Online Video Stream Stylization

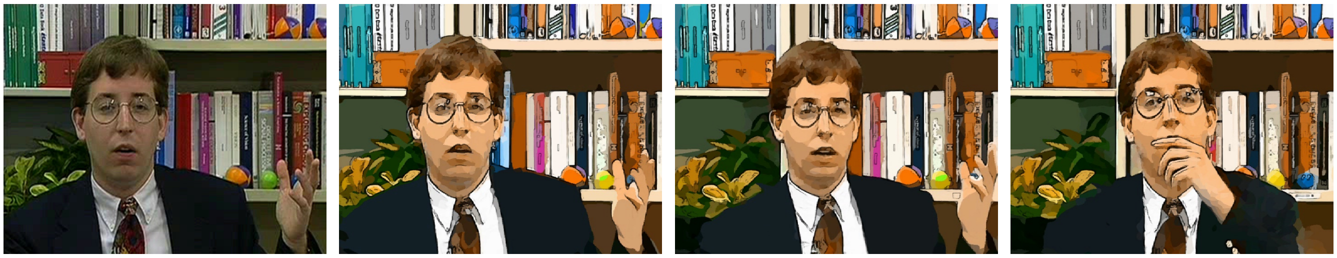

Figure 1: *Stylization example. 1: Original video frame; 2 - 4: Three frames of the stylized video with color scheme replacement.*

Abstract

This paper gives an automatic method for online video stream stylization, producing a temporally coherent output video stream. Our system transforms video into an abstract 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, allowing our approach to be used for live communication, video games, and related applications.

1 Introduction

Video stream processing has many applications to areas such as live broadcast and communications, video games, and entertainment. Stylization and abstraction of video streams are popular special effects in such live applications [Winnemöller et al. 2006], and can also reduce the perceptual and cognitive effort required to understand the content.

We give here an online, automatic stylization method which performs abstraction to generate cartoon-like animation from an input video stream. 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.

Various potential applications exist for such a method which can work in real time. In live communications, stylization may be used to save bandwidth. It may hide someone's identity or location, for example during an investigatory interview. It may be used to apply artistic effects to live scenes for tourist information. It may simply be used for fun in games or online chatting.

Processing live 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 live video streams. Furthermore, because of computational limits, such video has to be processed frame by frame, or at most make use of just a few past frames or an averaged history,.

The particular goal of our video stylization approach is to produce an output style which is cartoon-like, having simplified regions with user-guided colors, and high contrast. In comparison, Winnemöller [2006] produces a different artistic style based on simplified but smoothly shaded contents. Our cartoon-like style means that temporal coherence requirements are particularly strict.

Current stylization methods fall into three categories, each having limitations. Some methods focus on image processing and do not readily generalize to video. Others use simple image filters to achieve real-time performance, producing simplified and smoothly shaded contents, but the output may lack temporal coherence if input video streams are of low-quality. Yet others require significant user interaction to produce high-quality artistic results, and have a high computational cost.

We present a real-time system for a particular style of video stylization while providing good coherence. 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,
- 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.

2 Related work

A number of significant papers have addressed stylization, but most of them concentrate on images rather than video [DeCarlo and Santella 2002; Yang and Yang 2008; Pang et al. 2008; Wen et al. 2006; Hertzmann 1998]. 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' [Agarwala 2002], keyframe based rotoscoping [Agarwala et al. 2004], and videotooning [Wang et al. 2004]. 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 video-processing tools inherently work in an offline manner: to process any given frame they consider future as well as past information. Such methods typically require considerable artistic skill for good results.

