Online Video Stream Abstraction and Stylization

Song-Hai Zhang, Xian-Ying Li, Shi-Min Hu, *Member, IEEE*, and Ralph R. Martin, *Member, IEEE*

***Abstract*—This paper gives an automatic method for online video stream abstraction, producing a temporally coherent output video stream, in the style with large regions of constant color and highlighted bold edges. Our system includes two novel components. Firstly, to provide coherent and simplified output, we segment frames, and use optical flow to propagate segmentation information from frame to frame; an error control strategy is used to help ensure that the propagated information is reliable. Secondly, to achieve coherent and attractive coloring of the output, we use a color scheme replacement algorithm specifically designed for an online video stream. We demonstrate real-time performance for CIF videos, allowing our approach to be used for live communication and other related applications.**

***Index Terms*—Abstraction, color scheme replacement, optical flow, segmentation, temporal coherence, video stream.**

I. Introduction

THE abstraction of images and videos becomes popular in non-photorealistic rendering, which simplifies scene information while retaining meaningful features in a stylistic fashion, so that the output is nice looking, and can serve the purpose of communicating the messages and reducing the perceptual effort to understand the contents.

Segmentation is a natural choice of tool for abstraction. DeCarlo *et al.* [1], [2] propose an *image* abstraction and stylization approach that transforms images into cartoon style using large regions of constant color, by a hierarchical segmentation. We also pursue this artistic style, and also use explicit image structure, but produce temporally coherent cartoon-like *animations* . The particular goal of our video stream abstraction approach is to produce an output style which is cartoon-like, having simplified regions with user-guided colors, and high contrast, which has many applications to areas such as live broadcast and communications, video games, entertainment, and virtual reality [3].

However, temporally coherent video stream abstraction is a challenging task. Firstly, existing segmentation-based methods require intensive user interaction to produce high-quality artistic results, and have a high computational cost. Although there are some methods [4]–[7] using feature-preserving smoothing filters to produce stylized videos automatically, their results are smooth shaded, and more like the painterly effects rather than cartoon-like effects. Secondly, processing video streams requires efficient algorithms in order to cope with real-time data. Offline algorithms can make use of future as well as past information, but this is not available in video streams. Furthermore, video streams have to be processed frame by frame, or at most make use of just a few past frames or an averaged history.

In this work, we present an online, automatic method to generate cartoon-like abstract animation from an input video stream (see Fig. 1). Little user interaction is required, other than selection of a reference video which determines preferred output colors (or alternatively a hue histogram), and setting of preferences which control the amount of detail retained in colored regions and line drawing before processing. Our approach benefits from two novel aspects:

- A segmentation strategy which uses optical flow to propagate segmentation information from frame to frame, with an error control strategy to help ensure that the propagated information is reliable. The segmentation constructs an explicit structure, which provides correspondences of matched regions, and achieves the temporally coherent abstraction results of videos.
- A video stream color scheme replacement method that does not require complete knowledge of the source color distribution (i.e., does not need to know future frames), and which applies the color scheme from a reference video (or image, or a user designed histogram of hue) to the input video, while keeping color consistency between frames thanks to the explicit structure of the video.

II. Related Work

A number of significant papers have addressed stylization, but most of them concentrate on images rather than videos [1], [8]–[11]. Much research has also considered the generation of cartoon-like video from real-world captured video with the aim of reducing the amount of work required for cartoon production, and the need for skilled artists. Examples include papers on "SnakeToonz" [12], keyframe based rotoscoping [13], and "VideoTooning" [14]. Such methods produce high-quality results, but still require intensive user interaction, and generally restart video processing every few tens of frames to avoiding error accumulation. Such methods typically require considerable artistic skill for good results.

DeCarlo and Santella [1], [2] propose an *image* abstraction and stylization approach relying on eye-tracking data to guide image simplification or stylization, using hierarchical segmentation, which well preserves the image structure. We pursue an

Manuscript received September 23, 2010; accepted July 29, 2011. Date of publication August 18, 2011; date of current version November 18, 2011. This work was supported in part by the National Basic Research Project of China (Grant No. 2011 CB302205), in part by the National Natural Science Foundation of China (Grant Nos. 60970100, 61033012), and in part by the EPSRC Travel Grant. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Nadia Magnenat-Thalmann.

S.-H. Zhang, X.-Y. Li, and S.-M. Hu are with the National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: shz@tsinghua.edu.cn; shimin@tsinghua.edu.cn).

R. R. Martin is with the School of Computer Science Informatics, Cardiff University, Cardiff, Wales, CF24 3 AA, U.K. (e-mail: Ralph.Martin@cs.cardiff.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2011.2165052

Fig. 1. Abstraction example. 1: original video frame, and 2–4: three frames of the stylized video with color scheme replacement.

artistic style similar to [1], and also use explicit image structure, but produce temporally coherent cartoon-like *animations*.

As a natural scheme to extend DeCarlo *et al.*'s work, several video abstraction approaches have been proposed based on the coherent video segmentation. We should notice that video segmentation is a fundamental and challenging problem as surveyed in [15], which has classified video segmentation techniques into three categories: the 3 D stack scheme, the $2D + t$ scheme, and the trajectory clustering scheme. For online video streams, which ask the algorithm to be fully automatic and do not need to know future frames, the 3 D stack scheme [14], motion layer segmentation scheme [16], and trajectory clustering scheme methods are not preferred because they all need costly whole-video analysis. For $2D+t$ scheme, Paris [17] uses isotropic diffusion and Gaussian convolution for real-time video stream segmentation; Collomosse *et al.* [18] preform a key-frame-based approach. However, these methods adopt a simple frame alignment, which limits the coherence for fast-moving scenes and objects. Actually, as Paris mentioned in the paper, using optical flow may provide better results on moving boundaries.

Other than segmentation-based approaches, various filtering techniques have been applied for both image and video abstraction [4]–[7]. Those works use image filters instead of segmentation, which are fast in practice and naturally fit to videos and video streams. Their coherence are guaranteed by the implicit image structure itself, and the stability of the filters. But different from the cartoon-like effects that we pursue, they tend to produce painterly effects.

In this paper, we use optical flow to guide coherent video segmentation for cartoon-like effects. Our results achieve comparable coherence as those filtering-based methods and, what is more, as we have constructed an explicit representation with region correspondences, performing coherent and stable color replacement becomes possible.

There are several papers [19]–[24] using optical flow to direct brush stroke movement or point placement during frames, but they do not take into account errors in optical flow, so limited coherence is achieved in image sequences. Bousseau *et al.* [25] also use optical flow in a watercolor generation pipeline, but rely on temporally bi-directional optical flow interpolation to reduce optical flow errors, an option which is not available in a live video system as future information is unavailable.

Several works consider the problem of color scheme extraction [26] and color transfer [27]–[30]. These mainly focus on image processing, and analyze source and reference features to determine the color transformation. However, we cannot extend such a method directly to live video stream color transfer, again because of the lack of future information. On the other hand, performing image color transfer for each frame independently cannot guarantee color consistency in the output as source colors vary in each frame. Wang *et al.* [31] give a color transfer method for still images forming a sequence; parameters are adjusted to present gradually changing effects. Image analogies have also been used for color transfer [28]. These take into account local similarities in the image, and process it pixel by pixel. The latter is most suited to producing complex textures, but our output style uses large regions of slowly varying color. Thus, we have instead devised a new efficient color replacement method which captures the color style from a given reference video, and applies it with temporal coherence to produce a stylized video stream without the need for source video feature analysis.

III. Overview

Our goal is to replace incoming live video by abstract animation, in real time. We must thus avoid time-consuming non-local analysis of the video (e.g., background reconstruction). During abstraction, our three goals are to:

- simplify the content, without oversimplifying important areas;
- modify the coloring in a controllable manner;
- retain temporal coherence.

Taking these into account, our pipeline is as shown in Fig. 2, and includes six steps.

Of these six steps, importance map computation, optical flow computation, and edge detection provide inputs for coherent image segmentation. The importance map is used to ensure greater detail in key areas. As optical flow information is inherently unreliable, we must be careful to avoid propagating and accumulating optical flow errors during segmentation. We use a careful error control strategy when propagating segmentation labels from frame to frame, to ensure temporal coherence of region representations between frames.

For simple output, we could simply replace each region by its mean color. Better results are obtained by using a color scheme replacement process based on a desired "color style" learnt offline from a reference video (or image, or provided by the user as a histogram). We use a mean-shift-based color transformation to move input colors closer to a reference color distribution. We are careful to preserve color consistency between frames, while still not needing to know the potential colors in future frames.

Finally, we smooth boundary curves using a low pass filter to produce more artistic effects, and then use a difference-of-Gaussians (DoG) operator on the input frames to detect and overlay edges, as in [4].

Algorithm 1 gives pseudocode for the whole algorithm, and details of image segmentation are described in Sections IV–VII.

Algorithm 1 Pseudocode for online video abstraction

```
while (! end of video stream) do
    //—Perform coherent image segmentation—
    Compute importance map;
    Compute Canny edges;
    if !(first frame) then
        //—Update segmentation using optical flow—
        Compute optical flow;
        Propagate labels from previous frame;
        Apply morphological filtering to those propagated labels;
        Grow regions;
    end if
    for (r = initial radius; r ≥ 1; r/ = 2) do
        Do trapped-ball filling with ball radius r;
        Grow regions;
    end for
    //—End of image segmentation—
    Apply color scheme replacement;
    Smooth regions;
    Compute DoG edges and overlay them;
end while
```

IV. Coherent Image Segmentation

As in [1], we perform segmentation in order to simplify image content. Several approaches exist for temporally coherent video segmentation [32]–[34]. Most solve the problem via optimization or a mean-shift approach, and thus perform a non-local analysis of the video. To process a *live* video stream, we perform a $2D + t$ segmentation by propagating information from one frame to the next.

Our approach uses optical flow but in a way which avoids accumulation of errors. Many optical flow computation methods exist [35]; we use Zach's [36] as it provides good results in real time. The first frame is segmented using the trapped-ball algorithm [37]; this algorithm is explained later. In each successive frame, optical flow is used to propagate segmentation labels for each pixel. In the result, some pixels may have *no* segmentation label (e.g., because they have been newly exposed from an occluded area), or may have *multiple* labels (e.g., because of optical flow errors). Labeling such pixels is the key to coherent image segmentation. Those which are sufficiently close in distance and color to an existing region are given the label of that region, while the remainder are segmented into *new* regions again using the trapped-ball algorithm. Algorithm 1 gives pseudocode for image segmentation; we now consider these steps in detail.

A. Importance Map and Edge Detection

Edges provide strong hints as to where region boundaries should exist. In image and video abstraction, *important areas*, such as faces, should be depicted in more detail than other areas, which requires segmenting them into more, smaller regions. Eye tracking [4] can find important areas, but is of limited applicability. We compute the importance map by faces detection [38]. Alternative strategies could also be used to compute the importance map, e.g., taking into account saliency and motion information [39], but at a higher computational cost. The important map is used to control the number of edges found during edge detection: the hysteresis thresholds of a Canny edge detector [40] are set inversely proportional to importance map values. In areas of greater importance, more edges are detected.

