency when manipulation. Our representation can provide a consistent and robust manipulation manner to overcome this limitation. Noris et al. [14] vectorize lines, strokes, and pencil drawing to detect lines and faithfully capture the junctions of drawn strokes using topology analysis. However, Manga contain not only lines and strokes but also screentone and solid regions. The algorithm fails to detect and vectorize these shading regions. In addition, all these methods cannot decompose a patch into contextual regions for reasonable manipulation of the contextual components. Although conventional tracing applications such as Illustrator can vectorize a raster image, they require a large amount of manual work to clean up noise, artifacts, and distortions, and they treat screentone patterns as a set of discrete elements which are hard for manipulation. Our algorithm binarizes a scanned patch while preserving the screentone and resolving false positives of screentone detection along with a curve-space Gaussian refiner for holing and bleeding artifacts. Additionally, we also extract pattern properties of regularly distributed simple screentone for later

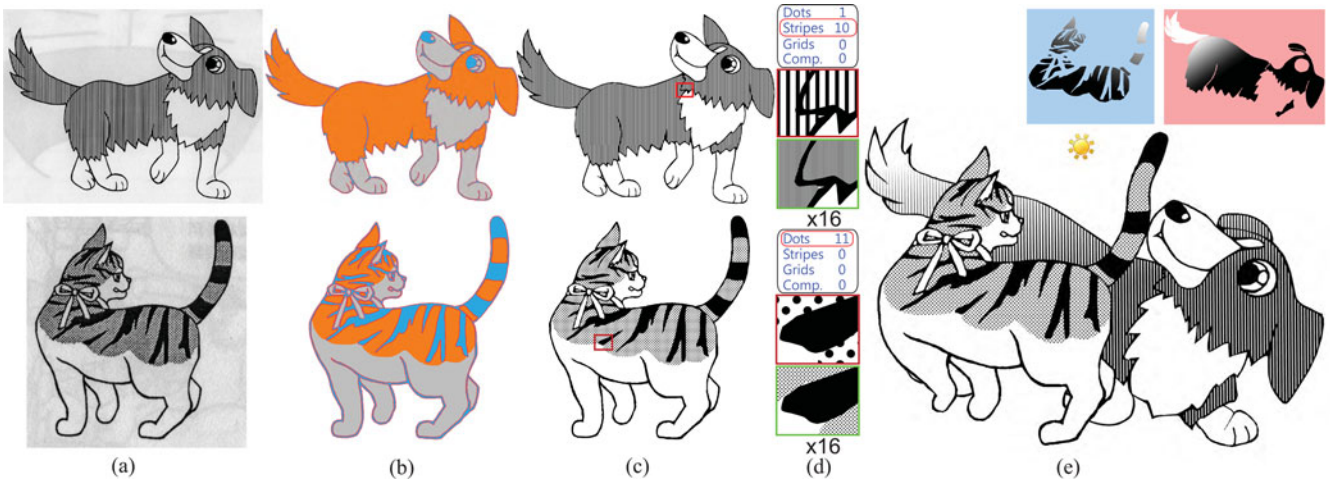

Fig. 1. This illustrates our vectorization and manipulation results with the scanned inputs of a dog and cat ©Pin-Ci Yeh (a), element decomposition results (b) where orange marks the screentone regions, red marks the borders, and blue marks the solid shading regions, vectorization results (c), the LBP voting (d), the resized results at a rate of 16 times when maintaining the size (red box) and the density (green box) of the screentone elements (d), and composition results with manipulation and lighting (g).

procedurally rendering the pattern for arbitrary resolution. Our system also designs a semi-automatic contextual completion method to create Manga components and controls for intuitive manipulation. We also design a GPU-based shader to shade three dominantly seen screentone patterns: Dots, Stripes, and Grids.

Texture-based segmentation: Screentone detection requires texture analysis and modeling. Research [2], [5], [24], [25], [28] decomposes textures using filtering techniques and fit them to statistical models for locating textured regions. We can roughly categorize these locating methods as unsupervised segmentation [6] and supervised segmentation [17]. In addition to the method of automatic detection, Matsui et al. [13] proposes a semi-automatic method to roughly identify where the major characters are and apply for retargeting. However, screentone patterns are simple and can be located by our designed screentone filter without the need of complex textural decomposition as used in these methods. Additionally, these methods do not consider interference of scanned noise and artifacts such as paper texture for precise extraction of the procedural screentone.

Image vectorization: Vector images can fulfill the need of high-definition images and videos. Therefore, photograph vectorization algorithms are proposed to generate vector images from raster images. These methods can be classified into three categories: mesh-based representations [10] encode a vector image with a set of meshes; curve-based methods [4] involve the use of diffusion curves as the color constraints to create a vector image; parametric patch-based representations [9], [27] aim at providing more editability and flexibility. They generally use color-based segmentation to decompose a raster image into patches, select features such as curves and points based on a patch's color distribution, and interpolates these feature for real-time rendering with arbitrary resolution. However, we cannot directly apply these methods for Manga vectorization due to the following limitations. First, Manga contains no color information, and general color-based segmentation does not work to determine these screentone and hatching shading regions. In other words, these fundamental elements must be detected and identified from a macro view. Second, since these patterns are used for shading and texturing, it is important to reproduce them procedurally for maintenance of author intention. The color interpolation cannot maintain the densities, shapes, and other properties of these elemental structures. Third, Manga contain both solid and screentone shading regions, and they should be vectorized into a unified format for consistent manipulation. Therefore, this work focuses on vectorizing Manga patches.

3 Overview

Manga are generally drawn with lines and shading regions in black and white. Furthermore, a mangaka draws lines with different thickness and styles to represent borders, textures, and shading effects, and our system treats them as solid shading regions; a mangaka also expresses shading with black (solid), white, or screentone patterns for colors, textures, and shadows. This study vectorizes Manga patches for intuitive manipulation and real-time rendering with arbitrary resolution, and Figs. 1 and 2 illustrate the intermediate results and procedure. Our system takes a raster-scanned Manga patch as its input and binarizes the patch with our designed method to preserve all screentone regions with little artifacts. Then, our system roughly detects the shading regions by classifying them as solid shading and screentone regions using our adapted screentone detector. Our system classifies screentone patterns into simple and complex by looking up a collected database using Local Binary Pattern and estimates simple screentone properties including the orientation, size of gaps, radius, and thickness for later procedural rendering and element refinement. We treat the screentone elements of the others as solid shading regions. Inside simple screentone shading regions, white strokes and lighting may exist, and thin black strokes may miss. Our system uses the extracted properties to recover these regions for successful decomposition. Our system applies a line tracer for determination of closed borders and smoothes rug and coarse borders based on their curve-based Gaussian distributions. Next, a user draws scribbles to connect corners and edges on the borders to form contextual components. Our system vectorizes the

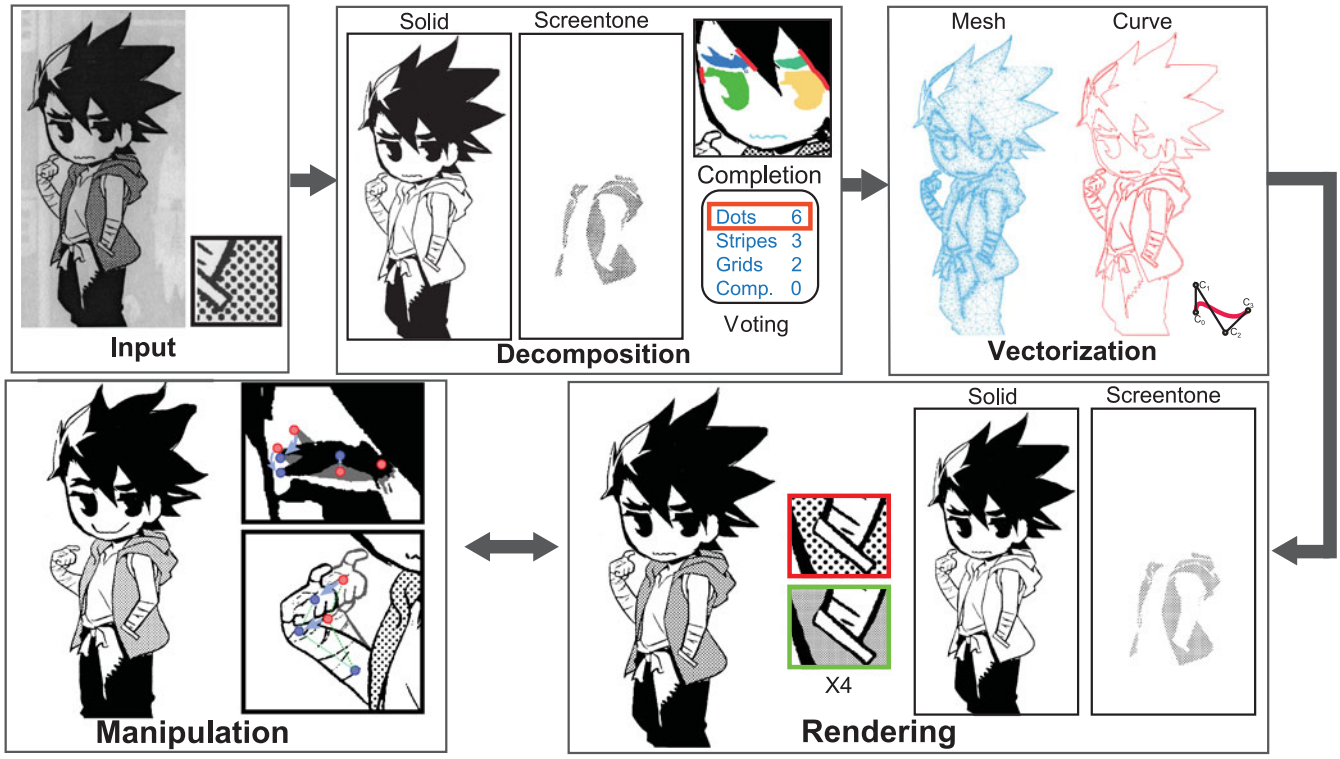

Fig. 2. After loading a scanned patch of Kid from Hero Robot Fight! ©Kurudaz and Iiman, our system first binarizes the patch, detects the Manga elements of lines and shading regions, and extracts their corresponding properties. Then, we determine the contextual components and their controls with user help for component vectorization. The user can manipulate the vectorized components, and our system can resolution independently render them in real time.

borders as oriented Bézier curves and triangulates all Manga components with constraints on all end points of these Bézier curves. Our system embeds the Bézier curves onto the mesh by finding the curve-shading triangles defined as those triangles intersecting with the Bézier curves. After triangulation, our system semi-automatically computes and assign a set of control knobs to intuitively manipulate shading meshes and border curves with Bounded Biharmonic Weights [7]. Later, our system can update the embedded triangles and control information accordingly based on user manipulation for robust and consistent shading and stroke drawing. Finally, our system renders the borders with the curve-based shader proposed by Loop et al. [12] and the interior triangles with a procedural screentone shader. All the shaders run in GPUs, and our system can provide interactive rate of rendering and manipulation after vectorizing the content and placing control points.