DeCarlo [2002; 2004] proposed an *image* stylization approach relying on eye-tracking data to guide image simplification or stylization, using a hierarchical segmentation structure. We pursue an

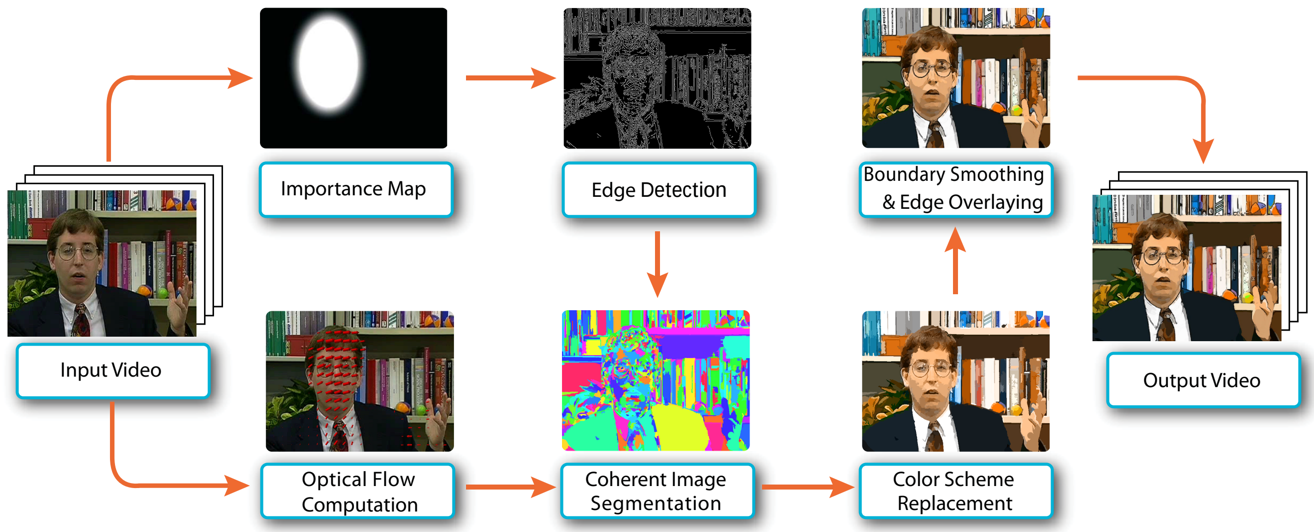

Figure 2: *Pipeline. Frame segmentation uses optical flow and edges as inputs, tuned by an importance map, to produce coherent regions. The color scheme replacement step transforms region colors according to a reference video.*

artistic style similar to DeCarlo [2002], and also use explicit image structure, but produce coherent cartoon-like *animations*.

Several video stylization systems have been designed with the main aim of helping artists with labor-intensive procedures [Wang et al. 2004; Collomosse et al. 2005]. Such systems use three-dimensional video volume segmentation, and are so computationally expensive that they must be used offline. Our method uses frames pairwise during segmentation, rather than costly multi-frame analysis, while being able to provide coherent online video stream stylization.

Other online video stylization methods exist. Fischer et al. [2009] use an automatic stylization method for augmented reality, applying stylization to both virtual and real inputs to fuse virtual objects into a live video stream. Winnemöller [2006] and Kyprianidis [2008] used simple image filters in real-time to process each frame into an abstract style. Neither method takes any particular steps to achieve temporal coherence, simply relying on the implicit image structure itself, and stability of filter outputs. Our method uses an explicit image structure representation which allows for stronger, better defined stylization in terms of region shapes, and explicitly propagates information from frame to frame to help achieve coherence. Kang [2009] proposed a flow based image abstraction algorithm which also considered shape smoothing, but it cannot be readily applied to video.

Paris [2008] give a coherent approach to analyzing and filtering video streams. Combining concepts of isotropic diffusion and Gaussian convolution, this method achieves temporal coherence via bilateral filtering and mean-shift segmentation in real time. However, as Paris notes, using optical flow can provide better results at moving boundaries.

Litwinowicz [1997], Hertzmann [2000; 2001], Kovács [2002], Hays [2004] and Vanderhaeghe [2007] use 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 [2007] also uses optical flow in a watercolor generation pipeline, but relies on temporally bidirectional 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 [Greenfield and House 2003] and color transfer [Chang et al. 2003; Hertzmann et al. 2001; Reinhard et al. 2001]. 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 [2006] gives 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 [Hertzmann et al. 2001]. 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, instead, we have 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.

