Video Painting based on a Stabilized 3D Flow Field

Jong-Chul Yoon In-Kwon Lee Henry Kang

Abstract

We present a method for generating a 3D feature flow field from a video and its application to video stylization. Our method extracts smoothly aligned 3D vectors that describe the smallest variation of colors in both spatial and temporal dimensions of the video, and thus efficiently preserves both spatial and temporal coherence in a relatively inexpensive manner. We use this flow field forms the basis of a particle-based video stylization technique which can produce feature-enhancing painting-style renderings of a video. Furthermore, we show our method is performed in real-time using the GPU based implementation.

Index Terms

non-photorealistic rendering, flow-based filtering, video abstraction, painterly rendering.

I. Introduction

Non-photorealistic rendering (NPR) generally involves abstraction and stylization of a scene: NPR could extract important visual cues that convey the essence of the scene more effectively. Certain existing NPR techniques [1], [2], [3], [4], [5], [6], [7], [8] have been specifically designed to process video, re-rendering it in artistic styles such as painting or cartoon-like abstraction. In painterly rendering, brush strokes are used as the basic primitives to describe a scene, while cartoon rendering often consists of smoothed or flattened regions.

To produce a painterly rendering of a video, painting algorithm designed for still images must be extended to cope with a sequence of frames in a coherent manner. Individual brush strokes are usually placed along some 2D feature flow field in each frame [1], [2], [4] so as to generate consistent and visually pleasing brush patterns. While such a smooth 2D feature flow field naturally suits the purpose of painting, it has also proved useful for other stroke-based rendering effects such as stippling, mosaics, engraving, pen illustration, and so on [9], [10], [11], [12]. Moreover, it can even improve the spatial coherence of smoothed regions and boundary lines in cartoon-style rendering [13], [8], [14], [15].

On the other hand, the bigger challenge in video painting is achieving temporal coherence of primitives between frames. A conventional solution is to extract a smooth motion field along the time axis [1], [2], [4], [7], and then to move the primitives along the flow. However, we argue that such separate treatment of spatial and temporal flow fields are less effective not only in terms of the computational cost but also the consistency of shapes both within and across frames. In case of cartoon-style video stylization, while keeping the primitives (smoothed regions) temporally coherent across frames would be less of a hassle [5], [6], these solutions are not applicable to a more general stroke-based video stylization problem. This motivates our work to develop a unified approach to deal with both spatial and temporal data flow in painting video.

In this paper, we introduce the notion of 3D flow field for video, and show how it helps process both spatial and temporal alignment of primitives in a unified manner. We extend a nonlinear 2D vector field stabilization technique to video and generate a 3D vector field that conforms to both spatial and temporal data flow. We first extract the gradient vector field using a 3D Sobel operator in a local spatio-temporal volume, and then stabilize the gradient directions using saliency features and color similarities. We then compute the partial differentials of the gradient vectors in the local volume, which we use to account for both spatial and temporal data flow. As an application of this 3D flow field, we introduce a new

Jong-Chul Yoon is with Kangwon National University at Dogye, Korea
In-Kwon Lee is with Yonsei Univsersity, Korea
Henry Kang is with University of Missouri, St. Louis.

particle-based video painting technique that retains the major spatial and temporal shapes of the original video cube. In terms of artistic styles, both cartoon style abstraction and painting style are obtainable, as will be shown in the experimental results.

In comparison to the previous video painting techniques, our unified flow-driven approach provides a couple of advantages: (1) *improved feature preservation and enhancement*, due to the nature of our spatio-temporal feature-preserving flow; (2) *real time performance*, as the flow field construction algorithm is computationally less involved than the typical motion estimation procedure.

The rest of this paper is organized as follows: In Section 2, we review the related work on image- and video-based stylization methods and briefly discuss their use of feature-preserving or feature-tracking flow fields. In Section 3, we present our method for extracting and stabilizing a 3D flow field from video. Section 4 describes the application of the 3D flow field to video painting, and Section 5 a strategy for accelerating the computation required by our system. In Section 6, we presents some experimental results. We conclude this paper and suggest future work in Section 6.