4 Manga Element Decomposition and Vectorization

Manga are drawn with lines, strokes, and solid and screentone shading, and thus, our algorithm must first decompose a Manga patch into the corresponding elements. This section describes the details of these decomposition processes as shown in Fig. 3.

4.1 Screentone Detail Preserving Binarization

As shown in Fig. 4, traditional decomposition methods such as [2], [19], [28] are affected by scanned noise such as lighting variation, scanning parameters, and paper textures, and binarization is important for successful screentone detection. Although the Otsu method selects an optimal threshold based on the criterion of maximizing the separability of the resultant classes in gray levels, binarization artifacts as shown in Fig. 5 are still induced especially in the detailed

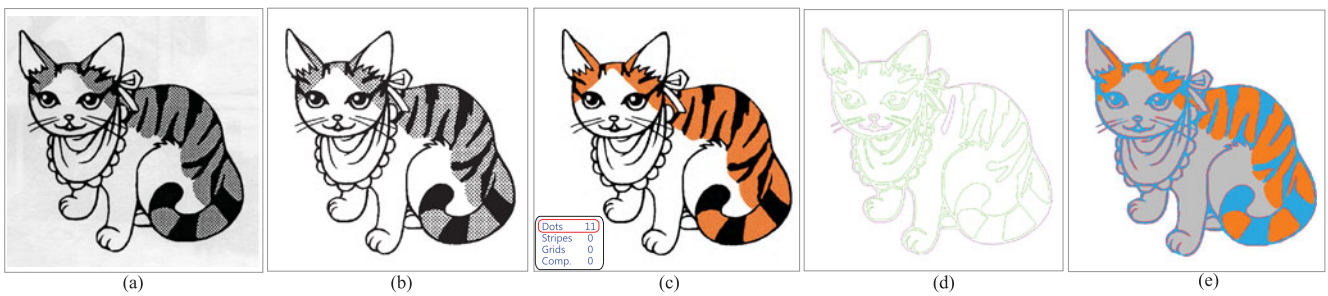

Fig. 3. This shows the intermediate results of element decomposition with the input of a cat ©Pin-Ci Yeh (a) , the binarization result (b), the screentone extraction and identification result (c), the border detection result (d) where green and purple mark the counter clockwise and clockwise borders respectively, and the decomposition result (e).

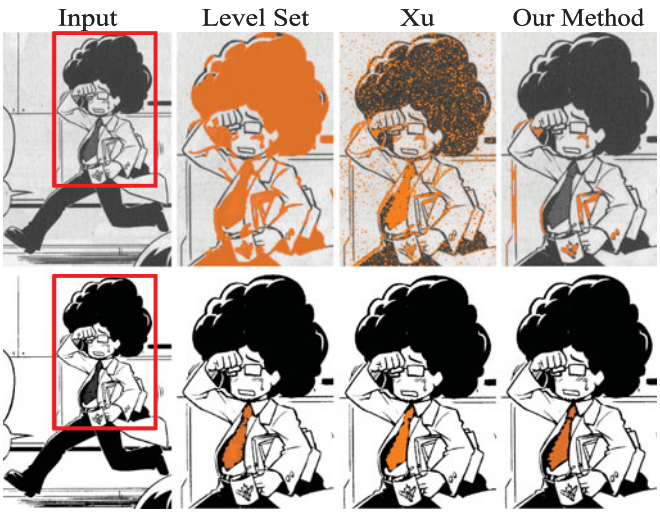

Fig. 4. The first and second rows show the screentone detection results using methods proposed in [2], [19], [28] when inputs are a scanned patch of Doctor from Hero Robot Fight! ©Kurudaz and Iiman and its binarized result respectively.

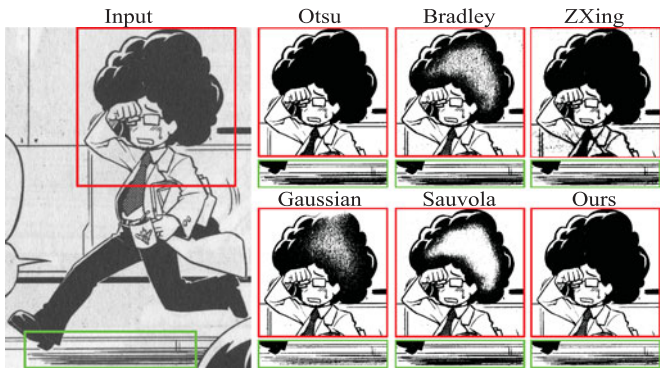

Fig. 5. This shows different binarization results of Doctor from Hero Robot Fight! ©Kurudaz and Iiman when using the Otsu algorithm, the Gaussian local operator, Sauvola's method [22], Bradley's algorithm [1], ZXing's method [21], and our algorithm.

regions because black and white detailed elements blur the scanned sharpness. Local operators such as the Gaussian operator and the adaptive-thresholded methods [1], [22] assume that black and white pixels are roughly evenly distributed and locally estimate thresholds in a neighborhood, but large black and white regions existing in Manga patches may induce false binarization as shown in Fig. 5. Hybrid methods such as ZXing [21] do not aim at binarizing Manga patches to miss the detailed elements as shown in Fig. 5. Generally, the Otsu threshold is good for large white and black patches and the Gaussian operator can successfully preserve the details in the largely detailed regions. Because of those black and white detailed elements, when scanned, their intensities generally fall in the middle of the histogram as the green section in Fig. 6 and can be differentiated when taking local properties into consideration. Therefore, our screentone detail preserving binarization takes these observations into consideration and binarizes the patch in the following steps as shown in Fig. 6. 1) After getting a gray-level patch denoted as $\mathbf{G} = \{\mathbf{G}_0, \ldots, \mathbf{G}_{N-1}\}$ where $\mathbf{G}_i$ is the ith pixel intensity, we compute its histogram, $\mathbf{P} = \{\mathbf{P}_0, \ldots, \mathbf{P}_{255}\}$ where $\mathbf{P}_k$ is the number of pixels whose intensity is equal to k and estimate its global Otsu threshold, T_{Otsu}. 2) We use T_{Otsu} to separate all pixels into two groups, $\mathbf{G}^B$ and $\mathbf{G}^W$, where $\mathbf{G}^B \in \mathbf{G}$ whose element, $\mathbf{G}_k$, is $0 \leq \mathbf{G}_k \leq T_{Otsu}$ and $\mathbf{G}^W = \mathbf{G} - \mathbf{G}^B$ and their corresponding histograms are $\mathbf{P}^B = \{\mathbf{P}_0, \ldots, \mathbf{P}_{T_{Otsu}}\}$ and $\mathbf{P}^W = \{\mathbf{P}_{T_{Otsu}+1}, \ldots, \mathbf{P}_{255}\}$. 3) We determine the maximal number of population, P^B_{max} and P^W_{max}, in $\mathbf{P}^B$ and $\mathbf{P}^W$ respectively. Furthermore, we statistically compute the gray-intensity means, (μ^B, μ^W), and standard deviations, (σ^B, σ^W) for $\mathbf{G}^B$ and $\mathbf{G}^W$. 4) We estimate the upper and lower bounds of the middle section in the histogram by $T_{lower} = P^B_{max} + 2\sigma_B$ and $T_{upper} = P^W_{max} - 2\sigma_W$ because 2σ in each group contains the majority of intensities into both ends. We then put those pixels whose intensity is between T_{lower} and T_{upper} into the Gaussian-operated group, $\mathbf{G}^{Gaussian}$, and the rest as the Otsu-thresholded group,$\mathbf{G}^{Otsu}$. 5) Finally, we apply the Gaussian local operator to all pixels in $\mathbf{G}^{Gaussian}$ and the Otsu threshold to all pixels in $\mathbf{G}^{Otsu}$. Through observation, the detail-determined group selection is important, but little deviation from the optimal group size does not affect the results as shown in Figs. 3b and 5. Due to the length limitation, complete binarization results are shown in supplemental materials and web site.[1]