Our method processes the video stream frame by frame, using a region based representation to achieve a cartoon-like style, and an optical flow based strategy to ensure coherence. Our method is designed to produce relatively highly stylized video, while still achieving real-time performance on a multi-core CPU. Compared to earlier methods such as videotooning, our method is much faster and needs much less user interaction.

3 Overview

Our goal is to replace incoming live video by stylized animation, in real time. We must thus avoid time consuming non-local analysis of the video (e.g. background reconstruction). During stylization, 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 Figure 2, and includes six key elements: importance map computation, op-

Algorithm 1 Pseudocode for online video stylization

```
while (! end of video stream) do
    //———Perform coherent image segmentation———
    Compute importance map;
    Compute Canny edges;
    if (not first frame) then
        //———Update segmentation using optical flow———
        Compute optical flow;
        Propagate labels from previous frame;
        Apply morphological filtering to 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
```

tical flow computation, edge detection, coherent image segmentation, color scheme replacement, and boundary smoothing with edge overlay.

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 [Winnemöller et al. 2006].

Algorithm 1 gives pseudocode for the whole algorithm, and details of image segmentation are described in the following section.

4 Coherent image segmentation

As in [DeCarlo and Santella 2002], we perform segmentation in order to simplify image content. Several approaches exist for temporally coherent video segmentation [Zitnick et al. 2005; Kumar et al. 2005; Xiao and Shah 2005]. 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 need a method which only uses past frames, and to keep processing requirements low, only a few previous frames can be considered. We perform 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 [Baker et al. 2007]; we use Zach's [Zach et al. 2007] as it provides good results in real time. The first frame is segmented using the trapped ball algorithm [Zhang et al. 2009]; 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.

Figure 3: *Edge detection controlled by importance map. From Left to right: input, importance map, Canny edge detection, detected edges controlled by importance map.*

4.1 Importance map and edge detection

Edges provide strong hints as to where region boundaries should exist. In image and video stylization, *important areas*, such as faces, should be depicted in more detail than other areas, which requires segmenting them into more, smaller regions. Eye tracking [DeCarlo and Santella 2002] can find important areas, but is of limited applicability. We detect and track faces [Lienhart and Maydt 2002]: an ellipse is found inside which pixels are assigned high importance, and outside, low importance; a smoothed real-valued map is used, as the face position is not entirely robust. Alternative strategies could also be used to compute the importance map, e.g. taking into account saliency and motion information [Zhai and Shah 2006], but at a higher computational cost.

Importance map I values lie between 0 and 1, with 1 being the most important. The important map is used to control the number of edges found during edge detection: the hysteresis thresholds of a Canny edge detector [Canny 1986] are set inversely proportional to importance map values. In areas of greater importance, more edges are detected, as shown in Figure 3.

4.2 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,

Figure 4: *Left: top, bottom: input frames 1, 2; Middle: top, bottom: segmentation results for frames 1, 2; Right: top: residual map before morphological filter; Right: bottom: region label propagation result after morphological filter.*

- 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. 4(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 resegmented.

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 is 0, then it is changed to the most frequent label in the neighborhood, otherwise it is changed to 0. Fig. 4(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 [Zhang et al. 2009] to finish the segmentation.

4.3 Region growing

For each labeled region, a single, simple, color model is assumed to be an adequate fit. For speed, 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 (i) edge pixels may not be added to a region and (ii) 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

Figure 5: *Color scheme replacement. Top, left: a frame of the reference video, right: histogram of its H channel; bottom, left: segmented frame before replacement, right: frame after replacement.*

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.

4.4 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 floodfilling 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 radii are used down to a radius of 1 to add more detailed parts of each region. In practice, the trapped ball approach is implemented using morphological erosion and dilation operations with a circular structuring element having the current ball radius.

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.