II. Related Work

Many of the existing techniques for image stylization employ a smoothly varying 2D vector field that preserves the dominant feature directions in the neighborhood. Such a feature flow field is especially useful in providing smooth alignment of primitives in stroke-based rendering. For example, the pen-and-ink illustration system by Salisbury et al. [9] allows the user to "paint" a direction field to ensure feature-preserving alignment of pen strokes. For oil painting style, scattered orientation interpolation has been successfully used to create such fields [1], [4] while an interactive approach [16] is also available. The digital facial engraving system by Ostromoukhov [10] relied on a user-provided direction field to align engraving lines. Hausner [11] performed distance transform from feature lines in order to align rectangular tile primitives, a technique that was later modified by Kim et al. [12] to locate circular stipple dots in a more rigorous manner. The versatility of 2D feature flow goes beyond stroke-based rendering. Kang et al. [13] suggested the use of 2D feature flow field to generate a spatially coherent set of lines from an image. Later, this idea was extended to provide improved feature preservation and enhancement in cartoon-style re-rendering of images and video [8], [14], [15].

In order to extend the image stylization techniques to video, one must address the problem of keeping temporal coherence of the graphical primitive across frames. The standard solution has been to estimate object motion by constructing an optical flow field [1], [2], [4]. While these approaches use optical flow to track individual brush strokes, Bousseau et al. [7] proposed an optical-flow-based texture advection method to enable video watercolorization. For non-stroke-based video stylization, a variety of approaches have been suggested. Agarwala [17] used curve evolution to track the boundary lines in cell animation. Later, Agarwala et al. [18] presented a rotoscoping method to track object motion and produce a highly stylistic animation. Wang et al. [5] generated temporally coherent cartoon-style video abstraction by segmenting regions from a space-time video volume. As an alternative, Winnemöller et al. [6] controlled the sharpness of region boundaries in each frame based on the variation in local color.

We note that many of these previous approaches, especially these for stroke-based rendering, independently solve the two important issues in video stylization: preserving spatial features and maintaining the coherence of temporal features. For example, in Litwinowicz's video painting system [1], the spatial flow is extracted from each frame without considering the temporal movement of strokes, while the temporal flow is extracted without considering the spatial alignment of strokes. As a result, time-consuming postprocessing of each frame is required to re-align strokes, and also to re-distribute strokes by addition and deletion. It seems inevitable that separate treatment of spatial and temporal data flow will cause mutual conflicts and thus could require significant effort to resolve them. In contrast, extracting a 3D flow field directly from the video not only saves processing time but also avoids the subsequent need to reconcile the spatial and temporal coherence of features.

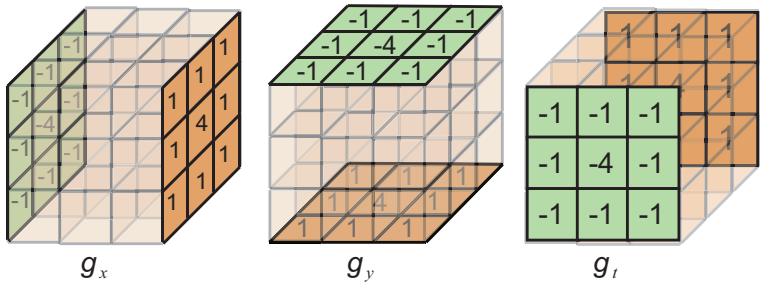


Fig. 1. 3D Sobel operators are used to calculate the 3D gradient vectors. We assign a Gaussian weight to each axis and calculate g_x, g_y and g_t.

III. 3D Flow Generation and Stabilization

We will now describe how to generate a smooth 3D flow field for a given input video $I(\mathbf{x})$, where $\mathbf{x} = (x, y, t)$ is a pixel in the space-time video volume.

A. Stabilization of the Gradient Vector Field

To generate the 3D gradient vectors $g(\mathbf{x}) = \nabla I(\mathbf{x})$, we use a 3D Sobel operator with a cubic mask. Separate operators are used to calculate $g_x(\mathbf{x})$, $g_y(\mathbf{x})$ and $g_t(\mathbf{x})$ (see Figure 1), which represent the degree of color variation along the x, y and t axes respectively. However, direct application of the Sobel operator is unstable due to video noise as well as the presence of homogeneous regions. Therefore we first stabilize the gradient vector field to address this and also to enhance the coherence of the gradient vectors.

While there are several existing ways to stabilize a vector field [19], [4], [20], [13], we build on the method by Kang et al. [13] to produce a 3D vector field that smoothly aligns with the dominant gradient directions in the neighborhood, and also to trace color similarities from frame to frame.