4.2 Screentone Detection

When analyzing Manga elements in frequency domain, lines, strokes, and solid shading regions contain black shapes which are low-frequency components, and screentone regions are with oscillating patterns corresponding to middle- and high-frequency components which are similar to textural element extraction [2], [28]. Therefore, screentone detection is equivalent to structural decomposition by solving

$$\arg\min_{\mathbf{u}}\{\mathbf{v}^2 + \lambda_{reg}\mathcal{F}(\mathbf{u}, \mathbf{v})\}, \tag{1}$$

where $\mathbf{u}$ is the structural part, i.e. the solid region, $\mathbf{v}$ is the cartoon part, i.e., the screentone region, and $\mathbf{I} = \mathbf{u} + \mathbf{v}$, $\mathcal{F}$ is a regulator function, and λ_{reg} is a user specified parameter. We solve The equation with either the total variation (TV) regulator [20], $\mathcal{F} = |\triangledown\mathbf{u}|$ where $\triangledown$ is the gradient operator, or the relative total variation (RTV) regulator [28], $\mathcal{F} = \frac{\Phi_{\sigma_c}(\mathbf{I})}{\Psi_{\sigma_c}(\mathbf{I})+\epsilon}$ where Φ is local variation function (LTV) [2], [28], Ψ is the overall variation function [28], and σ_c is the size of the Gaussian kernel. RTV generally performs well as a screentone detector as shown in Fig. 4, but it requires solving a large linear system for a long period of time. Buades et al. [2] simplify the linear solution of TV as a nonlinear filtering process for efficiency which is our preferred screentone detector. The total variation of screentone regions decreases very fast under low-pass filtering, and we can estimate the relative reduction rate of LTV as

$$\lambda_{\sigma_c}(\mathbf{I}) = \frac{\Phi_{\sigma_c}(\mathbf{I}) - \Phi_{\sigma_c}(\mathcal{L}(\mathbf{I}))}{\Phi_{\sigma_c}(\mathbf{I})}, \tag{2}$$

where $\mathcal{L}()$ is low-pass filtering, and we chooses a box filter. λ_{σ_c} indicates the local oscillatory behavior and can be

1. web site address: http://graphics.csie.ntust.edu.tw/pub/Manga/

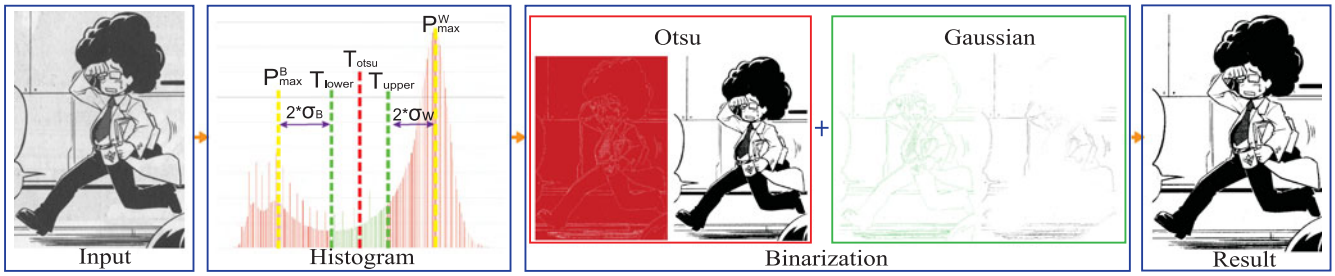

Fig. 6. Our system first separates pixels of Doctor from Hero Robot Fight! ©Kurudaz and Iiman into two groups, $\mathbf{G}^B$ and $\mathbf{G}^W$, based on the estimated Otsu threshold, T_{Otsu}. Then we compute P^W_{max}, P^B_{max}, μ^B, μ^W, σ^B, and σ^W from the histogram of $\mathbf{G}^B$ and $\mathbf{G}^W$ and estimate the upper and lower intensity bounds, T_{lower} and T_{upper}. Our system binarizes those pixels within $[T_{lower}, T_{upper}]$ marked in green using the Gaussian operator, and the rest marked in red using the Otsu operator.

plugged into the weighted average of $\mathbf{I}$ and $\mathcal{L}(\mathbf{I})$ (as described in Eqs. (9) and (10) in the cartoon+texture filter [2]) to decomposes the raster patch into its solid and screentone parts, $\mathbf{u}$ and $\mathbf{v}$. In all our experiments, we set $a_1 = 0.25$ and $a_2 = 0.75$ for the weighting function and $\sigma_c = \frac{1}{2}d$, where d is the distance between two nearest textural elements i.e. the distance between two nearest dots or two nearest lines. Figs. 3c and 4 show exemplar results of screentone detection.

4.3 Pattern Classification

Currently, we aim at procedurally formulating and rendering simple screentone patterns and treating the complex screentone elements as solid shading regions, and therefore, there are mainly two goals for our screentone classifier. 1) to separate detected screentone patterns into simple and complex patterns; 2) to categorize simple patterns into Dots, Stripes, and Grids for property extraction. To achieve these goals, we use Local Binary Pattern histogram proposed by Ojala et al. [15] to search through a collected database for pattern identification as shown in Fig. 7. Although Qu et al. [19] use Gabor filtering for convinced semi-automatic screentone identification, their classification rate generally performs worse than LBP does as shown later in Table 1. This is because some screentone patterns are quite different perceptually, but their Gabor features are too similar to separate. Additionally, automatically identifying proper Gabor features for screentone extraction is also tough, and the computation cost and implementation complexity of Gabor filtering is much higher. We must first create our screentone database which contains 70 Dots, 47 Stripes, and 74 Grids. Our system designs and creates simple patterns of a variety of spacing between neighboring elements, i.e., patterns of different frequencies. Our system computes and stores the LBP histogram of each database pattern when getting them. After locating a screentone region, we compute the LBP histogram of its pattern and compare against the histogram of all database patterns using the chi-square distance, $Distance(\mathbf{H}, \mathbf{J}) = \sum_{i=1}^{n}(h_i - j_i)^2/(h_i + j_i)$ where $\mathbf{H} = \{h_1, \ldots, h_n\}$ and $\mathbf{J} = \{j_1, \ldots, j_n\}$ are LBP histograms for the unknown and database patterns respectively. Then, our system classifies the unknown as follows: 1) When the distance is under T_s, the unknown can be identical to the database pattern, and we mark it as the database one directly. 2) Because if we directly use the one with the shortest distance to label the unknown, false classification happens. Therefore, we design a voting system using the majority type of the N-shortest-distance patterns where N is 11 for this work based on the observation that those patterns with short distances to each other generally have a similar type. However, there exist complex patterns. Because we design our simple screentone patterns to cover contents of most frequencies, it should be thorough, and when the distance is larger than T_o, two patterns are really different, and our system marks those whose distance are over T_o as complex. In other words, our designed simple screentone database patterns should already fulfill our goal of classifying a pattern into simple and complex as shown in Table 1. T_s and T_o are

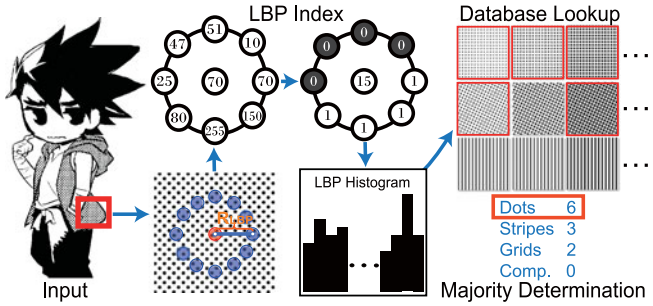

Fig. 7. Our system first selects a screentone pattern, chooses N_{LBP} samples on a circle of a radius R_{LBP}, and uses the number of sampled pixels whose intensity is smaller than the center as its LBP for each pixel. We collect all LBPs of Kid from Hero Robot Fight! ©Kurudaz and Iiman to form the LBP histogram which is normalized with the number of pixels to have a consistent LBP distribution for comparison. Our system uses the histogram to find N-shortest-distance patterns from our collected database and selects the majority type as the pattern type. LBP requires two user specified parameters, the number of samples, N_{LBP}, and the radius, R_{LBP}, where we set them to be 16 and 4 respectively.

TABLE 1
The Statistics for Screentone Pattern Classification for LBP without/with Complex Screentone Patterns, Gabor Filtering (GF), and Gabor Filtering Voting (GFV)

	LBP	LBP (C)	GF	GFV
D.B. Dots	(99, 100)	(99, 100)	(60, 57)	(81, 67)
D.B. Str.	(95, 94)	(95, 92)	(23, 28)	(24, 28)
D.B. Gri.	(99, 98)	(98, 94)	(50, 21)	(44, 29)
Sam. Sim.	(-,98)	(-,97)	(-,52)	(-,59)
Sam. Com.	(-,95)	(-,97)	(-,96)	(-,91)

The first three rows are for dots, stripes, and grids patterns created by our shading algorithm and the two numbers represent the recognition percentage when testing on the digital and scanned formats respectively. The fourth and fifth rows are simple and complex screentone patterns collected from different Manga patches. The time needed for computing LBP, LPB(C), GF, and GFV are averaging 235, 243, 4,713, *and* 4,714 ms *respectively for a* 400×400 *patch.*

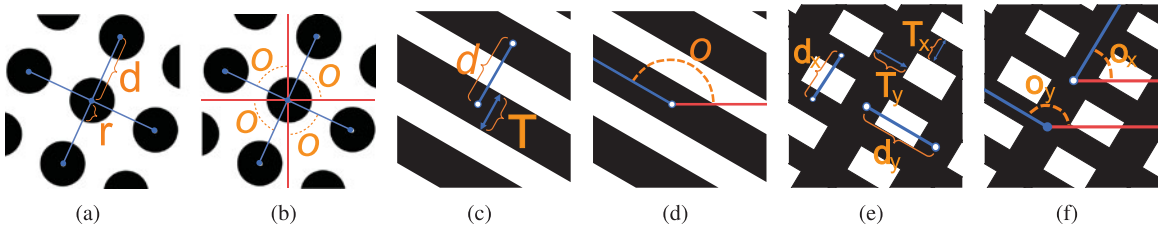

Fig. 8. (a) and (b) illustrate the properties of Dots. (c) and (d) illustrate the properties of Stripes. (e) and (f) illustrate the properties of Grids.

two user specific parameters and we set $T_s = 0.137$ and $T_o = 0.761$ according to our robustness test. Our classifier is invariant to illumination because we compute LBP based on the comparison against its neighbors instead of using absolute intensity. It is scale invariant because we scan patches under the same resolution setting, and our database contains patterns of different frequencies. It is also rotation invariant because when trying to find a match, our algorithm uses the shortest distance to rotated duplications of a pattern as its representative distance. Figs. 1d and 3c show exemplar results of property classification.

