

Vectorizing Cartoon Animations

Song-Hai Zhang¹ Tao Chen¹ Yi-Fei Zhang¹ Shi-Min Hu¹ Ralph R. Martin²

¹Tsinghua University ²Cardiff University

Abstract—We present a system for vectorizing 2D raster format cartoon animations. The output animations are visually flicker free, smaller in file size, and easy to edit. We identify decorative lines separately from colored regions. We use an accurate and semantically meaningful image decomposition algorithm, supporting an arbitrary color model for each region. To ensure temporal coherence in the output, we reconstruct a universal background for all frames, and separately extract foreground regions. Simple user-assistance is required to complete the background. Each region and decorative line is vectorized and stored together with their motions from frame to frame.

The contributions of this paper are: (i) the new trapped-ball segmentation method, which is fast, supports non-uniformly colored regions, and allows robust region segmentation even in the presence of imperfectly linked region edges, (ii) the separate handling of decorative lines as special objects during image decomposition, avoiding results containing multiple short, thin over-segmented regions, and (iii) extraction of a single patch-based background for all frames, which provides a basis for consistent, flicker-free animations.

Index Terms—Cartoon vectorization, Trapped-ball method, Image decomposition, Foreground extraction

I. INTRODUCTION

Cartoon animation, i.e. cel animation, has a long history, resulting in a large body of artistic work. Children and adults enjoy the stylized characters, the drawing and animation styles, plots, and soundtracks. Currently, such cultural heritage is preserved and disseminated as digital video after conversion from film using telecine. Such video is in raster format.

However, animated 2D cartoons have an intrinsically vector nature. Cartoon making has a relatively fixed composition procedure, including an unchanging but moving background, and an animated foreground. Compared to real-world videos, 2D cartoon animations have simpler, more artificial contents, composed of regions with simple coloring, and wide decorative lines, as shown for example in Fig. 2.

This particular nature of 2D cartoons means that there are potential advantages in converting them to a meaningful vector format, which:

- will generally allow higher compression ratios for storage and transmission than raster format, because of the small number of regions which can be described by simple color models,
- is resolution independent, readily allowing display on devices with differing capabilities,
- readily allows the cartoon to be edited, e.g. to relocate a character in a scene, or to add an object in front of some existing objects and behind others,



Fig. 1. left column: Original videos; right column: Vectorization results showing region boundaries and decorative lines

- offers advantages for multimedia information retrieval, allowing search for objects of a particular shape,
- avoids the undesirable artifacts caused by lossy raster compression, instead providing strongly colored regions with strong edges, better suited to the artistic style of cartoons. In turn, this allows better quality video to be transmitted with limited bandwidth, and also has potential applications to cartoon restoration.

In recent years, various approaches have been proposed for vectorizing rasterized *images* [1], [2], and commercial packages exist, e.g. Stanford VectorMagic. However, these do not take into account the particular features of cartoon animations, and in particular tend to suffer from oversegmentation. Furthermore, none of them considers temporal coherence. Sýkora [3]–[5] made significant advances in vectorizing car-



Fig. 2. Cartoon gallery

toons, but his approach relies on a particular drawing style in which meaningful regions are enclosed by thick bounding lines, which is not appropriate to many modern cartoon styles.

We note that specific challenges exist when performing 2D cartoon vectorization. The simplicity of the cartoon's contents can at times be a disadvantage as well as an advantage. Flicker is more noticeable in simple cartoon scenes than in real-world videos, and strongly decreases the visual quality. Thus, accurate boundaries and locations of segmented regions are necessary to achieve visual temporal coherence. Segmentation should be semantically meaningful in terms of the color regions and decorative lines output. We take into account the particular nature of cartoons to solve these problems.

Edge information provides a strong hint for decomposition of cartoon animations, but extracted edges often contain gaps and cannot always be reliably joined. We overcome this problem by using a novel trapped-ball method to stably segment each frame into regions. This process is guided by edges, but can cope with gaps in the edges. The color in each region need not be uniform: any desired color model may be used (our experiments use a quadratic model). We separately reconstruct the static background, and extract moving foreground objects, to provide a layered representation in which the regions and decorative lines are vectorized; we also record their motions. Simple user-assistance is required to complete the background.

The contributions of this paper are: (i) the new trapped-ball segmentation method, which is fast, supports non-uniformly colored regions, and allows robust region segmentation even in the presence of imperfectly linked region edges, (ii) the separate handling of decorative lines as special objects during image decomposition, avoiding results containing multiple short, thin over-segmented regions, and (iii) extraction of a single patch-based background for all frames, which provides a basis for consistent, flicker-free animations.

II. RELATED WORK

Sýkora's [3]–[5] work on vectorization of cartoon animations is the most closely related previous work to ours. In his approach, cartoons must have a foreground layer consisting of dynamic regions each enclosed by clearly visible outlines. He relies heavily on correct and *complete* detection of the enclosing outlines, which are detected using an edge detector similar to a Laplacian of Gaussian filter. Foreground and background are extracted using outlines in each frame, and then a graph-based region matching process is used to find the region relations and transformations between frames. Due to this requirement for strong outlines, his approach fails on many cartoons in practice, such as the one in Fig. 2. Our method can handle more complicated cartoons, with non-uniform shading, and weaker edges. Significantly, we are able to compute a high-quality segmentation without perfect edge detection.

High-quality vectorization of cartoons requires accurate segmentation into relatively few meaningful regions. There is a vast literature on image segmentation. Many sophisticated

color-based methods, such as mean-shift segmentation [6], typically generate an over-segmented result with too many regions with flat shading and meaningless shape when applied to cartoons. Commercial software, such as Adobe Live Trace, CorelTrace, and AutoTrace, also typically produces regions with flat shading. Ardeco [1] can find regions of quadratically varying color. However, as this method initially labels the pixels randomly and refines the labeling, it also often produces many small regions, and hence is unsuitable for our purpose. We generate larger initial regions based on edge information, and then refine these regions using color compatibility to find precise region boundaries. Because we use edge information to find initial regions, the final regions are larger and more meaningful. However, we only label each pixel once, so our method is much faster than many other segmentation methods, taking just a few seconds per frame.