In regularizing $g(\mathbf{x})$, we define two types of neighborhoods: a spatial neighborhood $\Omega(\mathbf{x})$ that consists of neighboring pixels in a frame, and a temporal neighborhood $\Psi(\mathbf{x})$ that consists of pixels in a sequence of adjacent frames from $t - v$ to $t + v$, where t is the index of the current frame. Figure 2 shows the two types of neighborhood for a gradient vector $g(\mathbf{x})$. We perform nonlinear vector smoothing by applying data-driven weights to the neighboring vectors in $\Omega(\mathbf{x})$ and $\Psi(\mathbf{x})$:

$$g(\mathbf{x})^{\text{new}} = \frac{1}{K}\left\{\sum_{\mathbf{y}\in\Omega_u(\mathbf{x})} \phi(\mathbf{x},\mathbf{y})g(\mathbf{x})w_m(\mathbf{x},\mathbf{y})w_d(\mathbf{x},\mathbf{y}) + \sum_{\mathbf{y}\in\Psi_v(\mathbf{x})} \phi(\mathbf{x},\mathbf{y})g(\mathbf{x})w_c(\mathbf{x},\mathbf{y})w_m(\mathbf{x},\mathbf{y})w_d(\mathbf{x},\mathbf{y})\right\}, \quad (1)$$

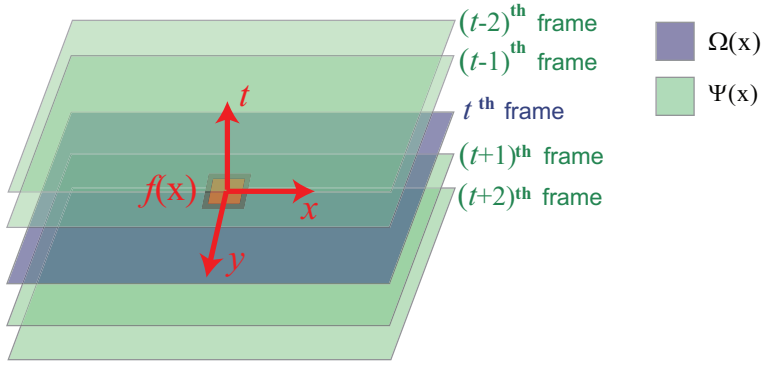


Fig. 2. Spatial and temporal neighborhoods: $\Omega(\mathbf{x})$ is a spatial neighborhood and $\Psi(\mathbf{x})$ is a temporal neighborhood.

where u and v denote the radii of the kernels $\Omega(\mathbf{x})$ and $\Psi(\mathbf{x})$ respectively, and K is a vector normalizing term. We use three weight functions w_m, w_d and w_c to assign smoothing weights adaptively according to the similarity between the gradient vectors at the center x and a neighbor y.

We first define a magnitude weight function w_m for feature preservation:

$$w_m(\mathbf{x}, \mathbf{y}) = (\hat{g}(\mathbf{y}) - \hat{g}(\mathbf{x}) + 1)/2, \tag{2}$$

where $\hat{g}(\cdot)$ denotes the normalized gradient magnitude. Note that w_m is within the range $[0, 1]$, and this weight function monotonically increases with respect to the difference in magnitude between $g(\mathbf{x})$ and $g(\mathbf{y})$. Therefore, w_m is larger when the neighboring pixels $\mathbf{y}$ has a higher gradient magnitude than the central pixel $\mathbf{x}$. This ensures the preservation of dominant gradient directions in the neighborhood.

Next, we define w_d, a directional weight function that reduces the variation of gradient vectors with similar orientations:

$$w_d(\mathbf{x}, \mathbf{y}) = |g(\mathbf{x}) \cdot g(\mathbf{y})|, \tag{3}$$

where $g(\cdot)$ is the normalized gradient vector. The value of w_d increases with the degree of alignment of the two vectors, and decreases if they become more orthogonal.

The color weight function w_c is used to smooth the gradient directions of the temporal neighborhoods:

$$w_c(\mathbf{x}, \mathbf{y}) = 1 - |\hat{I}(\mathbf{x}) - \hat{I}(\mathbf{y})|, \tag{4}$$

where $\hat{I}(\cdot)$ denotes a normalized color value in the range $[0, 1]$. This weight function increases as the color distance between the central pixel $\mathbf{x}$ and a neighboring pixel $\mathbf{y}$ become smaller. As a result, w_c reduces the variation of gradient vectors with similar color values, and improves the temporal coherence between gradient vector fields in successive frames.