5 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* [Comaniciu and Meer 2002]. 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, [Cohen-Or et al. 2006] and [Sawant and Mitra 2008] 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 the image segmentation results. 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, first, we 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 $V > 0.125$. Also we compute the mean μ_S and standard deviation σ_S for S channel pixels having $V > 0.125$, 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 $k = 3$:

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

Here h_0 is h, 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 precomputed for all possible h values and stored in a look-up table for speed). Simple formulae are used for s' and v' with the aim of adjusting the brightness and contrast to be closer to those of the reference video:

$$\begin{aligned} s' &= \mu_S + (s - \mu_s)\sigma_S/\sigma_s, \\ v' &= v(2 + \mu_V/\mu_v)/3. \end{aligned}$$

This approach ensures that a given hue is always mapped to the same new hue in different frames. Such a mapping is not appropriate for s and v if the input video varies in brightness.

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:

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

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 $o_i = c_i$.

An H channel color histogram extracted from a reference video and an example of color scheme replacement are shown in Fig. 5 (final results after boundary smoothing and edge overlaying are shown in Fig. 6).

Figure 6: *Boundary smoothing and edge overlaying. Top, left: segmentation result, right: after boundary smoothing; bottom, left: detected lines, right: smoothed result after edge overlay.*

6 Results and discussion

We have implemented our framework using a combination of CPU and GPU, on an Intel Core2 Quad Q9300 CPU at 2.5GHz, with 4GB memory, and a GeForce 8600GT, 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 communication, face detection and edge detection is done on one CPU core, taking 40ms per frame, while simultaneously, optical flow computation takes 20ms on the GPU. Image segmentation takes from 20–30ms, depending on the number of residual unlabeled pixels. Color scheme replacement and edge overlay take under 10ms. Boundary smoothing takes 20–50ms, depending on the number of boundary pixels. Typically we can process a CIF video stream with default parameter settings at

Figure 7: *Video stylization results. Columns 1, 3: original video frames; columns 2, 4: corresponding stylization results.*

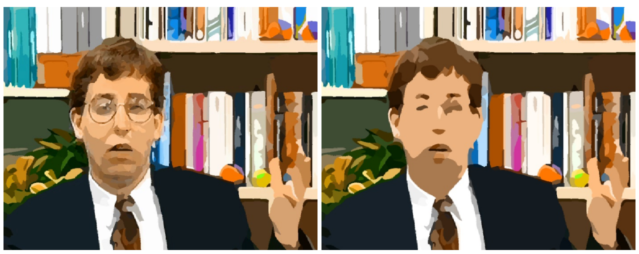

Figure 8: *Effect of importance map. Left: segmentation with importance map, right: segmentation without importance map.*

about 9–12 frames per second using the CPU for everything but optical flow (we directly use Zach's GPU optical flow code [Zach et al. 2007]). 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× the performance of the older GeForce 8600GT card.

Figs. 1 and 7 show various stylization results. The goals of stylization are subjective. Ours are to achieve cartoon-like effects with simplified content comprising well-defined regions with bold boundaries, while using a chosen color style, and retaining temporal coherence. This definition is consistent with that in [DeCarlo and Santella 2002]. In comparison, Winnemöller aims to produce soft, simplified content, achieved by smoothing, as shown in Fig. 9. Our method has the ability to perform object shape simplification. The side-by-side comparison is provided in the supplemental material. 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 20 employees to subjectively evaluate 5 video clips each. The average score awarded was 85/100, with 60/100 being considered acceptable; 97% of evaluations were at a level of 60/100 or higher.

Fig. 8 shows the effect of importance map in the segmentation. With importance map, the segmentation produces more regions but well maintains the detail of the faces, while other areas are simplified. The incoherence of importance map will also decrease the coherence of final stylized video. So if using importance map, better and faster temporal coherent saliency map is important to our method.

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. 4 and 10). When the scene changes slowly, few residual pixels remain (Fig. 10, above), 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. 10, below), 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. And image noise and motion blur may also decrease the quality of optical flow, any ultimately decrease the coherence of final results.

Fig. 11 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. To test color scheme replacement, we randomly choose 20 pictures as input, and applied our method using the 3 reference images in Figs. 5 and 11, giving 60 output pictures. These were evaluated by 10 participants, who were each shown 10 pictures randomly chosen from the above 60 results, and asked which of the three reference images had the most similar color style. A correct correspondence was suggested in 93/100 cases. The others contained many green pixels whose hue was almost unchanged in color scheme replacement, as they were far from peaks in the hue histogram.