B. Region Label Propagation by Optical Flow

The first frame is segmented by the trapped-ball method based on the detected edges. For subsequent frames, the previous frame has been segmented, and optical flow has been computed; these are used to provide an initial segmentation for the current frame. The segmentation is represented by labeling pixels with positive integers corresponding to regions.

We start by labeling all pixels of the current frame as 0, i.e., unassigned to any region. The optical flow is rounded to determine a spatial mapping of each source pixel from the previous frame to a target pixel in the current frame. This target pixel is given the same label as the source in the previous frame, except as below, when it retains its 0 label:

- the optical flow error is larger than a threshold;
- *more than one* source pixel ends up at this target pixel, and the labels of these source pixels are different;
- this target pixel has *no* corresponding source pixel.

To determine if an optical flow error has occurred, we compute the color difference in CIELAB color space between the source and a linear interpolation of the values of the 4-neighboring pixels of the target position. If this exceeds a threshold T_f, the target is labeled 0. T_f is chosen according to the video quality; values in the range 5–20 are typically appropriate. This prevents obvious optical flow failures. Fig. 3 (top, right) shows the residual unlabeled pixels (red) after label propagation. Note that we do not explicitly need to detect scene changes in the incoming video: in such cases, the optical flow will have large errors, causing many pixels to be labeled 0, and will hence be re-segmented.

After pixel label propagation, region boundaries are typically rather interlaced. These are smoothed, and other mislabelings due to errors in the optical flow corrected, using a morphological filter: if 5 or more of the 8-connected neighbors of the target pixel are the same, but differ from the target pixel, and if its label

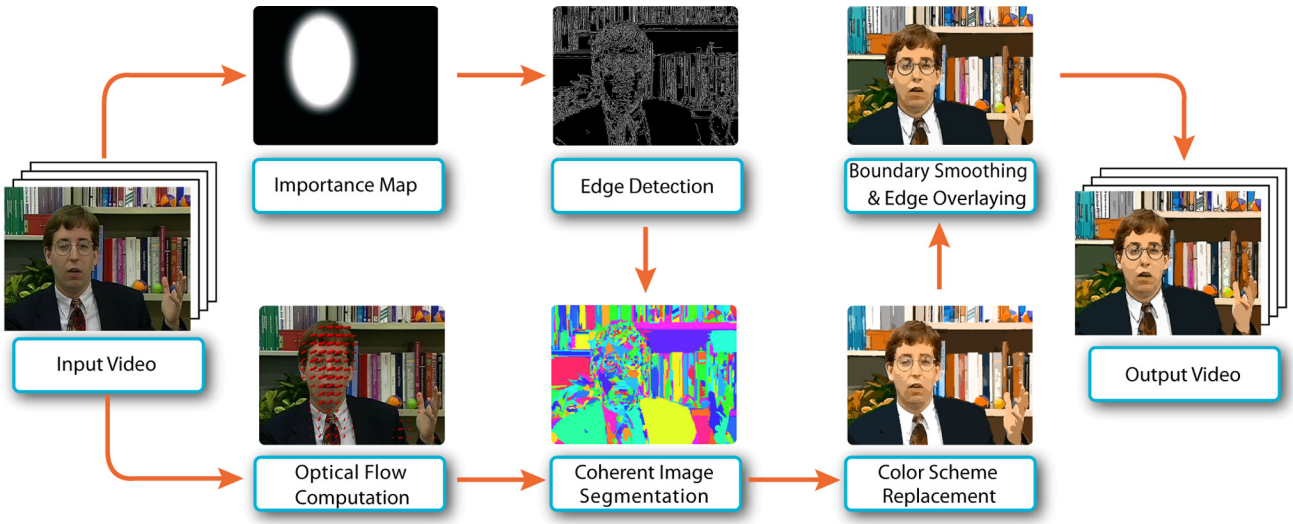

Fig. 2. Pipeline. Frame segmentation uses optical flow and edges as inputs, tuned by an importance map, to produce coherent regions. The color scheme replacement step transforms region colors according to a reference video.

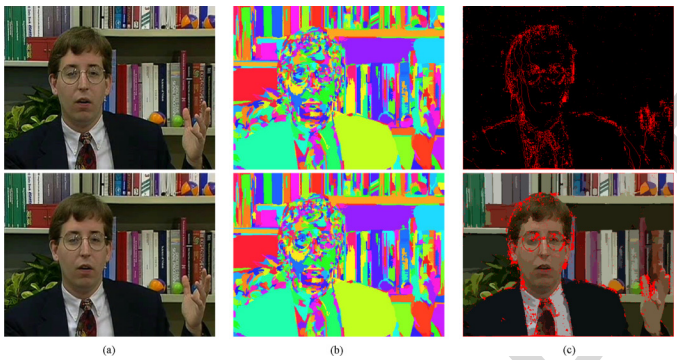

Fig. 3. Left: input frames 1, 2. Middle: segmentation results for frames 1, 2. Right-top: residual map before morphological filter. Right-bottom: region label propagation result after morphological filter.

is 0, then it is changed to the most frequent label in the neighborhood; otherwise, it is changed to 0. Fig. 3 (bottom, right) shows the propagated labels after morphological filtering, where red pixels are unlabeled.

We must now complete the segmentation by labeling the unlabeled pixels. This is a problem of minimizing label errors for those unlabeled pixels which should be allocated to existing regions, and giving new labels to those pixels which are insufficiently similar to existing regions. At the same time, the labels assigned must respect the edge information, so that the regions are in agreement with any superimposed edges drawn later. These requirements, taken together with efficiency considerations, make it difficult to solve the problem by traditional optimization or graph cut methods. Instead, we improve the region growing and trapped-ball filling [37] to finish the segmentation.

C. Region Growing

For each labeled region, a single, simple, color model is assumed to be an adequate fit. We use a constant color model, and assume pixels belonging to a region are near to its mean color.

We assign labels while respecting edges taking into account that 1) edge pixels may not be added to a region and 2) pixels added to a region should agree with its color to within perceptual limits. Let the *reconstruction error* of a labeled pixel be the difference between its actual color and that predicted by the corresponding color model. All unlabeled pixels adjacent to the boundary of any labeled region are put into a priority queue, sorted by reconstruction error with respect to the adjacent region. We then pop the pixel with minimum reconstruction error from the queue. If the reconstruction error is sufficiently small [in practice, below $20(1.2 - I(p))$ units in CIELAB space], the pixel's label is updated, and it is removed from the priority queue; at the same time, its unlabeled 4-connected neighbors are added to the queue. We repeatedly pop pixels until the queue is empty, or the least reconstruction error is too large.

D. Trapped-Ball Filling

Any pixels still remaining unlabeled belong to some *new* region. In general, multiple new regions may exist, so these unlabeled pixels still need to be segmented.

The basic idea of trapped-ball filling is to move a ball around, limited by a mask, formed here by the already labeled pixels and the detected edges. Each separate area in which the ball may move determines a new region of the image. The trapped-ball has the advantage over flood-filling that it cannot leak out between short gaps in the detected edges. Initially, a large ball is used to find each region's core, and then balls of successively smaller radius are used down to a radius of 1 to add more detailed parts of each region.

The initial ball radius is chosen according to the number of unlabeled pixels N_0. We set it to $\max(2, \sqrt{N_0}/30)$, which is typically in the range 2–32. In the first frame, *all* pixels are unlabeled, and an original segmentation for the whole frame is obtained using the trapped-ball method with a *large* initial ball size. In subsequent frames, unlabeled regions are typically small, and a *smaller* initial ball radius is used. If a scene change occurs, many pixels are now unlabeled, so a larger initial ball

size is again used. The extra processing entailed at a scene change can cause a slight lag, but as long as frames are processed faster than in real time, this lag can soon be overcome.

V. Color Scheme Replacement

Color selection is an important aspect of stylization, and different palettes can produce different emotional responses. Much work has been done on color transfer for images—applying the color scheme from a reference image to a source image. Clearly, in this case, the color distribution for the *entire* source and reference are known. However, when processing *live* video, we do not have complete information: future video content is unknown. Thus, our recoloring problem has two specific requirements: the transformation should cover the entire color space to allow for *any* possible future colors, and as we have video rather than images, we must also ensure inter-frame color stability for each region in the video.

Our color scheme replacement method uses the concept of *mean shift* [41]. Given a reference image or cartoon video clip, we count the number of pixels of each color to determine the color distribution. We must also choose the desired number of iterations and kernel radius for the mean shift procedure. Then, to determine the mapping of each input color to an output color, we determine a mean shift vector towards a local maximum in the color distribution, which shifts each input pixel's color towards the nearest peak in the reference distribution. Overall, given an input color distribution, the mapping transforms it towards the reference color distribution.

It would be costly to compute the reference distribution and mapping directly in CIELAB color space. Instead, we simply consider the hue value in HSV space, and compute the reference distribution and mean shift vector using H alone. In an analogous manner, [42] and [43] also use hue as the most important feature of color when performing color harmonization, i.e., mapping a source color distribution to a template distribution.

As these authors note, there is discontinuity in new color when two nearby colors are transformed to different local maxima, which may causing flickering between frames. Thus, we also use an interframe blending method which takes into account correspondence between regions in successive frames; this information is provided by our segmentation-based explicit structure, which is never achieved by filtering-based methods. Thus, after first computing an ideal *target color* by the mean shift procedure, we modify it to give the *actual output color* by interframe blending.

To find the color mapping, we first compute the color distribution of the reference video in HSV color space (which is more useful than RGB from a perceptual point of view), considering the three channels separately. In each frame, for the H channel, we compute a color histogram for all pixels having $S > 0.125$ and $V > 0.125$ (because the H channel is senseless for those colorless and dark pixels). Also we compute the mean μ_S and standard deviation σ_S for S channel pixels having $V > 0.125$ (also, S channel is senseless for dark pixels), and the mean pixel value μ_V of the V channel. (Alternatively, instead of providing a reference video, the user could provide a reference image, or even manually design a color histogram represented in this way.)