Additionally, to ensure that the vectors are aligned in the same direction, we reverse the direction of $g(\mathbf{y})$ if the angle between two gradient vectors is larger than $90^{\cdot}$, using the sign function $\phi(\mathbf{x}, \mathbf{y})$:

$$\phi(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & if \quad g(\mathrm{x}) \cdot g(\mathrm{y}) > 0, \\ -1 & otherwise. \end{cases} \tag{5}$$

The whole process of vector regularization is as follows: We start by assigning unit length to the gradient vector $g^0(\mathbf{x})$ and then stabilize it iteratively using Equation (1), and update the gradient vector incrementally: $g^i(\mathbf{x}) \rightarrow g^{i+1}(\mathbf{x})$. This usually involves two or three iterations.

B. 3D Flow Field Generation

The flow vectors we are looking for are orthogonal to the gradient vectors, because they must be in the direction of minimum color variation. In 2D, such a flow field can be easily obtained by creating vectors perpendicular to the gradient vectors. In 3D, however, an infinite number of vectors are perpendicular to the tangent planes which have the gradient vectors as normals. We therefore introduce a new way to extract the flow vectors from the 3D gradient vectors.

Basically, we derive a flow vector $f(\mathbf{x})$ by taking partial derivatives of the nearby gradient vectors $g(\mathbf{x})$. As shown in Figure 3(a), these gradient vectors follow the directions of maximum color variation, and should therefore be perpendicular to most of the edges in the image. If the gradient vector field is sufficiently smooth, and each gradient vector is of unit length, then we can obtain the tangent of an edge curve by determining the difference between nearby gradient vectors (see Figure 3(b)). An approximation to the tangent vector of the edge can thus be constructed from the partial derivatives of the gradient vectors and, this tangent vector will be aligned with the direction of minimum color variation. Using this configuration, we calculate the flow vector by taking the second-order partial derivatives of the color value $I(\mathbf{x})$, as follows:

$$f(\mathbf{x}) = \left(\frac{\partial^2 I}{\partial x^2}, \frac{\partial^2 I}{\partial y^2}, \frac{\partial^2 I}{\partial t^2} \right). \tag{6}$$

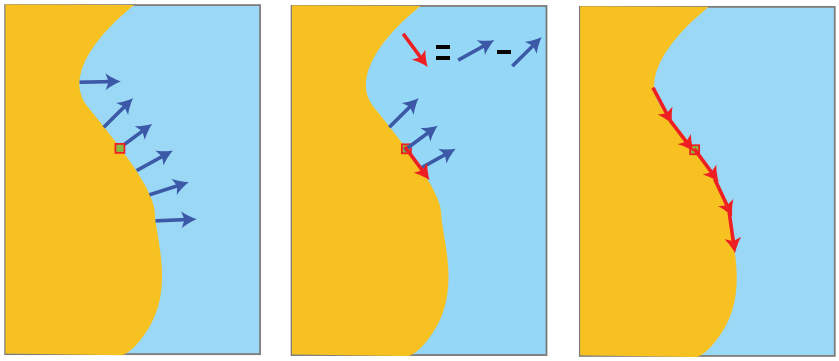


Fig. 3. Derivation of a flow field from a gradient vector field: (a) the gradient vectors on an edge points in the direction of maximum color variation; (b) a single flow vector (blue arrow), which can be obtained from the partial derivatives of nearby gradient vectors; (c) successive flow vectors.

We obtain $f(\mathbf{x})$ by applying the 3D Sobel operator to the gradient vector field. Ordinary second derivatives of gradient vectors would produce mixed terms, so we simplify the computation by obtaining the partial derivatives from the projections of the gradient vectors on to each axis.

To enhance the spatial and temporal coherence of the flow field further, we iteratively apply the stabilization filter of Equation (1), and update the flow vector incrementally: $f^i(\mathbf{x}) \rightarrow f^{i+1}(\mathbf{x})$. In practice, this requires just one or two iterations. We also make sure that the directions of the flow field are aligned with the temporal axis t by reversing the direction of $f(\mathbf{x})$ if $f_t(\mathbf{x}) < 0$. After stabilization, the flow field $f(\mathbf{x})$ describes directions which cross the boundary of the spatially dominant shapes with minimum temporal variation.