Qu et al. [7] proposed a colorization technique that propagates color over regions, and is suited to 'manga colorization'. A level-set based method is used to segment manga images. Since manga is drawn in black and white for printing, artists usually use patterns like hatching and screening to illustrate shading, unlike cartoon animations which usually contain colored regions and fewer patterns. However, both of our segmentation processes encounter the same problem of preventing region growing through incomplete boundaries. Their method depends on tuning parameters to determine the size of the gaps to close, whereas our approach gives good segmentation results without supervision: see Section IV.

Sun also presents a semi-automatic *image* vectorization method [2]. Complex images must be manually decomposed into several semantic parts. Each part is then modeled using an optimized gradient mesh Coons patch, allowing for smooth color variation. Use of a gradient mesh means that relatively few regions are needed to represent an object. As the author notes, his method has problems with images containing complicated topologies, very thin structures, or many small holes.

Clearly, simply applying segmentation and vectorization on a frame by frame basis will not produce good results, especially in the presence of raster compression artifacts and occlusion. Segmentation which is not coherent between frames will cause flickering in the vectorized output. Previous approaches to temporally coherent video segmentation [8]–[10] have tried to avoid such problems via optimization. Although such methods may be appropriate for real-world video, they do not produce precise enough results for cartoons, as even minute inappropriate regions and outlines are clearly visible, especially when present in the background, due to the smooth shading found in cartoons. We assume that the cartoon has an unchanging background over a sequence of frames, allowing us to achieve temporal coherence by extracting a unified background before detection of foreground objects and their motions.

Background subtraction has been extensively studied for complex scenes which e.g. have dynamic backgrounds or changing lighting [11]–[13]. Typically, two separate processes are used for background subtraction and foreground region extraction. While generally performing well, they often neglect pixels

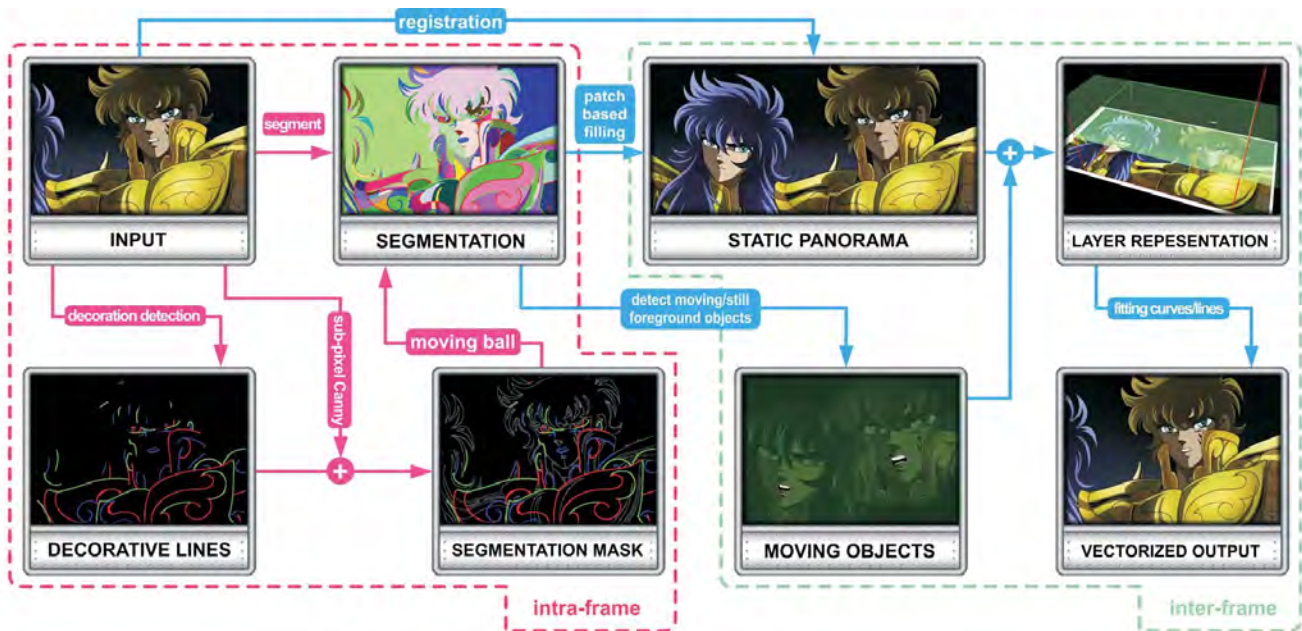


Fig. 3. Vectorization framework. The input at top left is the original cartoon video, and the output at bottom right the vectorized animation.

near boundaries, which may cause flickering of the background. As cartoons generally have fewer features with a clearer structure, we use a patch-based background filling method at the same time as performing foreground region extraction, to ensure all foreground pixels are appropriately allocated to regions.

III. OVERVIEW

Temporal video segmentation, which can segment a whole video into independent video sequences, each with a different (possibly moving) background shot, is a well-studied problem [14], [15]. We assume that such segmentation has already been performed. We focus on vectorization of a single sequence, comprising a static background, possibly with camera motion relative to it, and foreground moving objects. Such static backgrounds are widely used in cartoon making.

Fig. 3 shows the framework of our system. We assume that the input is a raster 2D animated cartoon, which is of low quality due to lossy compression at some stage. We vectorize each raster cartoon sequence as follows:

- In each frame, decorative lines are detected first, and these, together with edge detection results, are used to build a mask. The image is then segmented into regions using a trapped-ball method, controlled by this mask.
- To achieve inter-frame coherence, the frames in the sequence are registered by finding the homography, using the approach in [16]. A static panoramic background image is reconstructed by first initializing it with unchanging areas, and refined by adding regions identified as belonging to the background. The moving objects are extracted as a foreground layer, together with their between-frame motions.

- The background and foreground key objects are vectorized: their boundaries are represented as curves and their interiors filled using quadratic (or potentially any other) color models, and the vectorized animation is output.

We now consider particular aspects in detail.

IV. SINGLE FRAME DECOMPOSITION

An important requirement for improving visual coherence is to decompose the cartoon image into relatively few meaningful objects. Typically, cartoon images contain two types of objects: colored regions, and decorative lines—see Fig. 2. Colored regions need not have a uniform color, but may be based on some simple model of color as a function of position, e.g. a quadratic model.

Due to the typically large differences in shading between neighboring regions in cartoons, edge information provides a strong hint for meaningful region segmentation. We thus use a Canny edge detector [17] to extract edge information to guide image decomposition.