For each frame of the incoming video, all pixels in each segmented region are given the same target color. The *source color* (h, s, v) for a region is computed as the average color of all pixels in that region. This is mapped to a target color (h', s', v'). First, we compute μ_s, σ_s, and μ_v for this *frame* in an analogous manner to how they were computed for the reference video. We then compute (h', s', v') as follows. h' is set to the mean shift of h in the H channel histogram, by iterating the formula below k times; typically $k = 3$:

$$h_{i+1} = \frac{\sum_{c \in N(h_i)} cD(c)}{\sum_{c \in N(h_i)} D(c)}. \tag{1}$$

Here h_0 is h_t, and h_4 is the desired h'. $N(h_i)$ represents a 30° neighborhood in the histogram and $D(c)$ represents the histogram value for c. (This function is pre-computed for all possible h values and stored in a look-up table for speed.) As for the S and V channels, we simply perform the following formulas for s' and v' with the aim of adjusting the brightness and contrast to be closer to those of the reference video:

$$s' = \mu_S + (s - \mu_s)\frac{\sigma_S}{\sigma_s} \tag{2}$$

$$v' = v\left(\lambda + (1 - \lambda)\frac{\mu_V}{\mu_v}\right). \tag{3}$$

Here a linear function is adopted for s' to achieve target mean and standard deviation. For v', a direct proportional function is adopted instead because we should make sure that the darkness is still darkness, where λ is a constant to measure "how much we want the video retains its original brightness variation". In this paper, we choose $\lambda = 2/3$ for all results.

After obtaining the *target color* , we now compute the *actual output color* . Suppose some region R_i has a target color c_i, and the corresponding region R_{i-1} in the previous frame had actual output color o_{i-1}. We compute its actual output color o_i by color blending to improve color coherence:

$$o_i = (1 - \alpha(R_i, R_{i-1}))o_{i-1} + \alpha(R_i, R_{i-1})c_i. \tag{4}$$

Here $\alpha(R_i, R_{i-1})$ measures the correlation of the two corresponding regions: $\alpha(R_i, R_{i-1})$ is the ratio of: the number of pixels in R_{i-1} whose destination lies in R_i after optical flow, to the geometric average of the areas of R_{i-1} and R_i. If region R_i is a *new* region in this frame, we simply set $o_i = c_i$. This blending is executed in the CIELAB space.

An H channel color histogram extracted from a reference video and an example of color scheme replacement are shown in Fig. 4 (final results after boundary smoothing and edge overlaying are shown in Fig. 5).

VI. Results and Discussion

We have implemented our framework using a combination of CPU and GPU, on an Intel Core2 Quad Q9300 CPU at 2.5 GHz, with 4 GB memory, and a GeForce 8600 GT, using CUDA, OpenCV, and Visual C++ 2008's parallelizing compiler. Performance depends on image size and framework parameters. For a CIF (352×288) video stream, commonly used for live

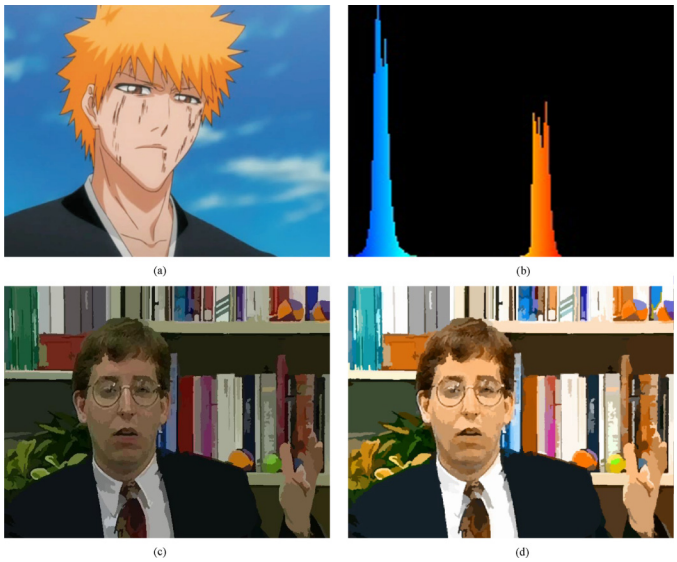

Fig. 4. Color scheme replacement. Top-left: a frame of the reference video. Top-right: histogram of its H channel. Bottom-left: segmented frame before replacement. Bottom-right: frame after replacement.

Fig. 5. Boundary smoothing and edge overlaying. Top-left: segmentation result. Top-right: after boundary smoothing. Bottom-left: detected lines. Bottom-right: smoothed result after edge overlay.

communication, face detection and edge detection is done on one CPU core, taking 40 ms per frame, while simultaneously, optical flow computation takes 20 ms on the GPU. Image segmentation takes from 20–30 ms, depending on the number of residual unlabeled pixels. Color scheme replacement and edge overlay take under 10 ms. Boundary smoothing takes 20–50 ms, depending on the number of boundary pixels. Typically we can process a CIF video stream with default parameter settings at about 9–12 frames per second using the CPU for everything but optical flow (we directly use Zach's GPU optical flow code [36]). We note that edge detection, morphological operations, and the DoG filter could also benefit from GPU implementation, permitting faster frame rates or higher video resolution. The current Nvidia GTX295 graphics card offers perhaps $10\times$ the performance of the older GeForce 8600 GT card.

Figs. 1 and 6 show various abstraction results. Our method achieves cartoon-like effects with simplified content comprising well-defined regions with bold boundaries, while using a chosen color style. Fig. 7 shows an example of the filtering-based method [4], which produces an painterly effect, and cannot achieve coherent cartoon-like effects by simple color quantization. In contrast, our results retain temporal coherence on both shape and color, benefited from the coherent image segmentation.

Different styles can be produced by the user, but objective evaluation of the level of success is difficult. Our results have been independently evaluated by a company who randomly chose 10 employees to subjectively evaluate 6 video clips of difference contents according to the temporal coherence, the cartoon shape style, and the cartoon color effect. The 10 participants include 5 males and 5 females, 3 professional designers and 7 novices, and their ages are between 20 and 40. Table I shows the average score of each video clip on three aspects. The result of the study shows that our method produces cartoon effects for streets and natural scenes better than the figures. The color scheme replacement greatly improves the stylization. The temporal coherence of our results is acceptable and comparable with [4].

Optical flow is the sole information used to guide interframe correspondence, so its accuracy is very important. Zach's method works well in most cases, as demonstrated by the number of residual pixels (Figs. 3 and 8). When the scene changes slowly, few residual pixels remain (Fig. 8, top), and most are labeled, according to color, during the region growing step. When the video content changes rapidly, the number of residual pixels increases. Indeed, essentially the whole image is composed of residual pixels at scene changes (Fig. 8, bottom), as useful optical flow information no longer exists. Thus, we do not need to explicitly detect scene changes: in cases with large numbers of residual pixels, trapped-ball filling provides resegmentation of the scene. While newly generated regions may cause certain inconsistencies, viewers find lack of coherence much more noticeable in slowly changing scenes than in fast moving scenes. Currently we use the rounded optical flow for label propagation, which may cause flickering in long and thin regions. In addition, image noise and motion blur may also decrease the quality of optical flow, and ultimately decrease the coherence of final results.

Fig. 9 illustrates the results of color scheme replacement using different reference images (these have been chosen to exaggerate the effects, and are atypical). The resulting hue distributions have been transformed towards the reference distribution so that peaks of hue agree with the reference image's hue peaks: output pixels exhibit colors which tend to follow the reference distribution.

Our method has other limitations. Like Winnemöller's bilateral filtering method, our method needs to estimate visual salience. Certain high-contrast features such as specular highlights that may be emphasized by our framework are actually deemphasized in human vision, and repeated texture causes unwanted edge responses. Another limitation lies in our handling

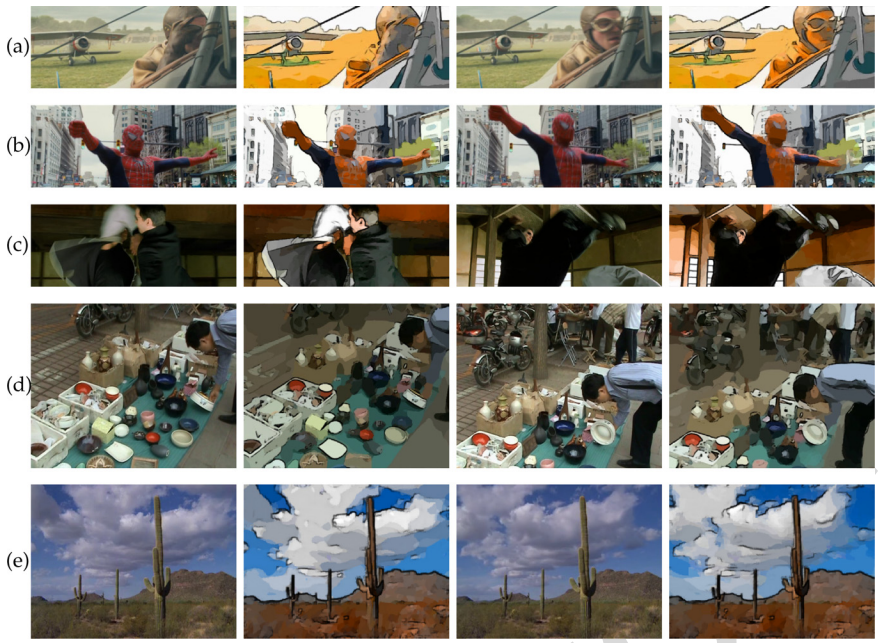

Fig. 6. Our results of video abstraction. Columns 1, 3: original video frames. Columns 2, 4: corresponding abstraction results of columns 1, 3.

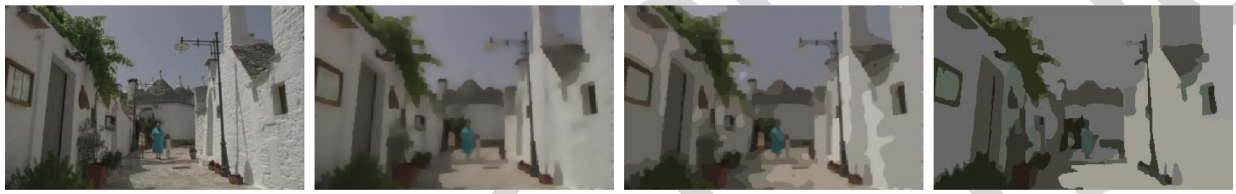

Fig. 7. Comparison of abstraction effects. From left to right: source image, result using filtering-based method, color quantization result using filtering-based method, result using our method.

TABLE I
SCORE OF THE USER STUDY

	Fig.1	Fig.6(a)	Fig.6(b)	Fig.6(c)	Fig.6(d)	Fig.6(e)
Cartoon shape style	4.5	3.9	3.7	2.8	4.7	3.5
Result with mean color	3.0	2.3	2.8	3.0	3.3	2.4
Result with color replacement	3.7	4.0	3.9	3.2	4.1	3.5
Temporal coherence of our result	4.5	4.2	4.2	3.1	4.3	3.7
Temporal coherence of [4]	4.4	4.1	3.5	3.7	4.3	3.9

of occluded regions. When such regions are gradually uncovered by an overlying object, they typically initially merge with the overlying object, and then suddenly separate from it when they have grown to a sufficient size. This may cause jumps in color, although our approach in Section IV alleviates such effects to some extent.