IV. Video Stylization using a Flow Field

We now apply our 3D flow field to video stylization, with a particular focus on two styles: abstraction and painting.

A. Video Abstraction by Particle Simulation

The mechanism of our video stylization technique resembles that of particle simulation. In essence, we treat each pixel $I(\mathbf{x})$ in a video as a particle and let it flow in the direction of the flow field. The colors of these particles accumulate on a pixel grid in proportion to the extent that they overlap each cell of that grid, in a similar manner to trilinear interpolation. These accumulated colors generate a smoothly colored pattern aligned with the flow directions. Because our flow field is temporally smooth, the color pattern is also coherent from frame to frame.

Since the vectors in the flow field are normalized, we need to assign a speed to each pixel particle. Lower speeds preserve details more faithfully (Figure 4(b)), and higher speeds produce more abstract results (Figure 4(c)). This leads us to formulate the following equation for flow-based particle simulation:

$$I(\mathbf{x}) \leftarrow \alpha I(\mathbf{x} + v_g v_c f(\mathbf{x})) + (1 - \alpha) I(\mathbf{x}), \tag{7}$$

where v_g and v_c are the velocity control parameters of each pixel particle, and α is a weight term which controls the rate of accumulation. The value of α, which is in the range $[0, 1]$, is determined by the volume of the region of overlap between $I(\mathbf{x})$ and $I(\mathbf{x} + v_g v_c f(\mathbf{x}))$. The degree of abstraction is controlled by v_g, which is the mean of the magnitudes of the gradients in a small region:

$$v_g = 1 - \frac{1}{W(\mathbf{x})} \sum_{\mathbf{x} \in W} ||g(\mathbf{x})||, \tag{8}$$

(a) (b) (c)

Fig. 4. The effect of particle velocity v_g on the degree of abstraction: (a) is an input image; (b) $v_g = 0.5$; (c) $v_g = 2$.

5 iterations 15 iterations 30 iterations

Fig. 5. The effect of the number of iterations on the degree of abstraction.

where $W(\mathbf{x})$ are the pixels in a spatial neighborhood around the central point $\mathbf{x}$. As a region of high gradients is likely to correspond to an edge or other important features, we assign low velocities to the pixel particles in these regions to preserve detail. If the region contains low gradients, we assign high velocities to the pixel particles to increase the smoothness of the image. We used a window W of size 8×8.

We define a further velocity v_c, based on color difference, which we use to avoid the incursion of pixel particles across an edge:

$$v_c = 1 - |\hat{I}(\mathbf{x} + v_g f(\mathbf{x})) - \hat{I}(\mathbf{x})|, \tag{9}$$

where $\hat{I}(\cdot)$ denotes a normalized color value in the range $[0, 1]$. This term prevents the destruction of edge features because the value of v_c decreases as the color difference between the original pixel and the translated pixel increases.

One advantage of this particle simulation approach is that it reduces the computational cost below that of kernel-based approaches. In addition, the use of a 3D feature flow does a good job of preserving the

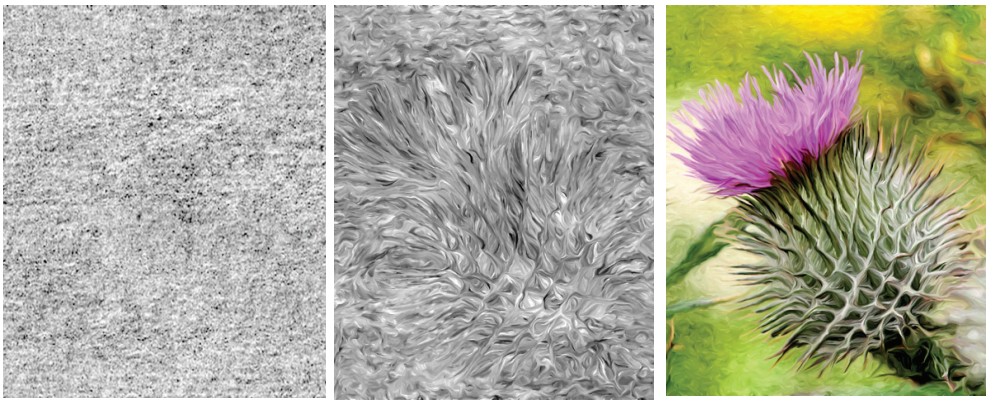


Fig. 6. Stroke texture map generation:(a) is the initial texture map $t^0(\mathbf{x})$; (b) is a texture map simulated using the flow field from the input video; (c) is a composite image by overlaying the stylized video and the stroke texture map.

spatial and temporal structure of objects in the video. Particle simulation can be applied iteratively, in a similar way to filter-based video abstraction [6], allowing control of the degree of abstraction (see Figure 5).