However, directly using edge information to find region boundaries and decorative lines has various challenges:

- Especially when processing compressed video, whatever parameter settings are used, any simple edge detector typically leads to various edges being missing, others containing gaps, and spurious noisy edges,
- Edges found only to pixel-position accuracy are insufficient for temporally coherent segmentation: camera motion does not generally involve a whole number of pixels per frame,
- Wide decorative lines produce two parallel edges, which can lead to many thin, and short, over-segmented regions.

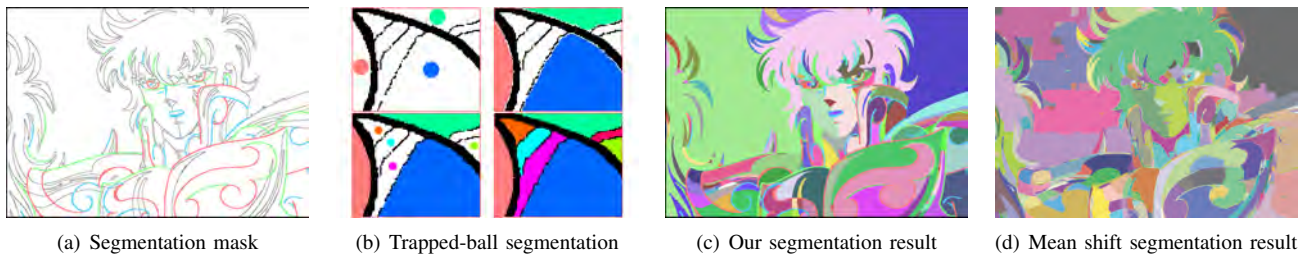


Fig. 4. Single cartoon frame segmentation

We take these into account as follows. First, the decorative lines are separately extracted in each frame.

The pixels covered by these lines are then combined with other pixels belonging to edges detected in the frame, using a standard Canny edge detector, to give a mask to control image segmentation. Segmentation of each frame is performed using a novel *trapped-ball model*, allowing us to overcome imperfections in the edge detection results. The idea is that a suitably large ball moves around inside a region delimited by the edge mask. Even if the edge mask locally has a gap, the ball is too big to go through it, and so will be limited to the initial region. In practice, we use morphological operations from image processing to implement this conceptual framework. We allow spatial variation of color within each region. By default a quadratic model is used, but other models could be substituted, depending on user requirements on computation time and image quality.

We now give further details of decorative line detection and trapped ball segmentation.

A. Decorative line detection

Unlike real video, which contains just *edges* between regions of different color, cartoons also often contain drawn *decorative lines*, with small but finite width. The former correspond to a high color gradient, while the latter are specifically drawn by artists to convey structure, motion, or other artistic meaning. Some decorative lines are located within regions, while others may emphasize region boundaries. Commonly-used edge detectors like the Canny edge detector are not suited to decorative line detection—such detectors would find two edges, one on either side of the decorative line, leading to the decorative line being considered as a narrow region. Furthermore, edge detectors typically lead to results with gaps, so to produce connected decorative lines as narrow regions, we would need either an edge-linking method, or a flood-filling method which could ignore gaps of a certain size. To avoid such issues, we detect decorative lines separately from edges, using a specialized approach.

While Sýkora’s method [3]–[5] also detects decorative lines forming outlines of regions, he relies on an artistic style which produces *closed* decorative lines. Unfortunately, decorative lines do not take this form in most recent cartoon art styles: often, at least some regions are only delimited by edges (i.e. color steps) in the image rather than explicit decorative lines.

Sýkora also assumes that decorative lines are the darkest regions in a cartoon, which again does not always hold. One way to link line points is to use alpha-shape techniques [18]. Though this work has theoretical importance, in practical use, the results are similar to those produced by traditional heuristic edge linking. We use a simple and fast algorithm based on traditional heuristic methods, which exploits the characteristics of cartoon images to improve the result. Experiments show that our result is much better than a traditional general purpose edge linking algorithm. Our processing comprises three steps.

First, we find points on the centreline of each decorative line (we call these *line points*, to distinguish them from edge points). To do so, we convolve the image with a *second* derivative of a Gaussian. Local maxima of the result correspond to line points, and are found by using non-maxima suppression as in the original Canny edge detector [17]. This also gives a local line orientation at each line point.

We next link the line points to give as-long-as-possible contiguous lines. Steger’s linking algorithm works by repeatedly adding points to the current line provided that their orientation is consistent with that of the line. This approach can break down due to noise in the orientation estimates. Suppose the orientations at successive points along the line are $\dots, \alpha_{k-2}, \alpha_{k-1}, \alpha_k$. Steger uses α_k as the orientation of the line to find the next point which will be added to the line. We overcome noise by replacing α_k by a smoothed version α'_k when deciding whether to add the next point to the line. We use a weighted average giving points closer to the current point a greater weight:

$$\alpha'_k = \frac{\sum_i \omega^{k-i} \alpha_i}{\sum_i \omega^{k-i}}.$$

The parameter ω should be in $(0, 1)$; we have used $\omega = 0.3$ in all our examples.

Finally, a traditional Canny edge detector is used to find both edges of each decorative line, from which we can estimate its width. For robustness, we check for and discard outliers, assuming the width changes smoothly, and also discard lines of width larger than 10 pixels.

Sample output is shown in Fig. 4(a), where each *colored* line is one decorative line. Detecting such decorative lines separately and representing them explicitly as lines with a known width greatly improves the quality of the final vectorized result: otherwise, many short, thin, regions are found instead, leading to an oversegmented result.



Fig. 5. Trapped-ball segmentation achieved by using morphological operations

B. Trapped-ball region segmentation

Next, we find the regions, which should be large and represented by a single consistent color model. A segmentation mask is formed comprising the decorative lines, and conventional edges where there is a sudden change of color (the latter are shown as black pixels in Fig. 4(a)). This mask gives guidance as to where region boundaries should be, but in practice the mask will not form contiguous, ‘watertight’ boundaries, precluding simple floodfilling.

One possibility is to try to make the mask watertight by performing edge-linking [19] on the mask. Unfortunately, such methods cannot *guarantee* to close all gaps, and even a single failure may result in two large regions being inappropriately fused. Another approach is to design a leak-proof floodfill algorithm of the kind often used in commercial software, e.g. Adobe Flash. If performing level set segmentation, a relaxation factor can be used to prevent leakage [7]. However, such methods can only prevent leaking from gaps up to a certain size which is either explicitly defined or implicitly determined by other parameters. There is no robust way to determine such parameters.