VII. CONCLUSIONS

Our real-time video stream abstraction method uses a segmentation strategy guided by optical flow, with care taken to appropriately allow for optical flow errors, so that we can consistently segment each frame while preserving temporal coherence. To achieve attractive stylization, we use a color scheme replacement method which applies colors learnt from a video clip or an image.

Various areas exist for future work. Firstly, we intend to investigate more general ways of computing the importance map while providing temporal coherence, yet with sufficient performance for a real-time system. One obvious consideration is that moving objects are typically more important than the static background. Secondly, we also intend to consider vectorization of the stylized video. Currently our methods directly produce vectorized output on a *per-frame basis* in the form of a boundary representation of each region together with its color. However, determining *inter-frame* correspondences in terms of projection transformation of coherent regions, and region overlap, would allow more semantically useful vectorization, as well as enabling greater compression. Finally, we would like to extend this work to other styles of rendering, such as oil paintings.

Fig. 8. Label propagation by optical flow: 2 examples. From left to right: previous frame, current frame, residual map before morphological filtering, propagation map after morphological filtering.

Fig. 9. Color scheme replacement examples. Left to right, top: source image, reference image 1, color transferred from reference image 1, reference image 2, color transferred from reference image 2. Bottom: corresponding hue histograms in HSV space.

ACKNOWLEDGMENT

The authors would like to thank all the reviewers for their helpful comments.

REFERENCES

[1] D. DeCarlo and A. Santella, "Stylization and abstraction of photographs," in *Proc. 29th Annu. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH '02)*, 2002, pp. 769–776.

[2] A. Santella and D. DeCarlo, "Visual interest and NPR: An evaluation and manifesto," in *Proc. 3rd Int. Symp. Non-Photorealistic Animation and Rendering (NPAR '04)*, 2004, pp. 71–150.

[3] J. Fischer, M. Haller, and B. Thomas, "Stylized depiction in mixed reality," *Int. J. Virtual Reality*, vol. 7, no. 4, pp. 71–79, 2009.

[4] H. Winnemöller, S. C. Olsen, and B. Gooch, "Real-time video abstraction," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1221–1226, 2006.

[5] J. E. Kyprianidis and J. Döllner, "Image abstraction by structure adaptive filtering," in *Proc. EG UK Theory and Practice of Computer Graphics*, 2008, pp. 51–58.

[6] H. Kang, S. Lee, and C. K. Chui, "Flow-based image abstraction," *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 1, pp. 62–76, Jan. 2009.

[7] J. E. Kyprianidis, H. Kang, and J. Döllner, "Image and video abstraction by anisotropic Kuwahara filtering," *Comput. Graph. Forum*, vol. 26, no. 7, pp. 1955–1963, 2009.

[8] C.-K. Yang and H.-L. Yang, "Realization of Seurat's pointillism via non-photorealistic rendering," *Vis. Comput.*, vol. 24, no. 5, pp. 303–322, 2008.

[9] W.-M. Pang, Y. Qu, T.-T. Wong, D. Cohen-Or, and P.-A. Heng, "Structure-aware halftoning," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 89:1–89:8, 2008.

[10] F. Wen, Q. Luan, L. Liang, Y.-Q. Xu, and H.-Y. Shum, "Color sketch generation," in *Proc. 4th Int. Symp. Non-Photorealistic Animation and Rendering (NPAR '06)*, 2006, pp. 47–54.

[11] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," in *Proc. 25th Annu. Conf. Computer Graphics and Interactive (SIGGRAPH '98)*, 1998, pp. 453–460.

[12] A. Agarwala, "Snaketoonz: A semi-automatic approach to creating cel animation from video," in *Proc. 2nd Int. Symp. Non-Photorealistic Animation and Rendering (NPAR '02)*, 2002, vol. 163, pp. 139–146.

[13] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz, "Keyframe-based tracking for rotoscoping and animation," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 584–591, 2004.

[14] J. Wang, Y. Xu, H.-Y. Shum, and M. F. Cohen, "Video tooning," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 574–583, 2004.

[15] D. Dementhon, "Spatio-temporal segmentation of video by hierarchical mean shift analysis," in *Proc. Statistical Methods in Video Processing Workshop (SMVP '02)*, 2002.

[16] H. Hua, L. Zhang, and T.-N. Fu, "Video painting via motion layer manipulation," *Comput. Graph. Forum*, vol. 29, no. 7, pp. 2055–2064, 2010.

[17] S. Paris, "Edge-preserving smoothing and mean-shift segmentation of video streams," in *Proc. 10th Eur. Conf. Computer Vision (ECCV '08): Part II*, 2008, pp. 460–473.

[18] J. P. Collomosse, D. Rowntree, and P. M. Hall, "Stroke surfaces: Temporally coherent artistic animations from video," *IEEE Trans. Vis. Comput. Graph.*, vol. 11, no. 5, pp. 540–549, Sep./Oct. 2005.

[19] P. C. Litwinowicz, "Processing images and video for an impressionist effect," in *Proc. 24th Annu. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH '97)*, 1997, pp. 407–414.

[20] A. Hertzmann and K. Perlin, "Painterly rendering for video and interaction," in *Proc. 1st Int. Symp. Non-Photorealistic Animation and Rendering (NPAR '00)*, 2000, pp. 7–12.

[21] A. Hertzmann, "Paint by relaxation," in *Proc. Computer Graphics Int. 2001*, 2001, pp. 47–54.

[22] L. Kovács and T. Szirányi, "Creating animations combining stochastic paintbrush transformation and motion detection," in *Proc. 16th Int. Conf. Pattern Recognition (ICPR'02): Part II*, 2002, pp. 1090–1093.

[23] J. Hays and I. Essa, "Image and video based painterly animation," in *Proc. 3rd Int. Symp. Non-Photorealistic Animation and Rendering (NPAR '04)*, 2004, pp. 113–120.

[24] D. Vanderhaeghe, P. Barla, J. Thollot, and F. Sillion, "Dynamic point distribution for stroke-based rendering," in *Proc. Eurographics Symp. Rendering*, 2007, pp. 139–146.

[25] A. Bousseau, F. Neyret, J. Thollot, and D. Salesin, "Video watercolorization using bidirectional texture advection," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 104:1–104:7, 2007.

[26] G. R. Greenfield and D. H. House, "Image recoloring induced by palette color associations," *J. Winter School Comput. Graph. (WSCG)*, vol. 11, pp. 189–196, 2003.

[27] Y. Chang, S. Saito, and M. Nakajima, "A framework for transfer colors based on the basic color categories," in *Proc. Computer Graphics Int. 2003*, 2003, pp. 176–181.

[28] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *Proc. 28th Annu. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH '01)*, 2001, pp. 327–340.

[29] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley, "Color transfer between images," *IEEE Comput. Graph. Appl.*, vol. 21, no. 5, pp. 34–41, Sep./Oct. 2001.

[30] H. Hua, Y. Zang, and C.-F Li, "Example-based painting guided by color features," *Vis. Comput.*, vol. 26, no. 6–8, pp. 933–942, 2010.

[31] Y.-H. H. C.-M. Wang and Y.-H. Huang, "An effective algorithm for image sequence color transfer," *Math. Comput. Model.*, vol. 44, no. 7–8, pp. 608–627, 2006.

[32] C. L. Zitnick, N. Jojic, and S. B. Kang, "Consistent segmentation for optical flow estimation," in *Proc. 10th IEEE Int. Conf. Computer Vision (ICCV '05): Part II*, 2005, pp. 1308–1315.

[33] M. P. Kumar, P. H. S. Torr, and A. Zisserman, "Learning layered motion segmentation of video," *Int. J. Comput. Vis.*, vol. 76, no. 3, pp. 301–319, 2008.
[34] J. Xiao and M. Shah, "Motion layer extraction in the presence of occlusion using graph cuts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 10, pp. 1644–1659, Oct. 2005.
[35] S. Baker, S. Roth, D. Scharstein, M. Black, J. Lewis, and R. Szeliski, "A database and evaluation methodology for optical flow," in *Proc. 11th IEEE Int. Conf. Computer Vision (ICCV '07)*, 2007, pp. 1–8.
[36] C. Zach, T. Pock, and H. Bischof, "A duality based approach for realtime tv-l1 optical flow," in *Proc. 29th DAGM Conf. Pattern Recognition*, 2007, pp. 214–223.
[37] 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.
[38] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," in *Proc. IEEE Int. Conf. Image Processing 2002*, 2002, pp. 900–903.
[39] Y. Zhai and M. Shah, "Visual attention detection in video sequences using spatiotemporal cues," in *Proc. 14th Annu. ACM Int. Conf. Multimedia (MULTIMEDIA '06)*, 2006, pp. 815–824.
[40] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, Nov. 1986.
[41] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, pp. 603–619, May 2002.
[42] D. Cohen-Or, O. Sorkine, R. Gal, T. Leyvand, and Y.-Q. Xu, "Color harmonization," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 624–630, 2006.
[43] N. Sawant and N. Mitra, "Color harmonization for videos," in *Proc. 6th Indian Conf. Computer Vision, Graphics & Image Processing (ICVGIP '08)*, 2008, pp. 576–582.

Song-Hai Zhang received the Ph.D. degree from Tsinghua University, Beijing, China, in 2007.

He is currently a lecturer of computer science at Tsinghua University. His research interests include image and video processing as well as geometric computing.

Xian-Ying Li received the B.S. degree in computer science from Tsinghua University, Beijing, China, in 2008. He is now pursuing the Ph.D. degree at Tsinghua University.

He received a gold medal of the International Mathematical Olympiad (IMO) in 2004 when he was 16 years old. His research interests include computer graphics, geometric modeling, and computational origami.

Shi-Min Hu (M'96) received the Ph.D. degree from Zhejiang University, Hangzhou, China, in 1996.

He is currently a chair professor of computer science in the Department of Computer Science and Technology at Tsinghua University, Beijing, China. His research interests include digital geometry processing, video processing, rendering, computer animation, and computer-aided geometric design.

Dr. Hu is on the editorial boards of *Computer Aided Design*, *The Visual Computer*, and *Computer & Graphics*.

Ralph R. Martin (M'??) **AUTHOR: WHAT YEAR?** received the Ph.D. degree in 1983 from Cambridge University, Cambridge, U.K.

Since, then he has been at Cardiff University, as a Professor since 2000, where he leads the Visual Computing research group. He is also a Guest Professor at Tsinghua and Shandong Universities in China, and Director of Scientific Programmes of the One Wales Research Institute of Visual Computing. His publications include over 200 papers and 12 books covering such topics as solid modeling, surface modeling, reverse engineering, intelligent sketch input, mesh processing, video processing, computer graphics, vision-based geometric inspection, and geometric reasoning.