B. Flow-guided Stroke Texture Mapping

We extended our the flow-based particle simulation to produce a painting-style abstraction. In particular, we enhanced the directional feature of the abstraction using flow-guided stroke texture mapping. Previous research on painterly rendering has generally used stroke textures to mimic artists' methods [21], [22], [23], [4]. Extending this approach to video naturally introduces the problem of frame-to-frame coherence, which can be addressed by using the optical flow technique [4] to track the sampled points. However, since the stroke texture is relatively large, and can have a different orientation in each frame, artifacts may remain (e.g. flipping and the shower-door effect). To reduce these sorts of artifacts in video painting, Bousseau et al. [7] introduced an advection-based texture mapping technique which applies a temporally coherent stroke texture to each pixel. Since a pixel can be expected to be smaller than the texture of a stroke, this greatly reduces the number of artifacts introduced by texture translation.

We use a similar pixel-based approach to generate a painterly video. An initial stroke texture map $t^0(\mathbf{x})$ is filled with pseudo-random numbers in the range $[0, 1]$, as shown in Figure 6(a). Using Equation (7), we apply particle simulation to $t^0(\mathbf{x})$ using the flow field of the input video. We call the result of this simulation $t^i(\mathbf{x})$, where i is the number of iterations. We then update the stroke texture map $t^i(\mathbf{x})$ with the same number of iterations of the stylization process, as shown in Figure 6(b). Finally, $t^i(\mathbf{x})$ is overlaid with the stylized video producing the result in Figure 6(c).

Stroke texture mapping enhances the directional features of a video, and produces a brush-like smoothing aligned with the flow field. Since the flow field is temporally aligned with the movement of color, there are few temporal artifacts.

V. Improving Computational Efficiency

In this section, we introduce the multi-resolution approach for particle simulation and a GPU-based implementation for efficient computation.

A. Multi-resolution particle simulation

The multi-resolution approach is well-known and has been used for multi-level stylization of image and video [1], [24], [4], [25]. We have adapted the multi-resolution method to our flow-based particle simulation system.

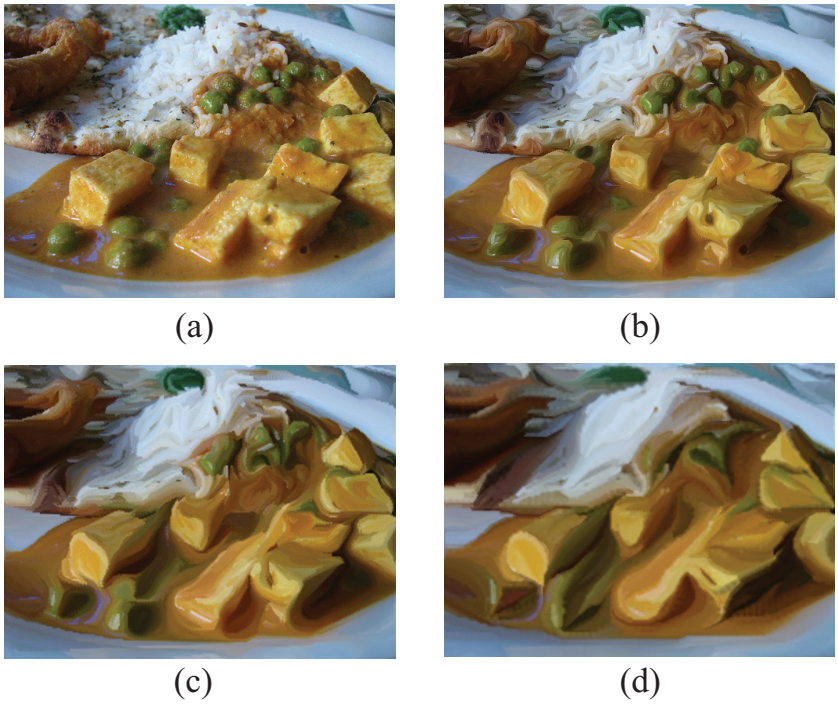


Fig. 7. The result of stylization using different video resolutions: (a) input video; (b) stylization result from I^0 - the same size as the original input video; (c) and (d) are stylization results from I^1 and I^2 respectively.