Thus, instead, we have designed a novel *trapped-ball* algorithm particularly suited to cartoon segmentation. The basic idea behind our segmentation method is to move a ball around, limited by the mask; each separate area in which the ball may move determines a region of the image. Fig. 4(b) illustrates *trapped-ball segmentation*. A *best first* rule is used: at the start of the algorithm, we use a large ball, and then iterate with balls of smaller and smaller radius. Choice of ball sizes is discussed later. In practice, the same effect as moving a ball can be achieved by using morphological operations [20] from image processing, as we now explain.

In detail, to find regions, the following three steps are iterated:

Trapped-ball Filling A ball-shaped morphological structuring element of radius R is used. Fig. 5 illustrates the morphological operations in use on part of the image. Given the original mask (a) (note the gaps in the thin lines), we first use flood-filling, giving (b), shown without the mask in (c). We then use a circular structuring element having the size of the trapped ball, and perform an erosion operation on (c), giving (d), and then a dilation operation on (d) to give (e), which is the result of the first pass of our trapped ball filling, shown with the mask for reference in (f). (Note that the blue region is presumed to extend outside the red box, explaining why erosion does not shrink the blue region at the edges of the red box).

Color Modeling After using the above process on the whole image, multiple large regions result. We assume that a single color model is an adequate fit to each such region, an assumption which works well in practice, in terms of producing visually acceptable results, even if counterexamples are not hard to construct—certainly, any sudden changes in color would have produced edges. By default we use a quadratic color model, in HSV color space, as used by Ardeco [1]. For each channel, the color of the i^{th} region at pixel (x, y) is modeled by the function $f_i(x, y) = a_{i0} + a_{i1}x + a_{i2}y + a_{i3}x^2 + a_{i4}xy + a_{i5}y^2$. The parameters a_{ij} are determined by solving a linear system equation for all pixels belonging to region i . As each region must be larger than the ball, we can be certain there is adequate data for fitting a color model.

Region Growing We next grow the regions generated by trapped-ball filling. This is necessary because the ball cannot penetrate into narrow pockets belonging to regions, or pass through narrow necks separating parts of the same region. Growing is subject to two constraints. Firstly, edge pixels may not be added to a region. Secondly, pixels added to a region should agree with its color mode, to within perceptual limits. We proceed as follows. Initially, each pixel is labeled to indicate its region, or as null if not yet assigned to any region. We define the reconstruction error of a labeled pixel to be the difference between its actual color and that predicted by the corresponding color model. To perform region growing, we put all region pixels at the boundary of each region into a priority queue, sorted by reconstruction error with respect to the region they belong to. Then, we repeatedly pop the pixel with minimum reconstruction error from the queue. We next consider each of its unlabeled neighbors, and consider the reconstruction error resulting if it is labeled as belonging to the same region. If the error is sufficiently small (in practice, less than 20), the pixel’s label is updated, and it is added to the priority queue. We repeat these steps until the queue is empty, or the least reconstruction error is too large.

As the above steps are iterated, all labeled pixels are added to the segmentation mask, so that subsequent iterations do not consider pixels which are already labeled.

After the first iteration, some pixels may remain unlabeled, so we reduce the ball radius by 1, and iterate, labeling and growing smaller regions, and so on, until the ball radius reaches 1. By this stage, all pixels must have been labeled, and the image fully segmented.

Theoretically, the initial radius used should be equal to the maximum distance of any pixel from some edge pixel. In

practice, for speed, we use a value of 8 pixels, as in our experience, any gaps in the mask are almost always smaller than this. Clearly, if highest quality results are desired, the user can increase the initial ball radius setting.

Finally, we may wish to eliminate regions which are too small. A user chosen ‘fine detail’ parameter in the range 0 to 200 pixels decides the minimum permissible size of a region. Any regions smaller than this are merged with the neighboring region with the most similar color model. This helps to provide a balance between file size and quality in the final result.

Unlike the method in [7], our trapped-ball filling step can deal with any size of gap, since the ball size is reduced iteratively. The region growing step utilizes color information allowing it to fill in narrow regions rather than leaving gaps.

Fig. 4(c) illustrates our segmentation result for the earlier example, and compares it with the *mean shift* segmentation result (see Fig. 4(d)). Here we used the EDISON system [21] based on edge detection results as a weight map for mean shift segmentation. The spatial and color bandwidths for mean shift were set to 7 and 8 respectively, with a gradient window size of 5×5 , mixture parameter of 0.5, and threshold of 0.5 for the weight map.

Because the mean shift approach neither uses color models, nor treats decorative lines separately, it produces oversegmented results for three particular kinds of regions: regions with shading, e.g. the character’s chest armor, regions with video compression artifacts, e.g. background regions, and regions corresponding to lines. Clearly, our result has fewer regions, and the regions found are more meaningful. This is mainly due to our use of edge masks as a hard constraint, which while unsuited to general images, can provide very good results for cartoons. Mean shift segmentation aims to segment images in general, so cannot use such a hard constraint, but instead uses gradient as a weight map. Furthermore, as the mean shift algorithm does not use color models, it will produce multiple segments for regions of varying shade.

As our segmentation algorithm is based on the assumptions that edge information in cartoons is very important and provides semantic hints, and hence regions delimited by the edge mask can be fitted by color models, it may fail when edges are too weak or regions are too complicated, as further discussed in Section VII.

The morphological operations needed to carry out the trapped-ball filling process take $O(n)$ time using the algorithm in [22]. Error sorting needed during region growing can be done by using an index heap or Fibonacci heap, so the whole segmentation process has linear time complexity.

V. INTER-FRAME COHERENCE

To achieve a high-quality vectorized result, we must avoid even small amounts of flickering due to lack of temporal coherence. This is especially important in the background, as it is static, making flicker more apparent. Most methods for

achieving temporal coherence in video are based on global optimization, and do not consider the particular requirements for background coherence.