Dr. Martin is a Fellow of the Learned Society of Wales, the Institute of Mathematics and its Applications, and the British Computer Society. He is on the editorial boards of *Computer Aided Design*, *Computer Aided Geometric Design*, *Geometric Models*, the *International Journal of Shape Modeling*, *CAD and Applications*, and the *International Journal of CAD/CAM*.

Online Video Stream Abstraction and Stylization

Song-Hai Zhang, Xian-Ying Li, Shi-Min Hu, *Member, IEEE*, and Ralph R. Martin, *Member, IEEE*

***Abstract*—This paper gives an automatic method for online video stream abstraction, producing a temporally coherent output video stream, in the style with large regions of constant color and highlighted bold edges. Our system includes two novel components. Firstly, to provide coherent and simplified output, we segment frames, and use optical flow to propagate segmentation information from frame to frame; an error control strategy is used to help ensure that the propagated information is reliable. Secondly, to achieve coherent and attractive coloring of the output, we use a color scheme replacement algorithm specifically designed for an online video stream. We demonstrate real-time performance for CIF videos, allowing our approach to be used for live communication and other related applications.**

***Index Terms*—Abstraction, color scheme replacement, optical flow, segmentation, temporal coherence, video stream.**

I. Introduction

THE abstraction of images and videos becomes popular in non-photorealistic rendering, which simplifies scene information while retaining meaningful features in a stylistic fashion, so that the output is nice looking, and can serve the purpose of communicating the messages and reducing the perceptual effort to understand the contents.

Segmentation is a natural choice of tool for abstraction. DeCarlo *et al.* [1], [2] propose an *image* abstraction and stylization approach that transforms images into cartoon style using large regions of constant color, by a hierarchical segmentation. We also pursue this artistic style, and also use explicit image structure, but produce temporally coherent cartoon-like *animations* . The particular goal of our video stream abstraction approach is to produce an output style which is cartoon-like, having simplified regions with user-guided colors, and high contrast, which has many applications to areas such as live broadcast and communications, video games, entertainment, and virtual reality [3].

However, temporally coherent video stream abstraction is a challenging task. Firstly, existing segmentation-based methods require intensive user interaction to produce high-quality artistic results, and have a high computational cost. Although there are some methods [4]–[7] using feature-preserving smoothing filters to produce stylized videos automatically, their results are smooth shaded, and more like the painterly effects rather than cartoon-like effects. Secondly, processing video streams requires efficient algorithms in order to cope with real-time data. Offline algorithms can make use of future as well as past information, but this is not available in video streams. Furthermore, video streams have to be processed frame by frame, or at most make use of just a few past frames or an averaged history.

In this work, we present an online, automatic method to generate cartoon-like abstract animation from an input video stream (see Fig. 1). Little user interaction is required, other than selection of a reference video which determines preferred output colors (or alternatively a hue histogram), and setting of preferences which control the amount of detail retained in colored regions and line drawing before processing. Our approach benefits from two novel aspects:

- A segmentation strategy which uses optical flow to propagate segmentation information from frame to frame, with an error control strategy to help ensure that the propagated information is reliable. The segmentation constructs an explicit structure, which provides correspondences of matched regions, and achieves the temporally coherent abstraction results of videos.
- A video stream color scheme replacement method that does not require complete knowledge of the source color distribution (i.e., does not need to know future frames), and which applies the color scheme from a reference video (or image, or a user designed histogram of hue) to the input video, while keeping color consistency between frames thanks to the explicit structure of the video.

II. Related Work

A number of significant papers have addressed stylization, but most of them concentrate on images rather than videos [1], [8]–[11]. Much research has also considered the generation of cartoon-like video from real-world captured video with the aim of reducing the amount of work required for cartoon production, and the need for skilled artists. Examples include papers on "SnakeToonz" [12], keyframe based rotoscoping [13], and "VideoTooning" [14]. Such methods produce high-quality results, but still require intensive user interaction, and generally restart video processing every few tens of frames to avoiding error accumulation. Such methods typically require considerable artistic skill for good results.

DeCarlo and Santella [1], [2] propose an *image* abstraction and stylization approach relying on eye-tracking data to guide image simplification or stylization, using hierarchical segmentation, which well preserves the image structure. We pursue an

Manuscript received September 23, 2010; accepted July 29, 2011. Date of publication August 18, 2011; date of current version November 18, 2011. This work was supported in part by the National Basic Research Project of China (Grant No. 2011 CB302205), in part by the National Natural Science Foundation of China (Grant Nos. 60970100, 61033012), and in part by the EPSRC Travel Grant. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Nadia Magnenat-Thalmann.

S.-H. Zhang, X.-Y. Li, and S.-M. Hu are with the National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: shz@tsinghua.edu.cn; shimin@tsinghua.edu.cn).

R. R. Martin is with the School of Computer Science Informatics, Cardiff University, Cardiff, Wales, CF24 3 AA, U.K. (e-mail: Ralph.Martin@cs.cardiff.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2011.2165052

Fig. 1. Abstraction example. 1: original video frame, and 2–4: three frames of the stylized video with color scheme replacement.

artistic style similar to [1], and also use explicit image structure, but produce temporally coherent cartoon-like *animations*.

As a natural scheme to extend DeCarlo *et al.*'s work, several video abstraction approaches have been proposed based on the coherent video segmentation. We should notice that video segmentation is a fundamental and challenging problem as surveyed in [15], which has classified video segmentation techniques into three categories: the 3 D stack scheme, the $2D + t$ scheme, and the trajectory clustering scheme. For online video streams, which ask the algorithm to be fully automatic and do not need to know future frames, the 3 D stack scheme [14], motion layer segmentation scheme [16], and trajectory clustering scheme methods are not preferred because they all need costly whole-video analysis. For $2D+t$ scheme, Paris [17] uses isotropic diffusion and Gaussian convolution for real-time video stream segmentation; Collomosse *et al.* [18] preform a key-frame-based approach. However, these methods adopt a simple frame alignment, which limits the coherence for fast-moving scenes and objects. Actually, as Paris mentioned in the paper, using optical flow may provide better results on moving boundaries.

Other than segmentation-based approaches, various filtering techniques have been applied for both image and video abstraction [4]–[7]. Those works use image filters instead of segmentation, which are fast in practice and naturally fit to videos and video streams. Their coherence are guaranteed by the implicit image structure itself, and the stability of the filters. But different from the cartoon-like effects that we pursue, they tend to produce painterly effects.

In this paper, we use optical flow to guide coherent video segmentation for cartoon-like effects. Our results achieve comparable coherence as those filtering-based methods and, what is more, as we have constructed an explicit representation with region correspondences, performing coherent and stable color replacement becomes possible.

There are several papers [19]–[24] using optical flow to direct brush stroke movement or point placement during frames, but they do not take into account errors in optical flow, so limited coherence is achieved in image sequences. Bousseau *et al.* [25] also use optical flow in a watercolor generation pipeline, but rely on temporally bi-directional optical flow interpolation to reduce optical flow errors, an option which is not available in a live video system as future information is unavailable.

Several works consider the problem of color scheme extraction [26] and color transfer [27]–[30]. These mainly focus on image processing, and analyze source and reference features to determine the color transformation. However, we cannot extend such a method directly to live video stream color transfer, again because of the lack of future information. On the other hand, performing image color transfer for each frame independently cannot guarantee color consistency in the output as source colors vary in each frame. Wang *et al.* [31] give a color transfer method for still images forming a sequence; parameters are adjusted to present gradually changing effects. Image analogies have also been used for color transfer [28]. These take into account local similarities in the image, and process it pixel by pixel. The latter is most suited to producing complex textures, but our output style uses large regions of slowly varying color. Thus, we have instead devised a new efficient color replacement method which captures the color style from a given reference video, and applies it with temporal coherence to produce a stylized video stream without the need for source video feature analysis.

III. Overview

Our goal is to replace incoming live video by abstract animation, in real time. We must thus avoid time-consuming non-local analysis of the video (e.g., background reconstruction). During abstraction, our three goals are to:

- simplify the content, without oversimplifying important areas;
- modify the coloring in a controllable manner;
- retain temporal coherence.

Taking these into account, our pipeline is as shown in Fig. 2, and includes six steps.

Of these six steps, importance map computation, optical flow computation, and edge detection provide inputs for coherent image segmentation. The importance map is used to ensure greater detail in key areas. As optical flow information is inherently unreliable, we must be careful to avoid propagating and accumulating optical flow errors during segmentation. We use a careful error control strategy when propagating segmentation labels from frame to frame, to ensure temporal coherence of region representations between frames.

For simple output, we could simply replace each region by its mean color. Better results are obtained by using a color scheme replacement process based on a desired "color style" learnt offline from a reference video (or image, or provided by the user as a histogram). We use a mean-shift-based color transformation to move input colors closer to a reference color distribution. We are careful to preserve color consistency between frames, while still not needing to know the potential colors in future frames.

Finally, we smooth boundary curves using a low pass filter to produce more artistic effects, and then use a difference-of-Gaussians (DoG) operator on the input frames to detect and overlay edges, as in [4].

Algorithm 1 gives pseudocode for the whole algorithm, and details of image segmentation are described in Sections IV–VII.

Algorithm 1 Pseudocode for online video abstraction

while (! end of video stream) **do**
  //—Perform coherent image segmentation—
  Compute importance map;
  Compute Canny edges;
  if !(first frame) **then**
    //—Update segmentation using optical flow—
    Compute optical flow;
    Propagate labels from previous frame;
    Apply morphological filtering to those propagated labels;
    Grow regions;
  end if
  for $(r = \text{initial radius}; r \geq 1; r/ = 2)$ **do**
    Do trapped-ball filling with ball radius r;
    Grow regions;
  end for
  //—End of image segmentation—
  Apply color scheme replacement;
  Smooth regions;
  Compute DoG edges and overlay them;
end while

IV. Coherent Image Segmentation

As in [1], we perform segmentation in order to simplify image content. Several approaches exist for temporally coherent video segmentation [32]–[34]. Most solve the problem via optimization or a mean-shift approach, and thus perform a non-local analysis of the video. To process a *live* video stream, we perform a $2D + t$ segmentation by propagating information from one frame to the next.

Our approach uses optical flow but in a way which avoids accumulation of errors. Many optical flow computation methods exist [35]; we use Zach's [36] as it provides good results in real time. The first frame is segmented using the trapped-ball algorithm [37]; this algorithm is explained later. In each successive frame, optical flow is used to propagate segmentation labels for each pixel. In the result, some pixels may have *no* segmentation label (e.g., because they have been newly exposed from an occluded area), or may have *multiple* labels (e.g., because of optical flow errors). Labeling such pixels is the key to coherent image segmentation. Those which are sufficiently close in distance and color to an existing region are given the label of that region, while the remainder are segmented into *new* regions again using the trapped-ball algorithm. Algorithm 1 gives pseudocode for image segmentation; we now consider these steps in detail.