4.4 Pattern Property Extraction

After type identification, our system extracts the properties of a screentone pattern for procedurally shading. Before extraction, we apply the screentone-preservation method inside the screentone region for further preservation of the properties against scanning noise and artifacts, again. Currently, this work aims at procedurally formulating the simple screentone patterns: Dots, Stripes, and Grids, and therefore, we define and extract the properties of each type as follows:

Dots denoted as $Dots(\mathbf{d}, \mathbf{o}, r)$: As shown in Fig. 8a and b, $\mathbf{d}$ is the distance between two nearest dots, $\mathbf{o}$ is orientation whose x component defines the rotation, and r is the radius of dots. First, the center of a dot should have the largest distance to the closest white pixel and thus, our system first computes the L_2 distance field of all black pixels to the nearest white pixel and marks the local maximal pixels as centers. Then, our system discards those points close to the borders because they are generally not reliable. The median L_2 of these centers is r. Then, our system builds a KD-tree to find the closest four neighbors to all centers. The median distance of two nearest centers is d and $\mathbf{d} = (d, d)$. The median angle of all neighboring centers relative to the x-axis is $\mathbf{o}_x$ and $\mathbf{o}_y = 0$. Finally, r is normalized with d for later procedural rendering.

Stripes denoted as $Stripes(\mathbf{d}, \mathbf{o}, T)$: As shown in Fig. 8c and d, $\mathbf{d}$ is the distance between two nearest lines, $\mathbf{o}$ is orientation whose x component defines the rotation, and T is line thickness. First, the center of a line should have the largest distance to the closest white pixel, and our system applies a thinning algorithm to the screentone region as skeleton ridges and computes the L_2 distance field of all ridges to the nearest white pixel. The median L_2 of these ridge points is T. Then, our system applies a line fitting algorithm to fit ridge points. The median distance between two neighboring lines is d and $\mathbf{d} = (d, d)$. The median angle of the lines relative to the x-axis is $\mathbf{o}_x$ and $\mathbf{o}_y = 0$. Finally, T is normalized with d.

Grids denoted as $Grids(\mathbf{d}, \mathbf{o}, \mathbf{T})$: As shown in Fig. 8e and f: $\mathbf{d}$ is the distance between two nearest lines in the two sets, $\mathbf{o}$ is orientation which whose x component defines the rotation and y component defines shearing, and $\mathbf{T}$ is the line thickness of the two sets. Similarly, our system applies a thinning algorithm to the screentone region as skeleton ridges and computes the L_2 distance field of all ridges to the nearest white pixel. Our system applies a line fitting algorithm to determine the two sets of lines with two different slopes. The median L_2 of these ridge points for two sets are $\mathbf{T}$. The median angle relative to the x-axis for two sets are $\mathbf{o}$, and the median distance between two neighboring lines of the two sets are $\mathbf{d}$. Finally, $\mathbf{T}$ is normalized with $\mathbf{d}$.

4.5 Element Refinement in Screentone Regions

After screentone detection, the following issues raise: First, mangakas might use varied sizes of screentone elements to express lighting effects, and our system needs them for later procedural rendering. Second, our system may not correctly locate white areas inside or along screentone regions. Third, our system may also miss solid thin strokes inside the screentone regions because of their frequency response. Finally, screentone elements may mix with solid shading regions for induction of rough and rug borders. Therefore, after property extraction, our system uses these properties to relieve these issues in a refinement process as shown in Fig. 9 for all three simple types as follows:

Dots whose elements are distinct without connecting with other elements: 1) We apply connected component labeling (CCL) to each extracted center identified in property extraction. 2) When the area of a connected region is larger than T_{top} set as $r^2 + \sigma_C$ where σ_C is a user specified parameter, our system labels the regions as missing solid strokes. When it is smaller than T_{bottom} which is a user specific parameter, our system labels the regions as noises and removes them. 3) Since printed and scanned offsets and distortions can induce large errors when globally constructing the screentone coordinate, we use the extracted spacing and orientation to locally locate neighboring centers as shown in the top fourth column in Fig. 9. We label a center as an interior one when having all four valid neighboring centers, a solid corner when having least one of its four neighboring centers inside the solid region, and a white corner when missing one or more of its four neighboring centers. 4) We connect the white boundary points for white area borders. 5) Finally, we apply a simple 2D Gaussian smoother on the valid screentone element size to recover the lighting information of the region.

Stripes whose elements are distinct without connecting with other elements: 1) We apply connected component labeling to each extracted central ridge sample. 2) When the area of a connected region is larger than T_{top}, our system labels the regions as missing solid strokes. Our system also removes regions under T_{bottom}. Since stripe lines extends

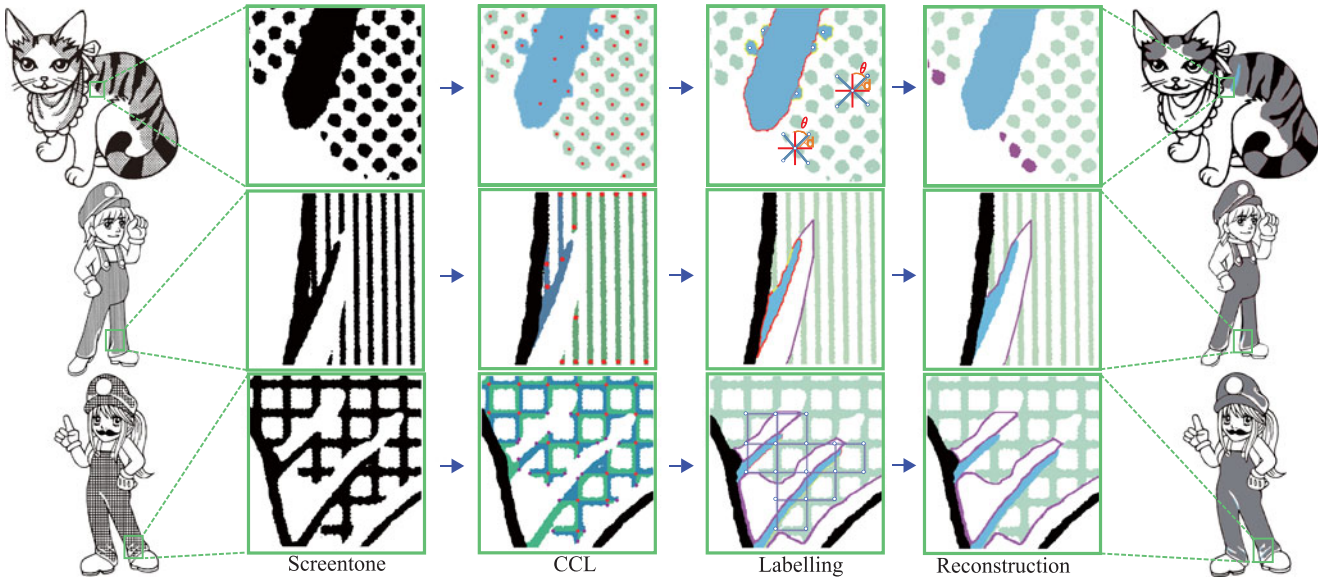

Fig. 9. These three show the element refinement process for Dots, Stripes, and Grids of a cat, princess, and girl ©Pin-Ci Yeh. Our refinement process consists of application of connecting component labeling to extracted centers and ridges, missing stroke identification and white boundary detection, boundary reconstruction, and element size filtering for shading.

until they hit solid and white regions, our system traces effective ridge points to determine these intersections of stripe lines and sets T_{top} as $T * l + \sigma_C$ where l is the effective length when accumulating all valid line segments. 3) Since stripe lines should be connected, when an end point hits a solid/white region, our system marks it as a solid/white border corner as shown in middle fourth column of Fig. 9. 4) We connect the white boundary points for its borders. 5) Finally, we apply a simple 2D Gaussian smoother on the line width of valid ridge samples to recover the lighting information of the region.

Grids whose elements intersect with other elements: 1) We use those identified four neighboring intersections of grid lines to locate a cell and then applying connected component labeling to intersections of this cell. 2) When the area of a connected region is larger than T_{top}, our system labels the regions as a part of a missing solid stroke. T_{top} is $T * 2C + \sigma_C$ where C is the contour length of the cell. Our system also removes regions under T_{bottom}. We use a similar process for determining boundary intersections and the effective length of grid lines. 3) We form missing strokes regions by grouping the neighboring missing-stroke cells. 4) We connect the white boundary points for its borders. 5) Finally, we apply a simple 2D Gaussian smoother on the valid screentone element size to recover the lighting information of the region.

4.6 Solid Shading Region Tracing

After element refinement, our system needs to trace the solid shading regions. Generally, there are two categories of raster tracing methods: contour-based and skeleton-based. Skeleton-based methods such as the one proposed by Buchanan et al. [3] estimates skeletons of shading regions and builds fresh based on skeletons. However, because the fresh shapes depend on the embedded skeletons, we need to have hierarchical deformation and embedment for the fresh to induce manipulation and shading complexity when incorporating with screentone. Contour-based methods such as Potrace [23] vectorize solid regions based on contour detection, and contours can be easily embedded into a mesh for manipulation and screentone shading as shown later. Therefore, we choose contour-based tracing because the detected borders are closed curves and have orientation information for convexity determination of Bézier curves during the rendering process. Our refinement results generally well preserve the features of solid black regions and thus, our system can directly trace their borders. We use the boundary detection method of Potrace [23] to pick a boundary pixel whose right neighbor pixel is white to start tracing a border and choose the tracing direction to form a segment whose black and white pixels are always in its left and right respectively. We construct a border if the tracing returns to the start pixel and closes the curve, and we invert the color of interior pixels. By letting the color filling regions in the left side of a segment, the detected poly-lines are either counter clockwise or clockwise from the start to end vertices. Furthermore, the poly-line for a region with no hole is counter clockwise. The exterior poly-line for a region with holes is counter clockwise and the interior poly-lines along the holes are clockwise. Fig. 3d shows an example.