Registering the images in a sequence allows us to locate each frame I_i in a global coordinate system by transforming it to \tilde{I}_i via a homography H_i . To do so, we make various assumptions: (i) the foreground area is not too large compared to the whole image, and (ii) the background motion comprises translation and scaling. While these may sound restrictive, they generally hold for many cartoon animations. Under these assumptions, registration methods such as SIFT [16] matching or even brute force search work well. Using RANSAC, we can detect outliers and overcome the jitter between frames. We obtain visual coherence by separately reconstructing the background as a single static panoramic image and extracting the foreground objects in the global coordinate system.

A. Background filling and foreground extraction

Usually, a cartoon comprises several scenes. Each scene is a sequence of frames with a constant background, on top of which moving foreground objects are placed. A camera motion is then often added to produce the final cartoon scene. We remove the camera motion by registration, resulting in a background which should be entirely static. We now discuss how we find the initial background for a scene and then refine it using per-frame decomposition results.

Let B be the global background image (larger than individual frames, because of camera motion), and let M be a map of the same size which for each pixel estimates the probability that it belongs to the background. Initially M_q is set to 0 for every pixel q , meaning that B_q is unknown.

Unlike [11] which is based on Gaussian mixture models, we construct an *initial background* based on median pixel values in appropriate frames, as the background of a cartoon should be unchanging. At the same time, we estimate the background probability of each pixel. Taking each image \tilde{I} in turn, we find the set S_q of pixels corresponding to each pixel B_q , by extracting the corresponding pixel $P_{i,q}$ in each registered image \tilde{I}_i , if such a pixel exists. B_q is set to the median color of S_q , and M_q is the fraction of S_q where the pixel color differs from the median value by less than T_c , the limit of perceptual ability to distinguish color differences, here taken to be 10 units. If M_q is sufficient large, i.e. larger than 90%, we regard B_q as a ‘stable’ background pixel, in that its color is almost unchanging over time. Fig. 6(a) shows the ‘stable’ pixels belonging to the background for a particular video.

As a cartoon has well-defined features, we assume that each region in each frame either belongs to the background or the foreground. However, different parts of the background may be occluded by foreground regions in each frame. By estimating the probability that each region belongs to the background, we then use a *patch based filling method* to dynamically change the probability map and refine the background image as well as to decide which are foreground objects. To be able to recover

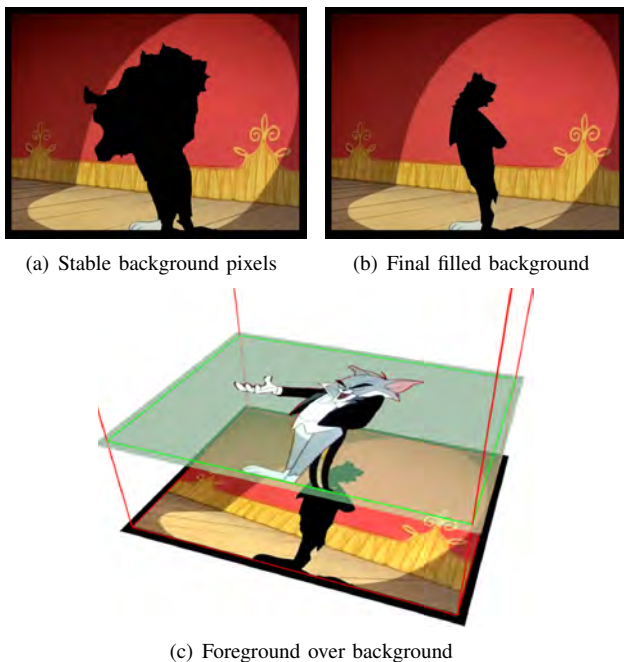


Fig. 6. Illustration of background filling and layer representation

those parts of the background that are occluded by foreground regions at various times, we examine all regions, and compute the probability that they belong to the background: P_{ij} is this probability for region R_{ij} , the j^{th} region in frame i , and is found by averaging the probabilities for its pixels:

$$P_{ij} = \sum_{|P_{iq}-B_q| < T_c, \forall P_{iq} \in R_{ij}} M_q / |R_{ij}|.$$

A larger value of P_{ij} means the region has a larger probability of belonging to the background. All regions in all frames are next sorted in descending order of probability. Pixels in the region with highest probability are added to the background, and M is updated accordingly. The background probability of each region remaining in the list is then updated and the list resorted. We then consider the remaining region with highest background probability, and so on. This process is quick as only a few regions need to be updated and resorted each time. This process terminates when either the background image has been completed, or when the highest probability of any remaining region being background is below some threshold value, at which point remaining objects in the list are considered to be *foreground objects*. We usually use 0.3–0.4 as this threshold, depending on the quality of the input video. Too low a value may incorrectly fuse background and foreground areas, while too high a value results in too small a detected background, with certain background areas being treated as foreground regions at higher cost. Generally, the lower the threshold that can be used, the better, as having a larger background area will provide better temporal coherence in the results.

At the end of this process, the background image may still contain holes where foreground objects *always* occlude the background. Clearly, such holes are irrelevant. Fig. 6(b) shows the results after filling the background in this way.

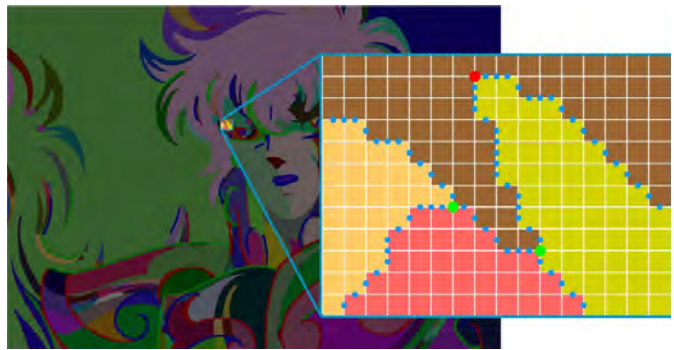


Fig. 7. Vectorization of segmentation results

While automatic background image filling usually works well, certain background areas may be uncovered in only a few frames, and thus cannot be detected automatically. Simple user interaction is thus used to add such regions to the background. This interaction needs only a few mouse clicks, combined with scanning through the video, and does not take long. At the same time, the user may remove regions which have been inappropriately assigned to the background image (e.g. the stationary feet of a cartoon character, the rest of whose body is moving).