A. Importance Map and Edge Detection

Edges provide strong hints as to where region boundaries should exist. In image and video abstraction, *important areas*, such as faces, should be depicted in more detail than other areas, which requires segmenting them into more, smaller regions. Eye tracking [4] can find important areas, but is of limited applicability. We compute the importance map by faces detection [38]. Alternative strategies could also be used to compute the importance map, e.g., taking into account saliency and motion information [39], but at a higher computational cost. The important map is used to control the number of edges found during edge detection: the hysteresis thresholds of a Canny edge detector [40] are set inversely proportional to importance map values. In areas of greater importance, more edges are detected.

B. Region Label Propagation by Optical Flow

The first frame is segmented by the trapped-ball method based on the detected edges. For subsequent frames, the previous frame has been segmented, and optical flow has been computed; these are used to provide an initial segmentation for the current frame. The segmentation is represented by labeling pixels with positive integers corresponding to regions.

We start by labeling all pixels of the current frame as 0, i.e., unassigned to any region. The optical flow is rounded to determine a spatial mapping of each source pixel from the previous frame to a target pixel in the current frame. This target pixel is given the same label as the source in the previous frame, except as below, when it retains its 0 label:

- the optical flow error is larger than a threshold;
- *more than one* source pixel ends up at this target pixel, and the labels of these source pixels are different;
- this target pixel has *no* corresponding source pixel.

To determine if an optical flow error has occurred, we compute the color difference in CIELAB color space between the source and a linear interpolation of the values of the 4-neighboring pixels of the target position. If this exceeds a threshold T_f, the target is labeled 0. T_f is chosen according to the video quality; values in the range 5–20 are typically appropriate. This prevents obvious optical flow failures. Fig. 3 (top, right) shows the residual unlabeled pixels (red) after label propagation. Note that we do not explicitly need to detect scene changes in the incoming video: in such cases, the optical flow will have large errors, causing many pixels to be labeled 0, and will hence be re-segmented.

After pixel label propagation, region boundaries are typically rather interlaced. These are smoothed, and other mislabelings due to errors in the optical flow corrected, using a morphological filter: if 5 or more of the 8-connected neighbors of the target pixel are the same, but differ from the target pixel, and if its label

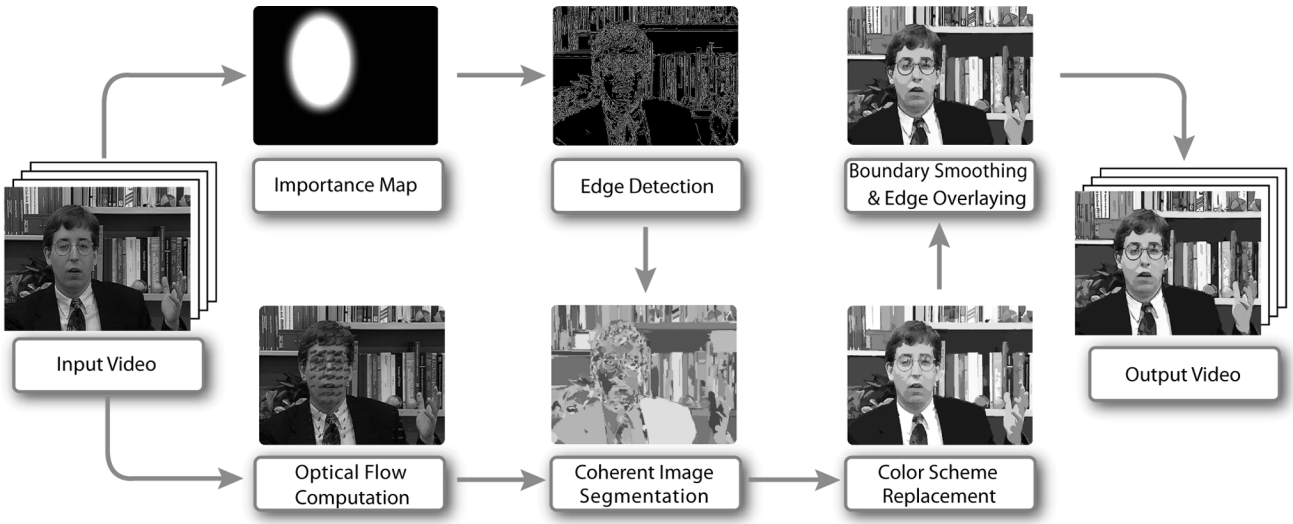

Fig. 2. Pipeline. Frame segmentation uses optical flow and edges as inputs, tuned by an importance map, to produce coherent regions. The color scheme replacement step transforms region colors according to a reference video.

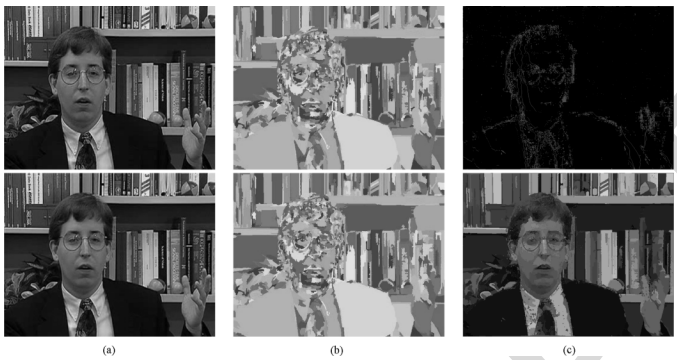

Fig. 3. Left: input frames 1, 2. Middle: segmentation results for frames 1, 2. Right-top: residual map before morphological filter. Right-bottom: region label propagation result after morphological filter.

is 0, then it is changed to the most frequent label in the neighborhood; otherwise, it is changed to 0. Fig. 3 (bottom, right) shows the propagated labels after morphological filtering, where red pixels are unlabeled.

We must now complete the segmentation by labeling the unlabeled pixels. This is a problem of minimizing label errors for those unlabeled pixels which should be allocated to existing regions, and giving new labels to those pixels which are insufficiently similar to existing regions. At the same time, the labels assigned must respect the edge information, so that the regions are in agreement with any superimposed edges drawn later. These requirements, taken together with efficiency considerations, make it difficult to solve the problem by traditional optimization or graph cut methods. Instead, we improve the region growing and trapped-ball filling [37] to finish the segmentation.

C. Region Growing

For each labeled region, a single, simple, color model is assumed to be an adequate fit. We use a constant color model, and assume pixels belonging to a region are near to its mean color.

We assign labels while respecting edges taking into account that 1) edge pixels may not be added to a region and 2) pixels added to a region should agree with its color to within perceptual limits. Let the *reconstruction error* of a labeled pixel be the difference between its actual color and that predicted by the corresponding color model. All unlabeled pixels adjacent to the boundary of any labeled region are put into a priority queue, sorted by reconstruction error with respect to the adjacent region. We then pop the pixel with minimum reconstruction error from the queue. If the reconstruction error is sufficiently small [in practice, below $20(1.2 - I(p))$ units in CIELAB space], the pixel's label is updated, and it is removed from the priority queue; at the same time, its unlabeled 4-connected neighbors are added to the queue. We repeatedly pop pixels until the queue is empty, or the least reconstruction error is too large.

D. Trapped-Ball Filling

Any pixels still remaining unlabeled belong to some *new* region. In general, multiple new regions may exist, so these unlabeled pixels still need to be segmented.

The basic idea of trapped-ball filling is to move a ball around, limited by a mask, formed here by the already labeled pixels and the detected edges. Each separate area in which the ball may move determines a new region of the image. The trapped-ball has the advantage over flood-filling that it cannot leak out between short gaps in the detected edges. Initially, a large ball is used to find each region's core, and then balls of successively smaller radius are used down to a radius of 1 to add more detailed parts of each region.

The initial ball radius is chosen according to the number of unlabeled pixels N_0. We set it to $\max(2, \sqrt{N_0}/30)$, which is typically in the range 2–32. In the first frame, *all* pixels are unlabeled, and an original segmentation for the whole frame is obtained using the trapped-ball method with a *large* initial ball size. In subsequent frames, unlabeled regions are typically small, and a *smaller* initial ball radius is used. If a scene change occurs, many pixels are now unlabeled, so a larger initial ball

size is again used. The extra processing entailed at a scene change can cause a slight lag, but as long as frames are processed faster than in real time, this lag can soon be overcome.

V. Color Scheme Replacement

Color selection is an important aspect of stylization, and different palettes can produce different emotional responses. Much work has been done on color transfer for images—applying the color scheme from a reference image to a source image. Clearly, in this case, the color distribution for the *entire* source and reference are known. However, when processing *live* video, we do not have complete information: future video content is unknown. Thus, our recoloring problem has two specific requirements: the transformation should cover the entire color space to allow for *any* possible future colors, and as we have video rather than images, we must also ensure inter-frame color stability for each region in the video.

Our color scheme replacement method uses the concept of *mean shift* [41]. Given a reference image or cartoon video clip, we count the number of pixels of each color to determine the color distribution. We must also choose the desired number of iterations and kernel radius for the mean shift procedure. Then, to determine the mapping of each input color to an output color, we determine a mean shift vector towards a local maximum in the color distribution, which shifts each input pixel's color towards the nearest peak in the reference distribution. Overall, given an input color distribution, the mapping transforms it towards the reference color distribution.

It would be costly to compute the reference distribution and mapping directly in CIELAB color space. Instead, we simply consider the hue value in HSV space, and compute the reference distribution and mean shift vector using H alone. In an analogous manner, [42] and [43] also use hue as the most important feature of color when performing color harmonization, i.e., mapping a source color distribution to a template distribution.

As these authors note, there is discontinuity in new color when two nearby colors are transformed to different local maxima, which may causing flickering between frames. Thus, we also use an interframe blending method which takes into account correspondence between regions in successive frames; this information is provided by our segmentation-based explicit structure, which is never achieved by filtering-based methods. Thus, after first computing an ideal *target color* by the mean shift procedure, we modify it to give the *actual output color* by interframe blending.

To find the color mapping, we first compute the color distribution of the reference video in HSV color space (which is more useful than RGB from a perceptual point of view), considering the three channels separately. In each frame, for the H channel, we compute a color histogram for all pixels having $S > 0.125$ and $V > 0.125$ (because the H channel is senseless for those colorless and dark pixels). Also we compute the mean μ_S and standard deviation σ_S for S channel pixels having $V > 0.125$ (also, S channel is senseless for dark pixels), and the mean pixel value μ_V of the V channel. (Alternatively, instead of providing a reference video, the user could provide a reference image, or even manually design a color histogram represented in this way.)