Generally, the boundary detection results are satisfied, but screentone elements may interferes with solid shading boundaries as shown in Fig. 10a to make the screentone-connected borders rough and rug. As shown in Fig. 10b, although Potrace uses intermediate connected polygons to smooth and refine these borders, these interferences still affect the final results. Therefore, our system tries to remove

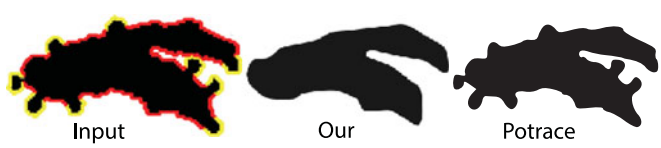

Fig. 10. The contour points (a) where yellow marks those lying on the screentone elements, and red marks the rest are traced to generate the shading border when using Potrace [23] (b) and our algorithm (c) on a missing stroke of a cat ©Pin-Ci Yeh in the vectorization process.

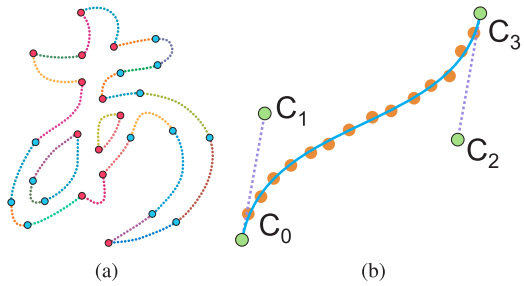

Fig. 11. This shows the progress of border vectorization. (a) We segment borders using RDP where red marks $C(0)$ connection points and blue marks $C(1)$ ones. (b) We construct a cubic Bézier curve marked with blue by fitting points in a segment marked with orange.

these interferences by invalidating those boundary points falling in the screentone elements by using those detected solid corners to identify those centers or ridge samples possibly interfering with solid shading regions, removing all the detected border points falling inside these possible border dots, connecting the rest boundary points as shown in Fig. 10c. These removals may invalidate the plausible formation of intermediate connected polygons of Potrace. Thus, our system further refines and smoothes the borders in twofold: first, we apply a curve-space Gaussian refiner described next to get smoother borders; second, we segment and fit the borders to Bévier curves to remove other high-frequency noise and artifacts in Section 4.7. Our system describes a border as a poly-line in the form of $\overline{\mathbf{P}} = \{\overline{p}_1, \ldots, \overline{p}_n\}$ where $\overline{p}_i = (x_i, y_i)$ is a vertex and $\mathbf{X}$ and $\mathbf{Y}$ denotes the x-axis and y-axis components of $\overline{\mathbf{P}}$ where $\mathbf{X} = \{x_1, \ldots, x_n\}$ and $\mathbf{Y} = \{y_1, \ldots, y_n\}$. The smoother refines a poly-line by convolving $\mathbf{X}$ and $\mathbf{Y}$ with a Gaussian kernel $\mathcal{G}_{\sigma_s}$ as

$$\begin{aligned} \mathbf{X}_s &= \mathcal{G}_{\sigma_s}(\overline{\mathbf{X}}) \\ \mathbf{Y}_s &= \mathcal{G}_{\sigma_s}(\overline{\mathbf{Y}}) \end{aligned}, \tag{3}$$

where $\mathbf{X}_s$ and $\mathbf{Y}_s$ are the x-axis and y-axis components of $\overline{\mathbf{P}}_s$ and $\mathcal{G}_{\sigma_s}$ is the Gaussian smoother with a chosen kernel size of 25 in all our test cases. Since confusion happens around the border, the smoothing operation only applies to those borders which are close to screentone regions i.e., the closest distance to screentone regions is smaller than a threshold which is chosen to be five pixels. Figs. 1b and 3e shows the final decomposition results.

4.7 Manga Vectorization

Although Kopf and Lischinski [8] first propose to vectorize a western color comic, their representation consisting of separate formats for color and black shading may induce inconsistency during manipulation because they deform each component independently. Therefore, we choose to vectorized Manga elements into a unified format of a shading mesh along with a set of embedded border Bézier curves for consistent and robust manipulation, and real-time rendering with an arbitrary resolution. The vectorization process consists of two steps:

Bézier border construction: A single cubic Bézier curve generally cannot represent well the complex borders and therefore, these border poly-lines are partitioned into shorter segments using the Ramer-Douglas-Peucker (RDP) algorithmby selecting the connection points instead of discarding the vertices. After segmentation, we fit a poly-line $\overline{\mathbf{P}} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ to a a cubic Bézier curve, $\mathcal{B}(t) = \overline{c}_0(1-t)^3 + 3\overline{c}_1 t(1-t)^2 + 3\overline{c}_2 t^2(1-t) + \overline{c}_3 t^3$, where $t \rightarrow [0,1]$ is the control parameter, and $\overline{c}_0, \ldots, \overline{c}_3$ are the chosen optimal control points based on $\overline{\mathbf{P}}$. After fitting all poly-lines to a set of Bézier curves, the borders are at least $C(0)$ at the connection points and $C(2)$ elsewhere. Fig. 11 shows the fitting process.

Manga patch triangulation: Since our system plans to render the shading regions with a procedural shader and their borders with the curve-based shader [12], we construct a mesh using Constrained Delaunay Triangulation [29] and embed the control points of all border segments into the vertices of the mesh. Therefore, our system uses the end points of all Bézier segments as constraints for triangulation. After triangulation, our system marks all triangles as one of these types: screentone interior triangles, black solid interior triangles, and curve-shading triangles and shades each type differently.

Later, we show that this representation can be effective, robust, and resolution independent for different manipulation.

5 Rendering

Our system render the vector Manga patch by shading the interior screentone triangles, solid black triangles, and curve-shading triangles consecutively. The following gives the details of our procedural screentone shader and curve-based shader.

5.1 Screentone Procedural Shader

Generally, screentone generally consists of repeated patterns and can be decomposed into the basic components depicted with their space, orientation, radius/thickness, and primitive shape as described in Section 4.4. We shad a pixel black if it locates inside the primitives as shown in Fig. 8. Through observation, we find that the repeated patterns in the shading space can be transformed to align with coordinate axes and normalized to have a uniform space between the primitives as shown in Fig. 12. Finally, our system determines the pixel color based on its type. 1) **Dots**: If the distance to one of 4 corners is smaller than r, the pixel is shaded black. 2) **Stripes**: If $\overline{p}'_x \leq T$ or $\overline{p}'_x \geq 1 - T$, the pixel is shaded black. 3) **Grids**: If $\overline{p}'_x \leq T_x$, $\overline{p}'_x \geq 1 - T_x$, $\overline{p}'_y \leq T_y$, or $\overline{p}'_y \geq 1 - T_y$, the pixel is shaded black. Additionally, our system can easily extend to render other regular repeated primitives such as the star shape shown in Fig. 12 by rendering the primitive shape into four corners of a texture and then use the texture to shade the pattern in the procedure described previously.

5.2 Resolution Independent Bézier Curve Rendering

When rendering Bézier curves, our system uses the curve-based shader proposed by Loop et al. [12] along with the embedded curve information to run the inside/outside test for the color of a pixel. However, after deforming a mesh and its corresponding border, our system dynamically adjusts the embedded triangles and information by intersecting the curve with the mesh and Bézier curves may swap from convex to concave and vice versa as

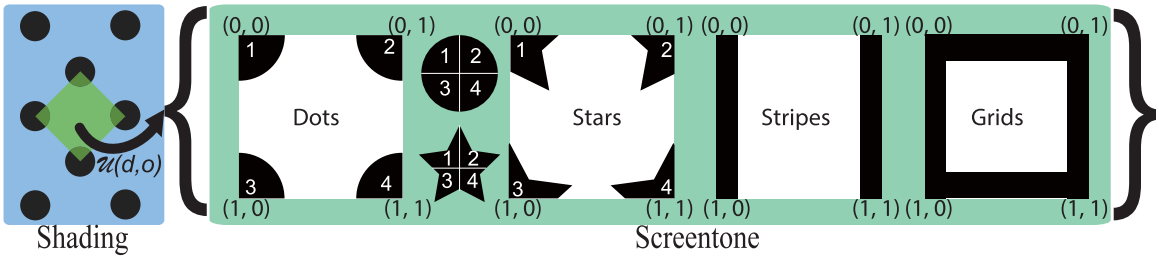

Fig. 12. This illustrates the concept of procedurally screentone shading. The left shows a shading screentone unit, and the right shows the defined Dots, Stars, Stripes, and Grids. Our system transforms a pixel to the screentone space using $\overline{p}' = \mathcal{U}(\mathbf{S}(\mathbf{d})\mathbf{Sh}(\mathbf{o})\mathbf{R}(\mathbf{o})\overline{p})$ where $\mathcal{U}()$ removes the digital part and keeps the decimal part, $\mathbf{Sh}(\mathbf{o}) = [1, -tan(o_y - o_x); 0, 1]$ is the inverse shear operator, $\mathbf{R}(\mathbf{o}) = [cos(o_x), sin(o_x); -sin(o_x), cos(o_x)]$ is the inverse rotation operator, and $\mathbf{S} = [1/d_x, 0; 0, 1/d_y]$ is the inverse scale operator.

shown in Fig. 13a. When swapping, white may fill those pixels outside the border and introduce white artifacts as shown in Fig. 13b. Our system designs a simple approach to solve this problem with an extra frame buffer object (FBO) as shown in Fig. 13c: 1) Our system renders the black interior triangles and Bézier curves into a FBO, and the content inside this FBO should still have white artifacts. 2) Our system passes the FBO to another fragment shader for the removal of all white pixels. 3) Our system shades the corresponding curve-embedded triangles which are next to the screentone regions with the screentone pattern. 4) Our system composites the result of Step 2 with the result of Step 3 using the "over" operation to obtain the final result.

6 Semantic Component Completion and Manipulation

Our system clusters extracted Manga elements into semantic components for intuitive manipulation. Then we assign control knobs and deform the meshes and borders with the bounded biharmonic weighting scheme.