B. Foreground object tracking

For such high-level tasks such as editing, and for the final video to be highly compressible, it is important to identify corresponding foreground regions whenever possible in successive frames and to find the key objects, which we then represent in vectorized form. However, as cartoons are hand drawn, the shapes of objects can differ slightly between neighboring frames, even if unchanged semantically. This issue hinders key object detection. We use tracking [16] and motion based segmentation methods [10] to be able to follow key objects between frames.

The motion of large foreground objects between adjacent frames can easily be detected by tracking or matching, as the motions are typically small. Small regions are more tricky to track, but tend to move along with neighboring large regions.

We follow the approach in [10], which groups pixels according to their motion, with the difference that we use large foreground objects as seeds for region tracking. We compute their homographies and place them, together with neighboring regions with similar transformations, into groups, each in a different initial layer. This may result in allocation of certain regions to more than one group, so we use a graph-cut algorithm [10] to decide the final grouping of regions, as well as occlusions.

Regions which are successfully tracked are replaced in subsequent frames by the original region modified by the corresponding transformation: the segmentation results found in subsequent frames are discarded. The groups found are represented as one key object and a sequence of transformations, and they typically have semantic significance (for example, an



Fig. 8. Vectorization results. Columns 1, 3: original video frames; columns 2, 4: corresponding vectorization results. (a), (b) Two clips from Tom & Jerry; (c) A clip from Saint Seiya.

arm of one color with a hand of another), which is useful for further editing and other high-level processes.

C. Vectorization

We now have a set of regions assigned to foreground layers or the background. To achieve a seamless and accurate vectorization result, the *mutual* boundaries of adjacent regions in the *same* layer (and hence which are moving together) must be determined, as well as other boundaries. Boundary curves are terminated at endpoints, which are points where more than two regions meet (green points in Fig. 7) or boundary points with a local maximum of curvature (red points in Fig. 7). We fit one or more cubic Bézier curves to each boundary point sequence between adjacent endpoints, to achieve a given tolerance. We use tolerances between 4 and 20 pixels, allowing a trade-off between quality and file size.

Solid regions are represented by their boundary curves and color model parameters. Decorative lines are represented by a polyline with accompanying width and color at each vertex. Regions in the background layer only need to be vectorized once. Each foreground object is also vectorized once, and its transformation in each frame is also stored. It is straightforward to convert our vectorized output into Adobe Flash `swf` format for use in the Flash vector animation editor.

VI. EXPERIMENTAL RESULTS

Figs. 1 and 8 show various cartoon vectorization results. In older cartoons (Figs. 8(a,b)), a constant color model is sufficient for each region; in modern 2D cartoons (Figs. 1 and 8(c)), a quadratic color model is more typically required.

We have implemented a prototype system on an Intel Core 2 Duo 2.4GHz machine, with 1Gb of memory. From a user-input source cartoon video, it can provide various intermediate outputs including per frame segmentation, a static background image and the final vectorized file which gives the color model and boundary representation of each region. We also provide an optional interactive step during background reconstruction: the user can browse the per frame segmentation results and add desired regions to the background or remove regions from it.

Our segmentation and inter-frame region matching algorithms are particularly efficient, as they take advantage of the relatively simple nature of cartoons, allowing stable detection of large regions. On 640×480 input, segmentation typically takes 2–3s per frame, background and foreground extraction take 4–8s, and vectorization takes under 1s; times vary with complexity of scene, but around 10s per frame is typical, which means that an entire cartoon can be vectorized in an acceptable length of time. In contrast, Ardeco [1] takes 72–125s per frame for 512×512 images, the Stanford VectorMagic image result in Fig. 9 took over 100s, and the method in [2] takes around 250–1000s per object.

Fig. 9 compares our vectorization results for a single frame to those produced by Stanford VectorMagic and Adobe Illustrator Live Trace. Our result gives more a visually appealing segmentation with fewer regions, with decorative lines also identified.

Figs. 10(a), 10(b) show how our results can be readily edited using Adobe Flash. The left hand example shows how Tom the cat has been extracted from one clip (see Fig. 8(a)), and inserted between the background and foreground (Jerry the



Fig. 9. Top: vectorization results using VectorMagic, our method and Adobe Illustrator Live Trace. Bottom: region boundaries.



Fig. 10. Editing: (a) taking a character from one scene and putting him in another, (b) changing the shape of the character's ears

Video clip	Frames	DivX	Vec1	Vec2
Fig. 1: Saint	91	1049kB	75kB	41kB
Fig. 8 (a): Tom	80	1022kB	542kB	283kB
Fig. 8 (b): Jerry	54	610kB	388kB	201kB
Fig. 8 (c): Saint	120	842kB	493kB	267kB
Fig. 10(a): Jerry	82	619kB	192kB	98kB

TABLE I
FILE SIZE COMPARISON (SEE TEXT).

mouse, in bed) of another clip, using simple drag-and-drop. The right hand example shows a simple edit to Jerry's ear which was done on just two frames, and continues as the region is propagated through the video (the original image sequence is illustrated in Fig. 8(b)). (Editing was needed on two frames as Jerry's ear underwent a large change in shape part way through the original input sequence—note that the original was hand-drawn).

Table I compares the file sizes of our vectorized output to the original DivX compressed raster videos. The table reports DivX 6 compressed video sizes (DivX), high quality vectorized output with decomposition detail level 0 and curve fitting tolerance 4 (Vec1), and acceptable quality with decomposition detail level 200 and curve fitting tolerance 20 (Vec2). For high quality output, our vectorization method typically leads to a file size of roughly half that required by DivX 6 compressed

video. This can be further reduced by another factor of 2 for lower but still acceptable quality output. We note that if our method is used to accelerate network transmission of cartoons, a desired bit rate can be adaptively achieved by adjusting the curve fitting and fine detail tolerances mentioned earlier. See also Fig. 13.