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 of occluded regions. When such regions are gradually uncovered by an overlying object, they typically initially merge with the overlying object, then suddenly separate from it when they have grown to a sufficient size. This may cause jumps in color, although our approach in Section 5 alleviates such effects to some extent.

Figure 9: *Comparison of stylization effects. From left to right: source image, results using Winnemöller's method, results of color quantization, results using our method.*

Figure 10: *Label propagation by optical flow: two examples. From left to right: previous frame, current frame, residual map before morphological filtering, propagation map after morphological filtering.*

7 Conclusions

Our real-time video stream stylization 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 affine transformation of coherent regions, and region overlap, would allow more semantically useful vectorization, as well as enabling greater compression. Thirdly, we would like to extend this work to other styles of rendering, such as oil paintings. Our experiments so far have shown that the more structured the appearance produced by the rendering method, the more objectionable any distortions due to optical flow errors appear. Finally, we only use constant color models during the segmentation, and higher order color models (e.g. quadratic models) would provide greater scope for stylization.

References

AGARWALA, A., HERTZMANN, A., SALESIN, D. H., AND SEITZ, S. M. 2004. Keyframe-based tracking for rotoscoping and animation. In *ACM SIGGRAPH '04*, 584–591.

AGARWALA, A. 2002. Snaketoonz: a semi-automatic approach to creating cel animation from video. In *NPAR '02*, 139–ff.

BAKER, S., ROTH, S., SCHARSTEIN, D., BLACK, M., LEWIS, J., AND SZELISKI, R. 2007. A database and evaluation methodology for optical flow. In *International Conference on Computer Vision*, 1–8.

BOUSSEAU, A., NEYRET, F., THOLLOT, J., AND SALESIN, D. 2007. Video watercolorization using bidirectional texture advection. *ACM Transactions on Graphics 26*, 3, 104.

CANNY, J. 1986. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 8*, 6, 679–698.

CHANG, Y., SAITO, S., AND NAKAJIMA, M. 2003. A framework for transfer colors based on the basic color categories. *Computer Graphics International Conference 0*, 176.

CHUNG-MING WANG, Y.-H. H., AND HUANG, Y.-H. 2006. An effective algorithm for image sequence color transfer. *Mathematical and Computer Modelling 44*, 7-8, 608–627.

COHEN-OR, D., SORKINE, O., GAL, R., LEYVAND, T., AND XU, Y.-Q. 2006. Color harmonization. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH) 25*, 3, 624–630.

COLLOMOSSE, J. P., ROWNTREE, D., AND HALL, P. M. 2005. Stroke surfaces: Temporally coherent artistic animations from video. *IEEE Transactions on Visualization and Computer Graphics 11*, 5, 540–549.

COMANICIU, D., AND MEER, P. 2002. Mean shift: a robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 24*, 5, 603–619.

DECARLO, D., AND SANTELLA, A. 2002. Stylization and abstraction of photographs. In *ACM SIGGRAPH '02*, 769–776.

FISCHER, J., HALLER, M., AND THOMAS, B. 2009. Stylized Depiction in Mixed Reality. *International Journal of Virtual Reality*.

GREENFIELD, G. R., AND HOUSE, D. H. 2003. Image recoloring induced by palette color associations. *International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision 11*, 189–196.

Figure 11: *Color scheme replacement examples. Left to right, above: source image, reference image 1, color transferred from reference image 1, reference image 2, color transferred from reference image 2. Below: corresponding hue histograms in HSV space.*

HAYS, J., AND ESSA, I. 2004. Image and video based painterly animation. In *NPAR '04*, 113–120.

HERTZMANN, A., AND PERLIN, K. 2000. Painterly rendering for video and interaction. In *NPAR '00*, 7–12.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *ACM SIGGRAPH '01*, 327–340.

HERTZMANN, A. 1998. Painterly rendering with curved brush strokes of multiple sizes. In *ACM SIGGRAPH '98*, 453–460.