6.1 Semantic Component Completion

In order to intuitively manipulate and render resolution-independent Manga patches, we must further process the Manga elements to build the semantic Manga components such as eyebrows or eyes. Although decomposition with convexity such as the one proposed by Liu et al. [11] can automatically segments a patch into components, they do not have semantic meanings. Fully manual assignment is strenuous, and our system aims at relieving this difficulties with a semi-automatic manner by using simple user scribbles and element selection for intuitive component completion. Since our representation already contains borders, it is easy to design a vector-space intelligent scissor to create enclosing regions with user scribbles in two steps, border segment decomposition and component construction, as shown in Fig. 14.

Border segment decomposition: A user can draw several scribbles to create border segments enclosing semantic components. However, user scribbles are generally imprecise and cannot be directly used to form the correct segments. Therefore, we snap the scribbles to the original curves using non-rigid registration algorithm [31] by building point-to-point correspondences between samples of users scribbles, $\overline{\mathbf{S}}_u$, and original curves, $\overline{\mathbf{S}}_o$. However, since scribbles only indicate local behaviors of borders, our system only chooses scribble segment samples which are close to the original curves. After registration and selection, we create candidate segments from these snapped points and remove those overlapped with original curves. All the remaining scribble-assisted segments along with the original curves are border segments.

Component construction: After border segment decomposition, our system can automatically compute the possible enclosing regions using the point location algorithm [26] to flood fill for acquisition of enclosing borders directly in geometry space based on user clicking. Additionally, the user can also merge two or more regions for acquisition of less to more complex meaningful components.

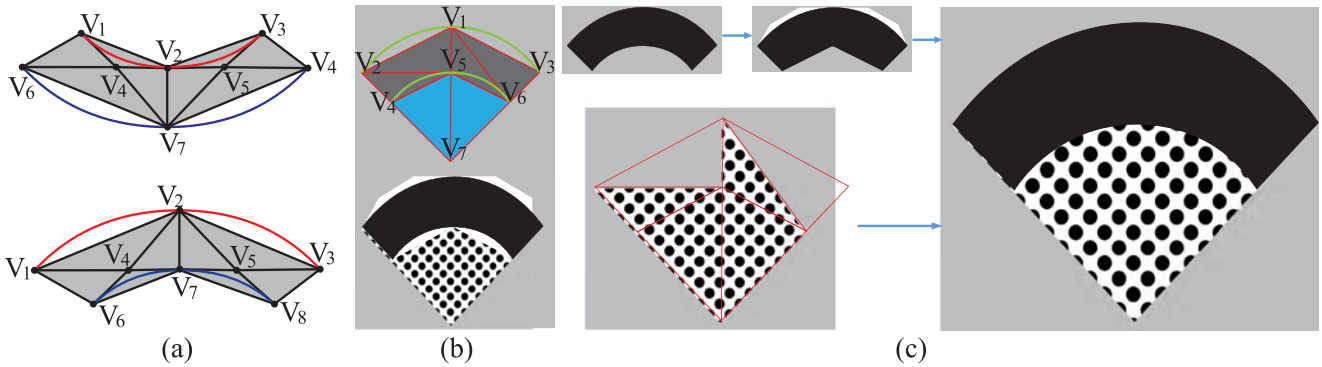

Fig. 13. (a) This illustrate that a convex curve (blue) becomes concave and the concave one (red) becomes convex while their embedded mesh deforms. (b) While directly shading the curve-embedded mesh where blue and gray mark triangles in screentone and solid regions and green marks borders, white artifacts occurring between the border and screentone region. (c) We remove the white artifacts by compositing the rendering of the solid region on a transparent background over the rendering of the screentone region.

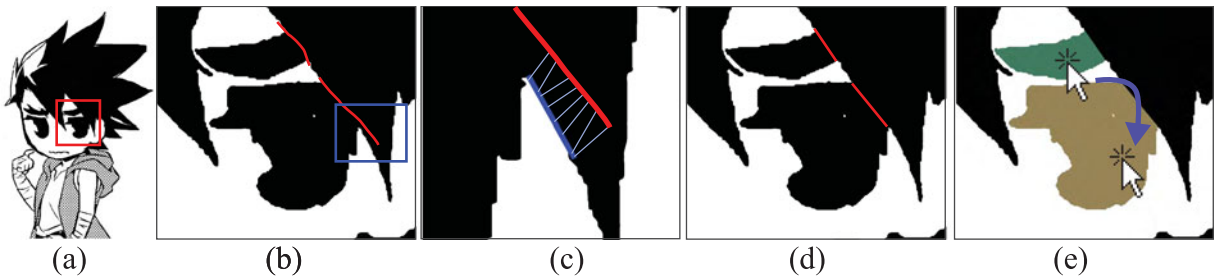

Fig. 14. This shows the progress of component completion with the input vector image of Kid from Hero Robot Fight! ©Kurudaz and Iiman (a), user-specified scribbles to complete the eyebrow (b), non-rigid registration to gain point-to-point correspondences between samples from the original curve (blue) and samples from the scribble (red) (c), the complete border formed by snapping segments (d), and combination of enclosing regions by mouse clicking (e).

Finally, our system also triangulates every semantic component selected with its border to create a separate mesh detached from the original mesh. These newly detached meshes can be manipulated independently.

6.2 Manipulation

After semantic component completion and vectorization, the Manga components are ready for manipulation. In order to reuse the existing patches for production of new Manga patches and cartoons, three manipulation manners are important, and our system can achieve them with a few user scribbles and clicks for simpleness and intuitiveness. The following introduces several interesting techniques to manipulate and enhance the Manga components.

Shading and screentone replacement: A user can change the shading of each semantic component by modifying the screentone type and its properties or selecting a desired pattern from our database. For example, the user can change the radius, space, and orientation of a Dots pattern. Digitization allows authors to reuse their drawing for other purposes by changing their appearance such as their screen patterns, and in the past, this is done by strenuous manual process, but now our system can achieve this by a single click.

Manga component deformation: Our system seeks to deform components with minimal amount of user intervention, the result should be smooth, local, and intuitive for arbitrary topology, and the operation should be minimal after preprocessing steps for real-time operations. Simple control points or bones are abundant and therefore, we adapt the bounded biharmonic weights deformation (BBW) method [7]. After component completion, a user can add a set of control knobs and pseudo edges. This addition is relatively free and intuitive, and the results are completed by BBW. Then, we can define the affine deformation based on the control vertices and bones as $\overline{p}' = \sum_{j=1}^{m} w_j(\overline{p})\mathbf{A}_j\overline{p}$ where $\overline{p}', \overline{p} \in \Omega$ are the deformed and original vertices of the semantic components, $\Omega \in R^2$ denotes the deformation domain, w_j is the weight function associated with handle $\overline{H}_j$, $\overline{H}_j \in \Omega, j = 1, \ldots, m$ are control handles which can be a single point or a skeleton bone represented as a set of connected vertices, and $\mathbf{A}_j$ defines the user specified affine transformation. Our system automatically estimates the weights by minimizing the Laplacian energy subject to upper and lower bound constraints of smoothness, non-negativity, shape-awareness, partition of unity, locality, sparsity, and no local maxima. Since we have a unified representation for both shading regions and strokes, it is easy to maintain their topological relationship for consistent shading and manipulation and smoothness. However, even with a consistent manipulation framework, the curve shading and interior shading may still exist inconsistency. In order to have right rendering results, our system shades Bézier curves and interior shading regions separately. For a single Bézier curve, the start and end points are embedded into the vertices of a curve-shading triangle, and the other control points are designed to have mobility. Under this design, our system can dynamically adjust the convexity of the curve. After deforming component triangles, we must deform the embedded borders according to those curve-embedded vertices. Our designed shading order can guarantee that the screentone rendering do not flow out of the border. Therefore, generally our manipulator is stable and robust. Fig. 15 illustrates the curve updating strategy for the maintenance of $C(1)$. We embed the newly computed control parameters into the vertices. After updating the control parameters of all deformed control vertices, our system can render the border curves and their shading accordingly as described in Section 5. This relieves mangakas from considering the screentone shading problem.

Lighting effect addition: Lighting is important for different effects and conveying different thought. However, it would take a large amount of manual cut-and-paste work for right lighting effects. After vectorizing the screentone, we can easily amplitude modulate our patterns for different lighting effects. Similar to lighting information extraction, we can construct an image mask with color gradation with its minimum and maximum grayscale value g in the range of $[0, 1]$. We use the virtual light to control the direction of linear interpolation for the color gradation. Then our system

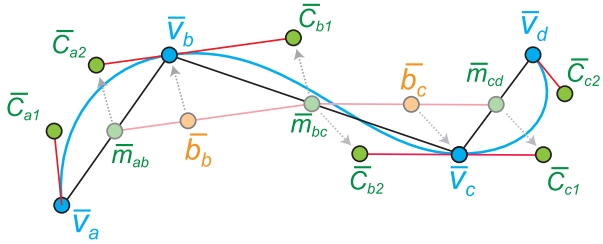

Fig. 15. $\overline{v}_a, \overline{v}_b, \overline{v}_c$, and $\overline{v}_d$ are four consecutive vertices after deformation. $\overline{m}_{ab}$, $\overline{m}_{bc}$ and $\overline{m}_{cd}$ are the middle points of their connected segments. $\overline{b}_b$ and $\overline{b}_c$ are points on the segment $\overline{m}_{ab}\overline{m}_{bc}$ and $\overline{m}_{bc}\overline{m}_{cd}$ having the property of $\frac{|\overline{v}_b - \overline{v}_a|}{|\overline{v}_c - \overline{v}_b|} \propto \frac{|\overline{b}_b - \overline{m}_{ab}|}{|\overline{m}_{bc} - \overline{b}_b|}$ and $\frac{|\overline{v}_c - \overline{v}_b|}{|\overline{v}_d - \overline{v}_c|} \propto \frac{|\overline{b}_c - \overline{m}_{bc}|}{|\overline{m}_{cd} - \overline{b}_c|}$. Finally, we translate $\overline{m}_{ab}$ and $\overline{m}_{bc}$ by $\vec{\overline{b}_b\overline{v}_b}$ to get $\overline{c}_{a2}$ and $\overline{c}_{b1}$ and translate $\overline{m}_{bc}$ and $\overline{m}_{cd}$ by $\vec{\overline{b}_c\overline{v}_c}$ to get $\overline{c}_{b2}$ and $\overline{c}_{c1}$.