VII. DISCUSSION

A. Comparison

We now compare our results with those from the methods in [1]–[5], using images taken from their work (see Fig. 11). Sýkora's method [3]–[5] imposes very strong constraints on the cartoons processed, i.e. regions must have thick, closed black outlines and constant color. As a result, their method failed on most cartoons we tried, making a direct comparison difficult. Our trapped ball model for segmentation means that we can handle region boundaries which need be neither closed nor clear, greatly increasing the applicability of our technique. Compared to Ardeco [1], we can produce almost the same quality of results even for non-cartoon images, despite these not being our main goal. In Fig. 11(b), the mean pixel difference between output of this method and the original image (L_2 norm in RGB color space) is 9 units, whereas in

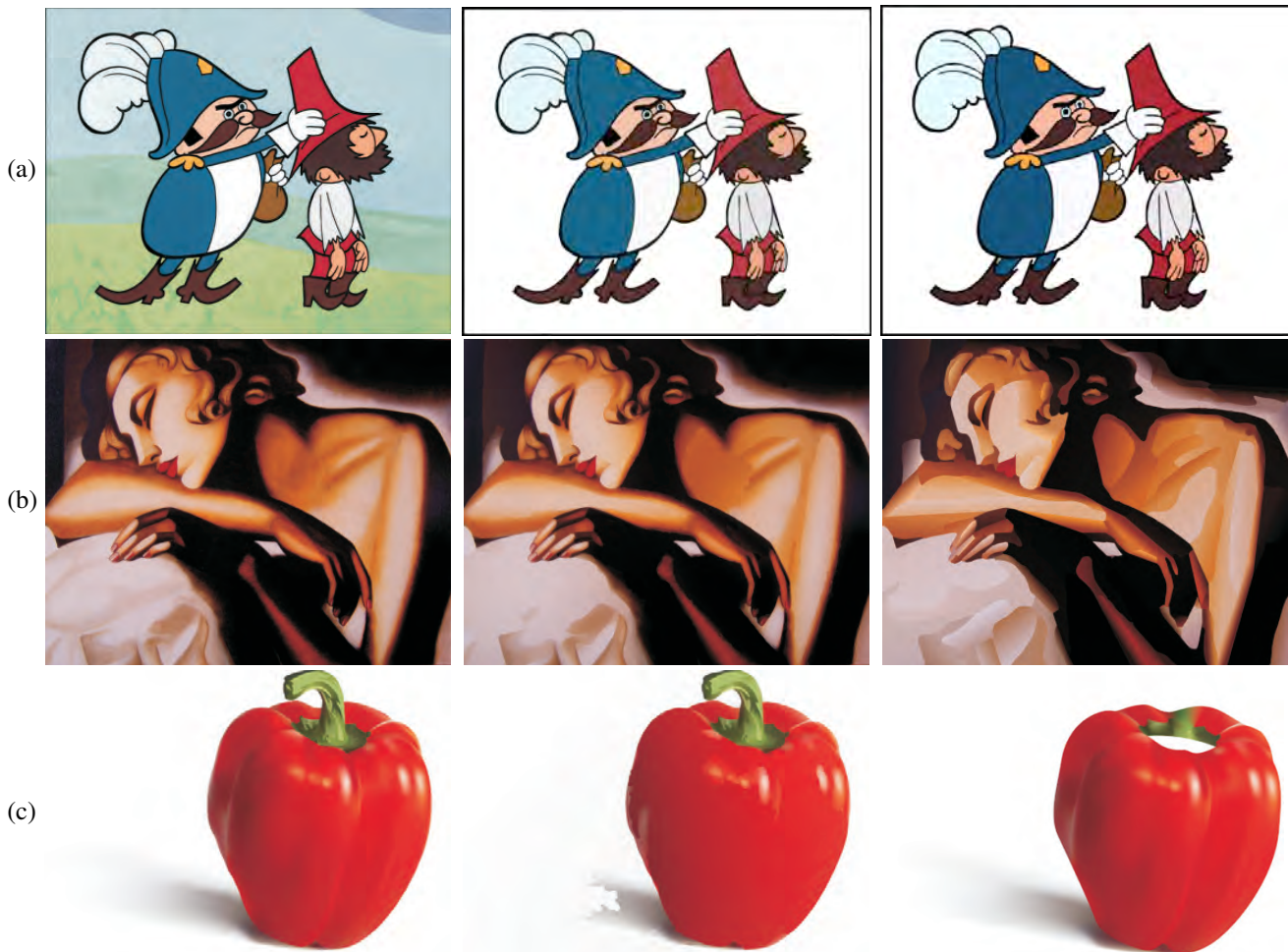


Fig. 11. Comparison with other vectorization methods. Columns: original video frames, our results and others' results. Rows: results produced by Sýkora's method, Ardeco, and Sun's method respectively.

our result, the mean error is 4 units. The resulting vectorized output from Ardeco has a more *posterized* appearance than our result: in particular the face and shoulder of the woman have more noticeable steps in shading. Price [23] and Sun [2] give methods for vectorizing images that contain objects with complex but smoothly changing textures; they use meshes to describe those textures. Their work thus has very different motivation from ours: cartoons often contain untextured regions with generally well-defined boundaries. (Considering texture as a particular kind of color model would be an interesting extension to our current work). Both of these methods need users to manually create initial meshes for each object in the image, and would be very hard to extend to video. However, these methods provide very low reconstruction errors, often of less than 1 unit per pixel, which our method cannot. We note that, often, input video clips suffer from compression artifacts. Our output may actually have higher visual quality than the input: least-squares reconstruction errors are not a satisfactory metric for assessing vectorization quality.

B. Limitations

Our method targets cartoons, which have a particular type of artistic content, including smoothly (but not necessarily

uniformly) colored regions and decorative lines. Our method produces suboptimal results when:

- Adjacent regions have similar color, or edges are weak for some reason: boundaries may be incorrectly placed or regions incorrectly fused. For example, in Fig. 11(c) the shadow of the pepper in our result is poorly segmented, while in Fig. 12(b) poor segmentation results where light crosses the character's hair.
- Foreground objects adjacent to the background have a similar color to it. This can result in foreground objects being assigned to the background. An example is Jerry's foot in Fig. 8(b). (Colors are compared after color modeling. After color modeling, the region color of the foot is considered to be similar to that of the background.)
- There are complex textures in the scenes such as grass or forest. In this case, an inappropriate color model is used for the kinds of regions present, and large colored regions does not represent the texture well, resulting in large reconstruction errors.

Certain art-styles such as watercolor painting, and highly detailed cartoon clips with shining light effects or complex textures are thus not well suited to our method. We provide



Fig. 12. Cases producing poor results. First column: original cartoon images, second column: our vectorized output, third column: our decomposition result.

examples of such cases in Fig. 12. Fortunately, scenes with effects like those in Fig. 12(b) are fairly rare. One possible way to handle them is to first model the lighting and compensate for it, as artists usually use a model to generate such lighting effects. Replacing color models by texture models is an interesting future extension to our work to cope with the scenes with textures (see Fig. 12(a)). Overall, while such difficult cases lead to suboptimal results in terms of fidelity to the original, or quality of segmentation, we are still able to obtain meaningful output—our algorithm degrades gracefully.