HERTZMANN, A. 2001. Paint by relaxation. In *CGI '01: Computer Graphics International 2001*, IEEE Computer Society, Washington, DC, USA, 47–54.

KANG, H., LEE, S., AND CHUI, C. K. 2009. Flow-based image abstraction. *IEEE Transactions on Visualization and Computer Graphics 15*, 1, 62–76.

KOVÁCS, L., AND SZIRÁNYI, T. 2002. Creating animations combining stochastic paintbrush transformation and motion detection. *International Conference on Pattern Recognition 2*, 1090–1093.

KUMAR, M. P., TORR, P. H. S., AND ZISSERMAN, A. 2005. Learning layered motion segmentation of video. In *International Conference on Computer Vision*, I: 33–40.

KYPRIANIDIS, J. E., AND D"OLLNER, J. 2008. Image abstraction by structure adaptive filtering. In *Proc. EG UK Theory and Practice of Computer Graphics*, 51–58.

LIENHART, R., AND MAYDT, J. 2002. An extended set of haar-like features for rapid object detection. vol. 1, I–900–I–903 vol.1.

LITWINOWICZ, P. C. 1997. Processing images and video for an impressionist effect. In *ACM SIGGRAPH '97*, 407–414.

PANG, W.-M., QU, Y., WONG, T.-T., COHEN-OR, D., AND HENG, P.-A. 2008. Structure-aware halftoning. *ACM Transactions on Graphics (SIGGRAPH 2008 issue) 27*, 3, 89:1–89:8.

PARIS, S. 2008. Edge-preserving smoothing and mean-shift segmentation of video streams. In *ECCV08*, II: 460–473.

REINHARD, E., ASHIKHMIN, M., GOOCH, B., AND SHIRLEY, P. 2001. Color transfer between images. *IEEE Comput. Graph. Appl. 21*, 5, 34–41.

SANTELLA, A., AND DECARLO, D. 2004. Visual interest and npr: an evaluation and manifesto. In *NPAR '04*, 71–150.

SAWANT, N., AND MITRA, N. 2008. Color harmonization for videos. In *Sixth Indian Conference on Computer Vision, Graphics & Image Processing, 2008. ICVGIP '08.*, 576–582.

VANDERHAEGHE, D., BARLA, P., THOLLOT, J., AND SILLION, F. 2007. Dynamic point distribution for stroke-based rendering. In *Rendering Techniques 2007 (Proceedings of the Eurographics Symposium on Rendering)*, 139–146.

WANG, J., XU, Y., SHUM, H.-Y., AND COHEN, M. F. 2004. Video tooning. *ACM Transactions on Graphics 23*, 3, 574–583.

WEN, F., LUAN, Q., LIANG, L., XU, Y.-Q., AND SHUM, H.-Y. 2006. Color sketch generation. In *NPAR '06*, 47–54.

WINNEMÖLLER, H., OLSEN, S. C., AND GOOCH, B. 2006. Real-time video abstraction. *ACM Transactions on Graphics 25*, 3, 1221–1226.

XIAO, J., AND SHAH, M. 2005. Motion layer extraction in the presence of occlusion using graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*, 10 (October), 1644–1659.

YANG, C.-K., AND YANG, H.-L. 2008. Realization of seurat's pointillism via non-photorealistic rendering. *The Visual Computer 24*, 5, 303–322.

ZACH, C., POCK, T., AND BISCHOF, H. 2007. A duality based approach for realtime tv-l1 optical flow. In *Pattern Recognition (Proc. DAGM)*, 214–223.

ZHAI, Y., AND SHAH, M. 2006. Visual attention detection in video sequences using spatiotemporal cues. In *ACM MULTIMEDIA '06*, 815–824.

ZHANG, S.-H., CHEN, T., ZHANG, Y.-F., HU, S.-M., AND MARTIN, R. 2009. Vectorizing cartoon animations. *IEEE Transactions on Visualization and Computer Graphics 15*, 4, 618–629.

ZITNICK, C. L., JOJIC, N., AND KANG, S. B. 2005. Consistent segmentation for optical flow estimation. In *International Conference on Computer Vision*, II: 1308–1315.