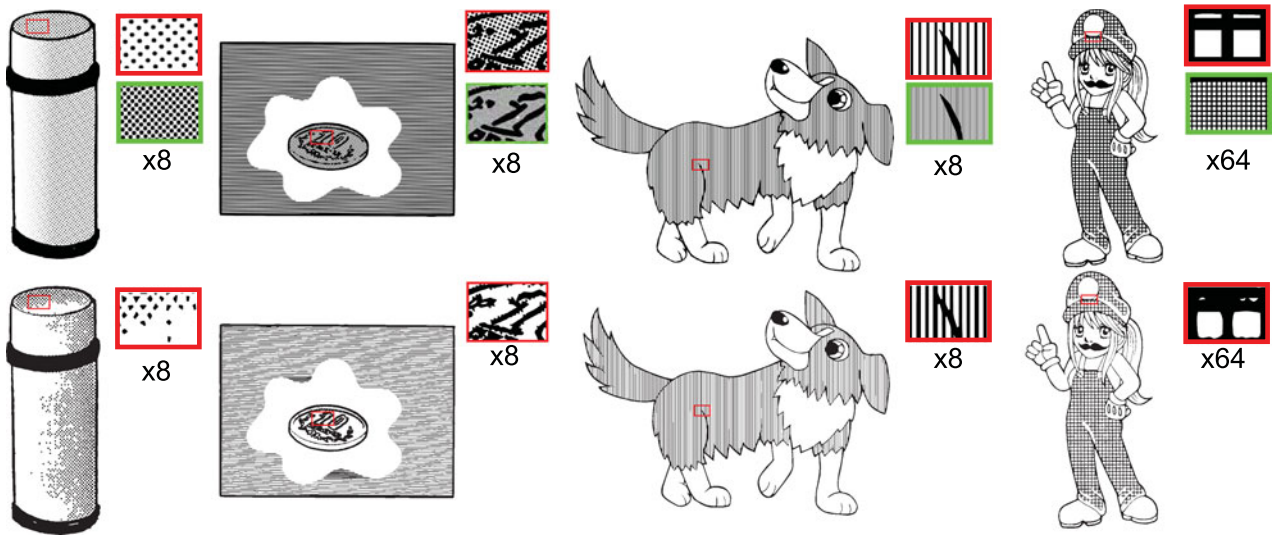

Fig. 16. This shows the vectorization results of a bottle and coin from Hero Robot Fight! ©Kurudaz and Iiman, and a dog and girl ©Pin-Ci Yeh generated by our algorithm (first row) and Adobe Illustrator (second row) respectively where green and red marks the resized results when rendering to maintain the primitive density and size of the screentone pattern respectively, and the number under the figure denotes the resized rate.

uses the color gradation of a pixel to modify the radius and thickness properties in the procedural shader by multiplication i.e., $r = r \times g$. Fig. 1e shows exemplar results.

7 Results and Comparison

Our vectorization pipeline uses a simple LBP pattern identifier to determine the pattern type. In order to use it, we have constructed a database of totally 70 Dots, 47 Stripes, 74 Grids generated with different properties by our procedural shader discussed in Section 5. Additionally, we would like to know whether extra complex patterns can help us more precisely separate patterns into simple and complex. We collect 52 complex patterns from public web sites and commercially available screentone sheets. We test LBP classifier using our screentone pattern database without/with complex patterns. Additionally, we collect the Gabor filtering code from Qu et al. [19] for a GF classifier of 24 features. We also add a similar voting system into the Gabor filtering for a GPV classifier. In order to verify its robustness, we have designed two different tests: generated pattern comparison and collected realistic pattern comparison. First, we used our procedural shader to generate a set of regular screentone patterns of different spacing, radii, and orientation (totally 135 Dots, 125 Stripes, 128 Grids) which are different from those in the database. We use these patterns to test the LBP, GF, and GFV classifiers. Furthermore, in order to get more close to the realistic situation, we design a method to print out the screentone on Manga-used papers and the printed out contents are scanned in for testing the robustness of these identifiers. The results are shown in the first three rows in Table 1. Second, we have collected 34 regularly distributed simple screentone patterns in Dots, Stripes, and Grids format from 29 patches of different Manga and totally 67 complex screentone patterns cut out from different scanned Manga pages and commercially available print-outs. We have also tested the identifier on the collected data sets, and the statistics are shown in the last two rows in Table 1. As results show, when comparing the performance of LBP identification without/with complex patterns, we find that although complex patterns can enhance the detection rate of an unknown complex pattern, they disturb the detection process of an unknown simple screentone pattern. Thus, in all our exemplar cases, we only use the database containing only simple screentone patterns. Furthermore, we have found that LBP can highly outperform GF and GFV in identifying an unknown simple screentone pattern in created and collected patterns. Its performance in detecting an unknown complex pattern is comparative to GF and GFV. Additionally, the time for LBP to compute the features for a pattern is much faster than GF and GFV.

We have tested our element decomposition algorithm on the same 27 Manga characters and patches to understand its performance. Figs. 1b and 3e shows two of the decomposition results. Generally, our algorithm performs efficiently in determining high density regular patterns with precise identification. Additionally, our algorithm can also successfully identify white areas and lighting effects inside screentone regions and reproduce them procedurally. For those thin strokes, screentone detection may mis-classify them as screentone, and our system can successfully recover them. Figs. 1 and 16 show the results of applying our intuitive Manga patch vectorization and manipulation methods for several Manga objects drawn by different artists to show the effectiveness of our algorithm. Our method can easily vectorize a Manga patch in a different style into the form of a mesh and Bézier borders and interesting deformation and lighting effects can be added for desired effects. Generally, screentone is an important element for Manga, and screentone patterns always maintain a similar structure and scale across rendering resolution. However, it differs from general structural texture elements embedding fine details in the image. Adobe illustrator is a powerful vectorization tool and can vectorize a raster image. Therefore, we have compared our vectorized results with those generate by Adobe Illustrator as shown in Fig. 16. Because Illustrator treats screentone patterns as a set of discrete elements, screentone elements lose their regular structure if there is no manual work to clean up noise, artifacts, and distortions. In the contrary, our proposed algorithm successfully extract most regular screentone patterns in Manga and transfer them to a

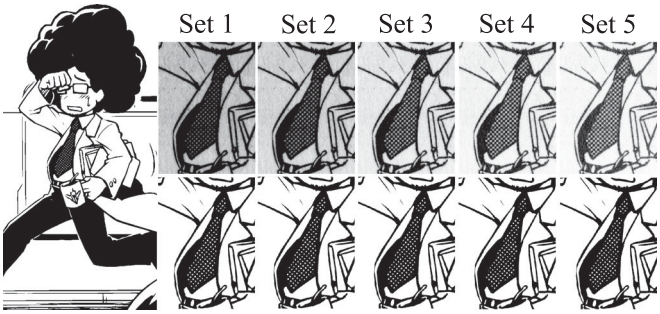

Fig. 17. This shows the vectorization results of Doctor from Hero Robot Fight! ©Kurudaz and Iiman while the input are scanned raster patches using different scanning parameters.

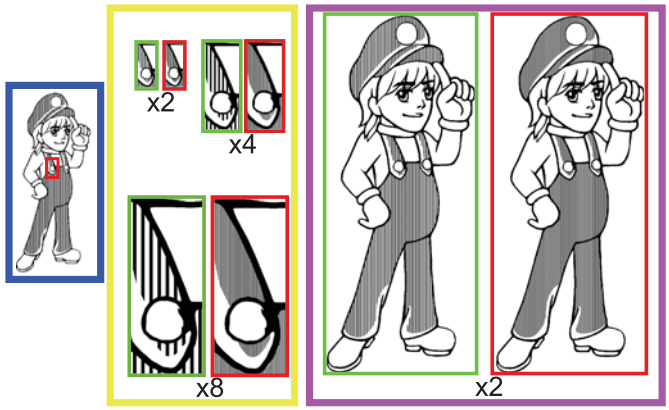

Fig. 18. This shows the resized results of our vectorized princess ©Pin-Ci Yeh with the original size (blue box), the resized highlighted region at a rate of 2, 4, and 8 times (yellow box), and the resized Manga patch at a rate of 2 times (purple box).

parametric form for our procedural shader. Additionally, when zooming-in, the results generated by Illustrator cannot maintain the primitive density of a screentone and our algorithm can easily maintain the density. Scanned parameters may affect the results and induce different amount of scanned noise and artifacts to interfere screentone detection and property extraction. In order to test the robustness and effectiveness of our vectorization algorithm, we have tested the algorithm on 13 Manga characters in five different set of scanning parameters. Fig. 17 shows the vectorization results, and our detector work well for all five scanning parameters. Furthermore, we have run our vectorization algorithm on a set of totally 111 Manga panels of the first four pages from Doraemon (Bring Giant up to be a good kid and Future chocolate car) and Black Jack (Unfinished house and The policeman and mannequin). Then, we have binarized these panels and manually removed the screentone region from these panels to act as ground truth results for computing the correct rate of black and white pixel detection respectively as 96.5 and 99.7 percent.

Additionally, our supplemental material and web site[1] also provides the detection results when using the levelset detector [19] and RTV [28]. We conclude that after proper binarization, these methods can provide reasonable means to detect the screentone regions.