In some cartoons, the background is much more complicated than the foreground, as it only needs drawing by the artist once, and more effort can thus be expended on it. Vectorizing such a background image would result in many small regions, and a large file size. One possible response is to produce a somewhat less detailed background by adjusting the region segmentation process to use a larger value for the fine detail parameter. (This may be acceptable, as the user mainly concentrates his gaze on the foreground objects). Another solution is to simply store the background as a bitmap—as this is a common background for all frames, this would add relatively little extra storage requirement to the final result.

VIII. CONCLUSIONS

We have presented a system for transforming 2D rasterized animated cartoons into vectorized representation. The output animations are visually flicker free, smaller in file size than the input, and contain large regions, potentially suited to applications such as editing and multimedia information retrieval. Decorative lines are output separately to colored regions.

Our system takes advantage of the particular nature of cartoons to rapidly achieve high quality image decomposition with more meaningful segmentation results than existing methods. The

segmentation approach supports arbitrary color models for each object.

A number of extensions would further enhance our system. The most desirable is to provide higher-level understanding of the foreground regions, by using a more sophisticated method to merge adjacent regions with semantic significance. This requires sophisticated handling of changes in shape of objects between frames to eliminate errors in hand-drawings. Furthermore, a whole film may have many scenes, shared amongst which there may be common characters in identical poses; these too should be identified, both to provide further compression, and to enhance the usefulness of the results for high level processing. Currently, we cannot handle cartoons with ill-defined region boundaries, caused e.g. by shining light, smoke, flames and motion blurs, nor can we handle cartoons with textured regions. We hope to extend our segmentation approach to cope with these.

ACKNOWLEDGEMENTS

We would like to thank Warner Bros. Entertainment Inc. and Toei Animation for granting us licenses to reproduce the following cartoons, respectively: Tom and Jerry, Saint Seiya. We would like to thank the anonymous reviewers for their valuable comments.

This work was supported by the National Basic Research Project of China (Project Number 2006CB303106), the Natural Science Foundation of China (Project Number U0735001) and Specialized Research Fund for the Doctoral Program of Higher Education (Project Number 20060003057), and an EPSRC UK travel grant.



Fig. 13. Vectorized representations with different fidelity of detail. Data sizes: left, 24kB, right, 9kB.

REFERENCES

- [1] G. Lecot and B. Levy, "Ardeco: Automatic Region DEtection and CONversion," in *Proceedings of the 17th Eurographics Symposium on Rendering*, pp. 349–360, 2006.
- [2] J. Sun, L. Liang, F. Wen, and H.-Y. Shum, "Image vectorization using optimized gradient meshes," *ACM Trans. Graph.*, vol. 26, no. 3, p. 11, 2007.
- [3] D. Sýkora, J. Buriánek, and J. Žára, "Colorization of black-and-white cartoons," *Image and Vision Computing*, vol. 23, no. 9, pp. 767–852, 2005.
- [4] D. Sýkora, J. Buriánek, and J. Žára, "Sketching cartoons by example," in *Proceedings of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pp. 27–34, 2005.
- [5] D. Sýkora, J. Buriánek, and J. Žára, "Video codec for classical cartoon animations with hardware accelerated playback," in *First International Symposium of Advances in Visual Computing*, vol. 3804, pp. 43–50, Springer, 2005.
- [6] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.
- [7] Y. Qu, T.-T. Wong, and P.-A. Heng, "Manga colorization," *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, vol. 25, no. 3, pp. 1214–1220, 2006.
- [8] C. L. Zitnick, N. Jovic, and S. B. Kang, "Consistent segmentation for optical flow estimation," in *International Conference on Computer Vision*, pp. II: 1308–1315, 2005.
- [9] M. P. Kumar, P. H. S. Torr, and A. Zisserman, "Learning layered motion segmentation of video," in *International Conference on Computer Vision*, pp. I: 33–40, 2005.
- [10] J. Xiao and M. Shah, "Motion layer extraction in the presence of occlusion using graph cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 1644–1659, October 2005.
- [11] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *CVPR*, pp. 2246–2252, 1999.
- [12] H. W. Kang and S. Y. Shin, "Tour into the video: image-based navigation scheme for video sequences of dynamic scenes," in *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, (New York, NY, USA), pp. 73–80, ACM, 2002.
- [13] A. Monnet, A. Mittal, N. Paragios, and V. Ramesh, "Background modeling and subtraction of dynamic scenes," in *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, (Washington, DC, USA), p. 1305, IEEE Computer Society, 2003.
- [14] I. Koprinska and S. Carrato, "Temporal video segmentation: A survey," *Signal Processing Image Communication*, vol. 16, pp. 477–500, Jan. 2001.
- [15] Y. Gong and X. Liu, "Video shot segmentation and classification," in *ICPR '00: Proceedings of the International Conference on Pattern Recognition*, vol. 1, pp. 860–863, IEEE Computer Society, 2000.
- [16] D. G. Lowe, "Object recognition from local scale-invariant features," in *International Conference on Computer Vision*, pp. 1150–1157, 1999.
- [17] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.
- [18] F. Bernardini and C. L. Bajaj, "Sampling and reconstructing manifolds using alpha-shapes," in *Proc. 9th Canadian Conf. Computational Geometry*, pp. 193–198, 1997.
- [19] C. Steger, "An unbiased detector of curvilinear structures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 2, pp. 113–125, 1998.
- [20] M. Sonka, V. Hlavac, and R. Boyle, "Image processing, analysis, and machine vision," in *Chapman and Hall*, 1993.
- [21] C. M. Georgescu, "Synergism in low level vision," in *International Conference on Pattern Recognition*, pp. 150–155, 2002.
- [22] A. Meijster, J. B. T. M. Roerdink, and W. H. Hesselink, "A general algorithm for computing distance transforms in linear time," *Mathematical Morphology and its Applications to Image and Signal Processing*, pp. 331–340, 2000.
- [23] B. Price and W. Barrett, "Object-based vectorization for interactive image editing," *The Visual Computer*, vol. 22, no. 9, pp. 661–670, 2006.