For each frame of the incoming video, all pixels in each segmented region are given the same target color. The *source color* (h, s, v) for a region is computed as the average color of all pixels in that region. This is mapped to a target color (h', s', v'). First, we compute μ_s, σ_s, and μ_v for this *frame* in an analogous manner to how they were computed for the reference video. We then compute (h', s', v') as follows. h' is set to the mean shift of h in the H channel histogram, by iterating the formula below k times; typically $k = 3$:

$$h_{i+1} = \frac{\sum_{c \in N(h_i)} cD(c)}{\sum_{c \in N(h_i)} D(c)}. \tag{1}$$

Here h_0 is h_t, and h_4 is the desired h'. $N(h_i)$ represents a 30° neighborhood in the histogram and $D(c)$ represents the histogram value for c. (This function is pre-computed for all possible h values and stored in a look-up table for speed.) As for the S and V channels, we simply perform the following formulas for s' and v' with the aim of adjusting the brightness and contrast to be closer to those of the reference video:

$$s' = \mu_S + (s - \mu_s)\frac{\sigma_S}{\sigma_s} \tag{2}$$

$$v' = v\left(\lambda + (1 - \lambda)\frac{\mu_V}{\mu_v}\right). \tag{3}$$

Here a linear function is adopted for s' to achieve target mean and standard deviation. For v', a direct proportional function is adopted instead because we should make sure that the darkness is still darkness, where λ is a constant to measure "how much we want the video retains its original brightness variation". In this paper, we choose $\lambda = 2/3$ for all results.

After obtaining the *target color* , we now compute the *actual output color* . Suppose some region R_i has a target color c_i, and the corresponding region R_{i-1} in the previous frame had actual output color o_{i-1}. We compute its actual output color o_i by color blending to improve color coherence:

$$o_i = (1 - \alpha(R_i, R_{i-1}))o_{i-1} + \alpha(R_i, R_{i-1})c_i. \tag{4}$$

Here $\alpha(R_i, R_{i-1})$ measures the correlation of the two corresponding regions: $\alpha(R_i, R_{i-1})$ is the ratio of: the number of pixels in R_{i-1} whose destination lies in R_i after optical flow, to the geometric average of the areas of R_{i-1} and R_i. If region R_i is a *new* region in this frame, we simply set $o_i = c_i$. This blending is executed in the CIELAB space.

An H channel color histogram extracted from a reference video and an example of color scheme replacement are shown in Fig. 4 (final results after boundary smoothing and edge overlaying are shown in Fig. 5).

VI. Results and Discussion

We have implemented our framework using a combination of CPU and GPU, on an Intel Core2 Quad Q9300 CPU at 2.5 GHz, with 4 GB memory, and a GeForce 8600 GT, using CUDA, OpenCV, and Visual C++ 2008's parallelizing compiler. Performance depends on image size and framework parameters. For a CIF (352×288) video stream, commonly used for live

Fig. 4. Color scheme replacement. Top-left: a frame of the reference video. Top-right: histogram of its H channel. Bottom-left: segmented frame before replacement. Bottom-right: frame after replacement.

Fig. 5. Boundary smoothing and edge overlaying. Top-left: segmentation result. Top-right: after boundary smoothing. Bottom-left: detected lines. Bottom-right: smoothed result after edge overlay.

communication, face detection and edge detection is done on one CPU core, taking 40 ms per frame, while simultaneously, optical flow computation takes 20 ms on the GPU. Image segmentation takes from 20–30 ms, depending on the number of residual unlabeled pixels. Color scheme replacement and edge overlay take under 10 ms. Boundary smoothing takes 20–50 ms, depending on the number of boundary pixels. Typically we can process a CIF video stream with default parameter settings at about 9–12 frames per second using the CPU for everything but optical flow (we directly use Zach's GPU optical flow code [36]). We note that edge detection, morphological operations, and the DoG filter could also benefit from GPU implementation, permitting faster frame rates or higher video resolution. The current Nvidia GTX295 graphics card offers perhaps $10\times$ the performance of the older GeForce 8600 GT card.

Figs. 1 and 6 show various abstraction results. Our method achieves cartoon-like effects with simplified content comprising well-defined regions with bold boundaries, while using a chosen color style. Fig. 7 shows an example of the filtering-based method [4], which produces an painterly effect, and cannot achieve coherent cartoon-like effects by simple color quantization. In contrast, our results retain temporal coherence on both shape and color, benefited from the coherent image segmentation.

Different styles can be produced by the user, but objective evaluation of the level of success is difficult. Our results have been independently evaluated by a company who randomly chose 10 employees to subjectively evaluate 6 video clips of difference contents according to the temporal coherence, the cartoon shape style, and the cartoon color effect. The 10 participants include 5 males and 5 females, 3 professional designers and 7 novices, and their ages are between 20 and 40. Table I shows the average score of each video clip on three aspects. The result of the study shows that our method produces cartoon effects for streets and natural scenes better than the figures. The color scheme replacement greatly improves the stylization. The temporal coherence of our results is acceptable and comparable with [4].

Optical flow is the sole information used to guide interframe correspondence, so its accuracy is very important. Zach's method works well in most cases, as demonstrated by the number of residual pixels (Figs. 3 and 8). When the scene changes slowly, few residual pixels remain (Fig. 8, top), and most are labeled, according to color, during the region growing step. When the video content changes rapidly, the number of residual pixels increases. Indeed, essentially the whole image is composed of residual pixels at scene changes (Fig. 8, bottom), as useful optical flow information no longer exists. Thus, we do not need to explicitly detect scene changes: in cases with large numbers of residual pixels, trapped-ball filling provides resegmentation of the scene. While newly generated regions may cause certain inconsistencies, viewers find lack of coherence much more noticeable in slowly changing scenes than in fast moving scenes. Currently we use the rounded optical flow for label propagation, which may cause flickering in long and thin regions. In addition, image noise and motion blur may also decrease the quality of optical flow, and ultimately decrease the coherence of final results.

Fig. 9 illustrates the results of color scheme replacement using different reference images (these have been chosen to exaggerate the effects, and are atypical). The resulting hue distributions have been transformed towards the reference distribution so that peaks of hue agree with the reference image's hue peaks: output pixels exhibit colors which tend to follow the reference distribution.

Our method has other limitations. Like Winnemöller's bilateral filtering method, our method needs to estimate visual salience. Certain high-contrast features such as specular highlights that may be emphasized by our framework are actually deemphasized in human vision, and repeated texture causes unwanted edge responses. Another limitation lies in our handling

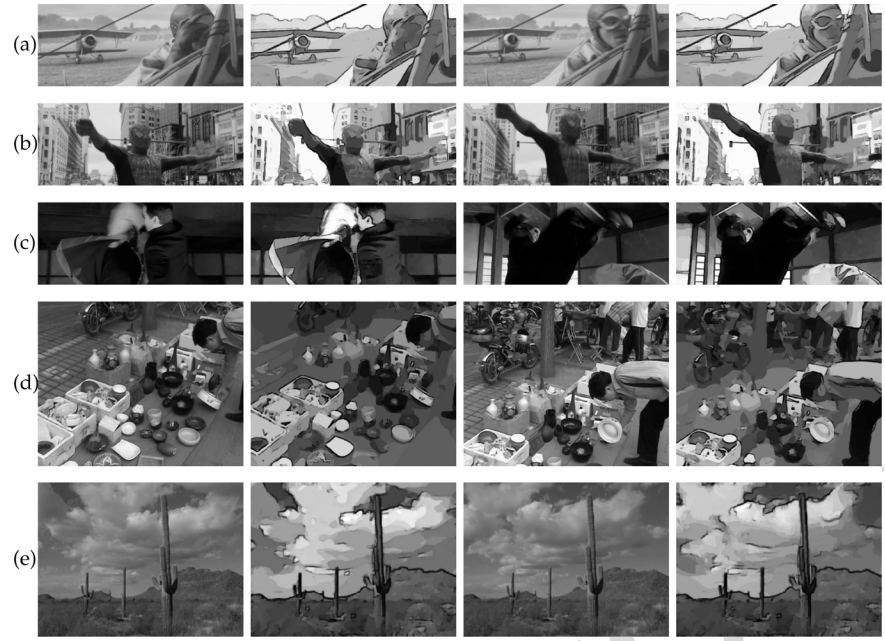

Fig. 6. Our results of video abstraction. Columns 1, 3: original video frames. Columns 2, 4: corresponding abstraction results of columns 1, 3.

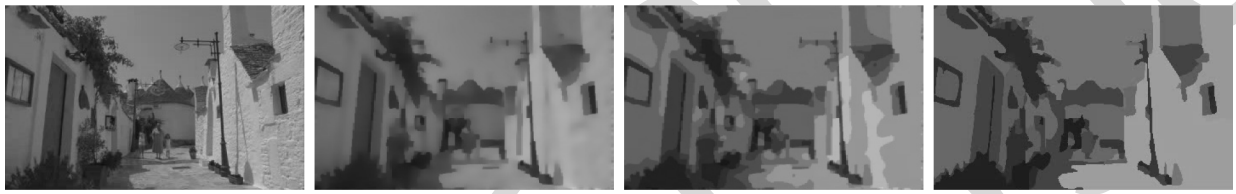

Fig. 7. Comparison of abstraction effects. From left to right: source image, result using filtering-based method, color quantization result using filtering-based method, result using our method.

TABLE I
SCORE OF THE USER STUDY

	Fig.1	Fig.6(a)	Fig.6(b)	Fig.6(c)	Fig.6(d)	Fig.6(e)
Cartoon shape style	4.5	3.9	3.7	2.8	4.7	3.5
Result with mean color	3.0	2.3	2.8	3.0	3.3	2.4
Result with color replacement	3.7	4.0	3.9	3.2	4.1	3.5
Temporal coherence of our result	4.5	4.2	4.2	3.1	4.3	3.7
Temporal coherence of [4]	4.4	4.1	3.5	3.7	4.3	3.9

of occluded regions. When such regions are gradually uncovered by an overlying object, they typically initially merge with the overlying object, and then suddenly separate from it when they have grown to a sufficient size. This may cause jumps in color, although our approach in Section IV alleviates such effects to some extent.

VII. CONCLUSIONS

Our real-time video stream abstraction method uses a segmentation strategy guided by optical flow, with care taken to appropriately allow for optical flow errors, so that we can consistently segment each frame while preserving temporal coherence. To achieve attractive stylization, we use a color scheme replacement method which applies colors learnt from a video clip or an image.

Various areas exist for future work. Firstly, we intend to investigate more general ways of computing the importance map while providing temporal coherence, yet with sufficient performance for a real-time system. One obvious consideration is that moving objects are typically more important than the static background. Secondly, we also intend to consider vectorization of the stylized video. Currently our methods directly produce vectorized output on a *per-frame basis* in the form of a boundary representation of each region together with its color. However, determining *inter-frame* correspondences in terms of projection transformation of coherent regions, and region overlap, would allow more semantically useful vectorization, as well as enabling greater compression. Finally, we would like to extend this work to other styles of rendering, such as oil paintings.

Fig. 8. Label propagation by optical flow: 2 examples. From left to right: previous frame, current frame, residual map before morphological filtering, propagation map after morphological filtering.