Fig. 18 shows that a user can magnify our vector Manga patches with clear and distinct features without blurring and aliasing artifacts. Additionally, after vectorization, our system can easily generate interesting composition of Manga objects as in the sixth column of Fig. 1. Due to the length limitation, complete data and results in this session can be found in Supplemental materials and web site.[1]

8 Discussion and Conclusion

This study proposes a Manga object vectorization system with a pipeline of a binarizer, screentone detector, LBP screentone classifier, element refiner, line tracer, semi-automatic semantic composer, and vectorizer. After vectorizing a Manga patch, we also design several manipulation methods to generate interesting effects. Later, our designed screentone procedural shader and curve-based shader can render the vectorized components in arbitrary resolution. Our proposed vector representation has the following two advantages: first, the vectorized Manga have an arbitrary resolution and a small file size when comparing to their raster version; second, the vectorized Manga are easier to edit and manipulate especially in deforming borders and shading regions and modifying screentone patterns, screentone properties, and lighting. Additionally, our designed stages are flexible for easy replacement in the future for other more effective methods. Although the results demonstrate that our pipeline is effective for vectorizing our examples, these stages are not without limitations, and therefore, there are a few future research directions. First, although we have properly tuned our screentone detector for all our cases, it is based on local oscillating variations and has robustness issues for screentone patterns with a wide variety of frequencies. We would like to develop a multi-level screentone detector or other types of structural detectors for more robust detection. Second, our system only procedurally shades regularly distributed simple screentone but handles complex screentone including regularly distributed complex screentone, irregularly distributed complex screentone, random complex screentone, and hatching screentone as solid shading regions. However, mangakas usually use these patterns for expressing lighting effects and material textures. We would like to develop a recognition procedure along with a procedural shader to generate these aesthetic shading effects. Third, our LBP identifier is good at separating patterns into simple and complex categories but it cannot correctly identify the exact pattern from the database. For simple patterns, this is fine to extract their properties based on geometric operations. However, it is problematic for complex patterns, and we would like to develop a Gabor-based identifier for complex screentone patterns for avoiding property extraction. Fourth, contextual component completion still requires a large amount of manual work. More automatic shape decomposition with less user interaction similar to grab cut is desired. Lastly, although our system can achieve interesting manipulation, it should add certain strokes for better concept conveying. For example, a mangaka can manipulate a stretching hand to become a banding hand, and he/she generally adds a few wrinkles onto the clothes, but currently our system cannot achieve these effects. In the future, we would like to collect a set of examples and develop a machine learning mechanism to automatically add these extra strokes after deformation. Another idea is to add 3D deformation effects by using 3D object proxies of a Manga object along with cloth simulation.

ACKNOWLEDGMENTS

We thank Pin-Ci Yeh and International Games System (IGS) to provide us the Manga patches for study, participants of all user studies, and the support of IGS Inc. This work was also supported by NSC-104-2221-E-011-029-MY3, NSC103-2221-E-011-114-MY2, NSC-104-2218-E-011-006, NSC-103-2218-E-011-014, NSC-104-2221-E-011-092 and NSC-103-2221-E-011-076, Taiwan. This work was supported by Ministry of Science and Technology of Taiwan under Grants NSC-104-2221-E-011-029-MY3, NSC103-2221-E-011-114-MY2, NSC-104-2218-E-011-006, NSC-103-2218-E-011-014, NSC-104-2221-E-011-092 and NSC-103-2221-E-011-076. Yu-Chi Lai is the corresponding author.

REFERENCES

[1] D. Bradley and G. Roth, "Adaptive thresholding using the integral image," *J. Graph., GPU, Game Tools*, vol. 12, no. 2, pp. 13–21, 2007.
[2] A. Buades, T. Le, J. M. Morel, and L. Vese, "Fast cartoon + texture image filters," *IEEE Trans. Image Process.*, vol. 19, no. 8, pp. 1978–1986, Aug. 2010.
[3] P. Buchanan, M. Doggett, and R. Mukundan, "Structural vectorization of raster images," in *Proc. 27th Conf. Image Vis. Comput.*, 2012, pp. 319–324.
[4] M. Finch, J. Snyder, and H. Hoppe, "Freeform vector graphics with controlled thin-plate splines," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 166:1–166:10, 2011.
[5] M. Galun, E. Sharon, R. Basri, and A. Brandt, "Texture segmentation by multiscale aggregation of filter responses and shape elements," in *Proc. 9th IEEE Int. Conf. Comput. Vis.*, 2003, pp. 716–723.
[6] T. Hofmann, J. Puzicha, and J. Buhmann, "Unsupervised texture segmentation in a deterministic annealing framework," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 803–818, Aug. 1998.
[7] A. Jacobson, I. Baran, J. Popović, and O. Sorkine, "Bounded biharmonic weights for real-time deformation," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 78:1–78:8, 2011.
[8] J. Kopf and D. Lischinski, "Digital reconstruction of halftoned color comics," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 140:1–140:10, Nov. 2012.
[9] Y.-K. Lai, S.-M. Hu, and R. R. Martin, "Automatic and topology-preserving gradient mesh generation for image vectorization," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 85:1–85:8, 2009.
[10] Z. Liao, H. Hoppe, D. Forsyth, and Y. Yu, "A subdivision-based representation for vector image editing," *IEEE Transactions on Vis. Comput. Graph.*, vol. 18, no. 11, pp. 1858–1867, 2012.
[11] H. Liu, W. Liu, and L. Latecki, "Convex shape decomposition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, Jun. 2010, pp. 97–104.
[12] C. Loop and J. Blinn, "Resolution independent curve rendering using programmable graphics hardware," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1000–1009, Jul. 2005.
[13] Y. Matsui, T. Yamasaki, and K. Aizawa, "Interactive manga retargeting," in *Proc. ACM SIGGRAPH Posters*, 2011, pp. 35:1–35:1.
[14] G. Noris, A. Hornung, R. W. Sumner, M. Simmons, and M. Gross, "Topology-driven vectorization of clean line drawings," *ACM Trans. Graph.*, vol. 32, no. 1, pp. 4:1–4:11, Feb. 2013.
[15] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern Recog.*, vol. 29, no. 1, pp. 51–59, 1996.
[16] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. Syst., Man Cybern.*, vol. 9, no. 1, pp. 62–66, Jan. 1979.
[17] N. Paragios and R. Deriche, "Geodesic active regions for supervised texture segmentation," in*Proc. 7th IEEE Int. Conf. Comput. Vis.* 1999, vol. 2, pp. 926–932.
[18] Y. Qu, W.-M. Pang, T.-T. Wong, and P.-A. Heng, "Richness-preserving manga screening," *ACM Trans. Graph.*, vol. 27, no. 5, pp. 155:1–155:8, Dec. 2008.
[19] Y. Qu, T.-T. Wong, and P.-A. Heng, "Manga colorization," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1214–1220, Jul. 2006.
[20] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenom.*, vol. 60, no. 1, pp. 259–268, 1992.
[21] P. Sánchez, "Android application (zxing) validation using hybrid simulation (ubiksim)," Master's thesis, Univ. Murcia (UMU), Murcia, Spain, 2011.
[22] J. Sauvola and M. Pietiknen, "Adaptive document image binarization," *Pattern Recog.*, vol. 33, no. 2, pp. 225–236, 2000.
[23] P. Selinger. (2003). Potrace: A polygon-based tracing algorithm [Online]. Available: http://potrace.sourceforge.net
[24] D. Sýkora, J. Buriánek, and J. Žára, "Unsupervised colorization of black-and-white cartoons," in *Proc. 3rd Int. Symp. Non-photorealistic Animation Rendering*, 2004, pp. 121–127.
[25] M. Varma and A. Zisserman, "Texture classification: Are filter banks necessary?," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recog.*, Jun. 2003, vol. 2, pp. II–691–698.
[26] R. Wein, E. Berberich, E. Fogel, D. Halperin, M. Hemmer, O. Salzman, and B. Zukerman, "2D arrangements," in *CGAL User and Reference Manual*. CGAL Editorial Board, 4.4 ed., 2000.
[27] T. Xia, B. Liao, and Y. Yu, "Patch-based image vectorization with automatic curvilinear feature alignment," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 115:1–115:10, 2009.
[28] L. Xu, Q. Yan, Y. Xia, and J. Jia, "Structure extraction from texture via relative total variation," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 139:1–139:10, Nov. 2012.
[29] M. Yvinec, "2D triangulations," in *CGAL User and Reference Manual*, CGAL Editorial Board, 4.3 ed., 2013.
[30] S.-H. Zhang, T. Chen, Y.-F. Zhang, S.-M. Hu, and R. Martin, "Vectorizing cartoon animations," *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 4, pp. 618–629, Jul. 2009.
[31] Y. Zheng and D. Doermann, "Robust point matching for nonrigid shapes by preserving local neighborhood structures," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 643–649, Apr. 2006.

Chih-Yuan Yao received the MS and PhD degrees in computer science and information engineering from the National Cheng-Kung University, Taiwan, in 2003 and 2010, respectively. He is an assistant professor in the Department of Computer Science and Information Engineering at the National Taiwan University of Science and Technology (NTUST), Taipei, Taiwan. His research interest is computer graphics, including mesh processing and modeling, and non-photorealistic rendering (NPR). He is a member of the IEEE.

Shih-Hsuan Hung received the BS degree from the NTUST, in 2014, in Department of Computer Science and Information Engineering. He is a currently working toward the MS degree at the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology (NTUST), Taipei, Taiwan. His research interests are computer graphics and multimedia.

Guo-Wei Lee received the BS degree from the Department of Mathematics, Tatung University, Taipei, R.O.C., in 2013. He is currently a student in NTUST at the Department of Computer Science and Information Engineering and his research interests are in the area of graphics, vision, and multimedia.

I-Yu Chen is a senior student in computer science and information engineering at the NTUST. He is interested in computer graphics and computer vision.

Reza Adhitya is Indonesian and received the undergraduate degree in informatics from the Department in Sepuluh Nopember, Institute of Technology, and the master's degree from the Computer Science and Information Engineering, National Taiwan University of Science and Technology (NTUST). He is currently working toward the PhD degree and is interested in computer graphics and game technologies in the University of Waterloo.

Yu-Chi Lai received the BS degree from the Electrical Engineering Department, National Taiwan University, Taipei, R.O.C., in 1996. He received the MS and PhD degrees from the University of Wisconsin Madison, in 2003 and 2009, respectively, in electrical and computer engineering and the MS and PhD degrees, in 2004 and 2010, respectively, in computer science. He is currently an associate professor in the NTUST and his research interests are in the area of graphics, vision, and multimedia. He is a member of the IEEE.