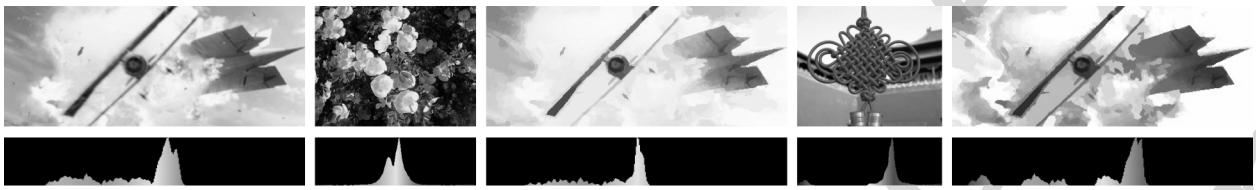

Fig. 9. Color scheme replacement examples. Left to right, top: source image, reference image 1, color transferred from reference image 1, reference image 2, color transferred from reference image 2. Bottom: corresponding hue histograms in HSV space.

ACKNOWLEDGMENT

The authors would like to thank all the reviewers for their helpful comments.

REFERENCES

[1] D. DeCarlo and A. Santella, "Stylization and abstraction of photographs," in *Proc. 29th Annu. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH '02)*, 2002, pp. 769–776.

[2] A. Santella and D. DeCarlo, "Visual interest and NPR: An evaluation and manifesto," in *Proc. 3rd Int. Symp. Non-Photorealistic Animation and Rendering (NPAR '04)*, 2004, pp. 71–150.

[3] J. Fischer, M. Haller, and B. Thomas, "Stylized depiction in mixed reality," *Int. J. Virtual Reality*, vol. 7, no. 4, pp. 71–79, 2009.

[4] H. Winnemöller, S. C. Olsen, and B. Gooch, "Real-time video abstraction," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1221–1226, 2006.

[5] J. E. Kyprianidis and J. Döllner, "Image abstraction by structure adaptive filtering," in *Proc. EG UK Theory and Practice of Computer Graphics*, 2008, pp. 51–58.

[6] H. Kang, S. Lee, and C. K. Chui, "Flow-based image abstraction," *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 1, pp. 62–76, Jan. 2009.

[7] J. E. Kyprianidis, H. Kang, and J. Döllner, "Image and video abstraction by anisotropic Kuwahara filtering," *Comput. Graph. Forum*, vol. 26, no. 7, pp. 1955–1963, 2009.

[8] C.-K. Yang and H.-L. Yang, "Realization of Seurat's pointillism via non-photorealistic rendering," *Vis. Comput.*, vol. 24, no. 5, pp. 303–322, 2008.

[9] W.-M. Pang, Y. Qu, T.-T. Wong, D. Cohen-Or, and P.-A. Heng, "Structure-aware halftoning," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 89:1–89:8, 2008.

[10] F. Wen, Q. Luan, L. Liang, Y.-Q. Xu, and H.-Y. Shum, "Color sketch generation," in *Proc. 4th Int. Symp. Non-Photorealistic Animation and Rendering (NPAR '06)*, 2006, pp. 47–54.

[11] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," in *Proc. 25th Annu. Conf. Computer Graphics and Interactive (SIGGRAPH '98)*, 1998, pp. 453–460.

[12] A. Agarwala, "Snaketoonz: A semi-automatic approach to creating cel animation from video," in *Proc. 2nd Int. Symp. Non-Photorealistic Animation and Rendering (NPAR '02)*, 2002, vol. 163, pp. 139–146.

[13] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz, "Keyframe-based tracking for rotoscoping and animation," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 584–591, 2004.

[14] J. Wang, Y. Xu, H.-Y. Shum, and M. F. Cohen, "Video tooning," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 574–583, 2004.

[15] D. Dementhon, "Spatio-temporal segmentation of video by hierarchical mean shift analysis," in *Proc. Statistical Methods in Video Processing Workshop (SMVP '02)*, 2002.

[16] H. Hua, L. Zhang, and T.-N. Fu, "Video painting via motion layer manipulation," *Comput. Graph. Forum*, vol. 29, no. 7, pp. 2055–2064, 2010.

[17] S. Paris, "Edge-preserving smoothing and mean-shift segmentation of video streams," in *Proc. 10th Eur. Conf. Computer Vision (ECCV '08): Part II*, 2008, pp. 460–473.

[18] J. P. Collomosse, D. Rowntree, and P. M. Hall, "Stroke surfaces: Temporally coherent artistic animations from video," *IEEE Trans. Vis. Comput. Graph.*, vol. 11, no. 5, pp. 540–549, Sep./Oct. 2005.

[19] P. C. Litwinowicz, "Processing images and video for an impressionist effect," in *Proc. 24th Annu. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH '97)*, 1997, pp. 407–414.

[20] A. Hertzmann and K. Perlin, "Painterly rendering for video and interaction," in *Proc. 1st Int. Symp. Non-Photorealistic Animation and Rendering (NPAR '00)*, 2000, pp. 7–12.

[21] A. Hertzmann, "Paint by relaxation," in *Proc. Computer Graphics Int. 2001*, 2001, pp. 47–54.

[22] L. Kovács and T. Szirányi, "Creating animations combining stochastic paintbrush transformation and motion detection," in *Proc. 16th Int. Conf. Pattern Recognition (ICPR'02): Part II*, 2002, pp. 1090–1093.

[23] J. Hays and I. Essa, "Image and video based painterly animation," in *Proc. 3rd Int. Symp. Non-Photorealistic Animation and Rendering (NPAR '04)*, 2004, pp. 113–120.

[24] D. Vanderhaeghe, P. Barla, J. Thollot, and F. Sillion, "Dynamic point distribution for stroke-based rendering," in *Proc. Eurographics Symp. Rendering*, 2007, pp. 139–146.

[25] A. Bousseau, F. Neyret, J. Thollot, and D. Salesin, "Video watercolorization using bidirectional texture advection," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 104:1–104:7, 2007.

[26] G. R. Greenfield and D. H. House, "Image recoloring induced by palette color associations," *J. Winter School Comput. Graph. (WSCG)*, vol. 11, pp. 189–196, 2003.

[27] Y. Chang, S. Saito, and M. Nakajima, "A framework for transfer colors based on the basic color categories," in *Proc. Computer Graphics Int. 2003*, 2003, pp. 176–181.

[28] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *Proc. 28th Annu. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH '01)*, 2001, pp. 327–340.

[29] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley, "Color transfer between images," *IEEE Comput. Graph. Appl.*, vol. 21, no. 5, pp. 34–41, Sep./Oct. 2001.

[30] H. Hua, Y. Zang, and C.-F Li, "Example-based painting guided by color features," *Vis. Comput.*, vol. 26, no. 6–8, pp. 933–942, 2010.

[31] Y.-H. H. C.-M. Wang and Y.-H. Huang, "An effective algorithm for image sequence color transfer," *Math. Comput. Model.*, vol. 44, no. 7–8, pp. 608–627, 2006.

[32] C. L. Zitnick, N. Jojic, and S. B. Kang, "Consistent segmentation for optical flow estimation," in *Proc. 10th IEEE Int. Conf. Computer Vision (ICCV '05): Part II*, 2005, pp. 1308–1315.

[33] M. P. Kumar, P. H. S. Torr, and A. Zisserman, "Learning layered motion segmentation of video," *Int. J. Comput. Vis.*, vol. 76, no. 3, pp. 301–319, 2008.
[34] J. Xiao and M. Shah, "Motion layer extraction in the presence of occlusion using graph cuts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 10, pp. 1644–1659, Oct. 2005.
[35] S. Baker, S. Roth, D. Scharstein, M. Black, J. Lewis, and R. Szeliski, "A database and evaluation methodology for optical flow," in *Proc. 11th IEEE Int. Conf. Computer Vision (ICCV '07)*, 2007, pp. 1–8.
[36] C. Zach, T. Pock, and H. Bischof, "A duality based approach for realtime tv-l1 optical flow," in *Proc. 29th DAGM Conf. Pattern Recognition*, 2007, pp. 214–223.
[37] 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.
[38] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," in *Proc. IEEE Int. Conf. Image Processing 2002*, 2002, pp. 900–903.
[39] Y. Zhai and M. Shah, "Visual attention detection in video sequences using spatiotemporal cues," in *Proc. 14th Annu. ACM Int. Conf. Multimedia (MULTIMEDIA '06)*, 2006, pp. 815–824.
[40] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, Nov. 1986.
[41] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, pp. 603–619, May 2002.
[42] D. Cohen-Or, O. Sorkine, R. Gal, T. Leyvand, and Y.-Q. Xu, "Color harmonization," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 624–630, 2006.
[43] N. Sawant and N. Mitra, "Color harmonization for videos," in *Proc. 6th Indian Conf. Computer Vision, Graphics & Image Processing (ICVGIP '08)*, 2008, pp. 576–582.

Song-Hai Zhang received the Ph.D. degree from Tsinghua University, Beijing, China, in 2007.

He is currently a lecturer of computer science at Tsinghua University. His research interests include image and video processing as well as geometric computing.

Xian-Ying Li received the B.S. degree in computer science from Tsinghua University, Beijing, China, in 2008. He is now pursuing the Ph.D. degree at Tsinghua University.

He received a gold medal of the International Mathematical Olympiad (IMO) in 2004 when he was 16 years old. His research interests include computer graphics, geometric modeling, and computational origami.

Shi-Min Hu (M'96) received the Ph.D. degree from Zhejiang University, Hangzhou, China, in 1996.

He is currently a chair professor of computer science in the Department of Computer Science and Technology at Tsinghua University, Beijing, China. His research interests include digital geometry processing, video processing, rendering, computer animation, and computer-aided geometric design.

Dr. Hu is on the editorial boards of *Computer Aided Design*, *The Visual Computer*, and *Computer & Graphics*.

Ralph R. Martin (M'??) **AUTHOR: WHAT YEAR?** received the Ph.D. degree in 1983 from Cambridge University, Cambridge, U.K.

Since, then he has been at Cardiff University, as a Professor since 2000, where he leads the Visual Computing research group. He is also a Guest Professor at Tsinghua and Shandong Universities in China, and Director of Scientific Programmes of the One Wales Research Institute of Visual Computing. His publications include over 200 papers and 12 books covering such topics as solid modeling, surface modeling, reverse engineering, intelligent sketch input, mesh processing, video processing, computer graphics, vision-based geometric inspection, and geometric reasoning.

Dr. Martin is a Fellow of the Learned Society of Wales, the Institute of Mathematics and its Applications, and the British Computer Society. He is on the editorial boards of *Computer Aided Design*, *Computer Aided Geometric Design*, *Geometric Models*, the *International Journal of Shape Modeling*, *CAD and Applications*, and the *International Journal of CAD/CAM*.