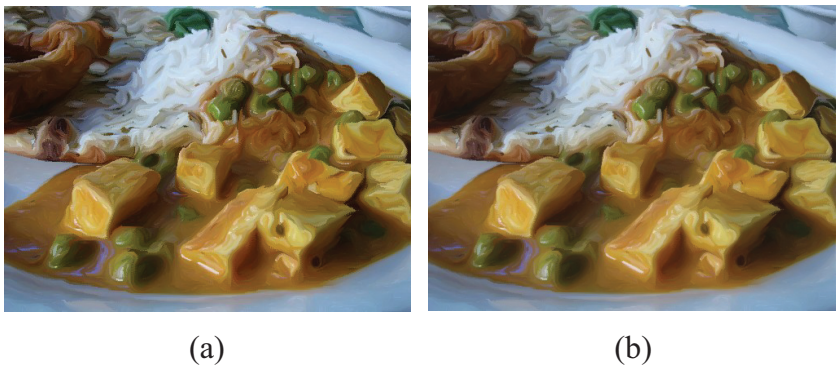


Fig. 8. Comparison between stylization results using (a) single-resolution video and (b) multi-resolution video.

First of all, we prepare low-resolution versions of an input video. We call these versions I^l, where l is a resizing parameter which reduces the scale of the input video by a factor of $1/2^l$. We first extract the flow field of each I^l and then apply flow-based stylization to each I^l separately. Figure 7 shows the result of applying the particle simulation process to I^0, I^1 and I^2 of an input video. Since, the size of a particle is determined by the size of a pixel, applying the particle simulation process to a low-resolution video produces a more abstract result (see Figure 7(c) and (d)). If we superimpose the stylized versions of each I^l with a coarse-to-fine then we obtain a stylized video with particles of different size.

To abstract a full-resolution video, we usually used twenty iterations of the particle simulation process. But we can obtain similar results from the multi-resolution approach by performing only five iterations on versions at three levels of resolution (see Figure 8). Because we used fewer iterations and stylizing the low-resolution versions requires fewer particles, this multi-resolution approach reduces the computational cost by 30%, compared with the direct approach, even though additional computation is required to calculate the flow field at each level. An additional benefit is the creation of a multi-resolution output video.

B. GPU implementation

We implemented both a GPU version of our painting technique using BSGP [26], and a CPU version using OpenCV. The test environment was a PC with an Intel Core2 Quad 2.40GHz processor with 2Gb of memory, running Windows XP, and a Geforce 9600GT.

The computation times required for CPU-based bilateral filtering and the CPU-based version of our method are shown in Table I. Performance mainly depends on the resolution of the input video and the size of the kernel used in stabilizing the flow field. In all of our tests, we performed three iterations to stabilize the gradient vector field and one iteration to stabilize the flow field. The kernel sizes for the neighborhoods $\Omega(\mathbf{x})$ and $\Psi(\mathbf{x})$ in the stabilization step were $u = 5$ and $v = 5$, respectively.

Our flow-based particle simulation approach has a complexity of $O(n)$, where n is the number of pixels, whereas filtering approaches have a complexity of $O(\mu \times n)$, where μ is the kernel radius. Our results demonstrate that our method is faster han the filtering approach. Also, since it mainly depends on the repetition of a local operation, it is easy to implement to BSGP.

Resolution	Flow stabilization	Bilateral filtering for 20 iterations ($\mu = 15$)	Particle simulation for 20 iterations
740×480	1.932 sec.	2.576 sec.	0.992 sec.

TABLE I
TIME REQUIRED TO GENERATE STYLIZED VIDEO (PER FRAME).

The computation times required for GPU-based acceleration using BSGP are shown in Table II. Implementing the multi-resolution approach on the GPU allowed stylization to run at up to 14 frames per second, suggesting that our approach will be suitable for real-time applications.

Resolution	Single-resolution approach	Multi-reolution approach with three steps
740×480	11.4 frames/sec.	14.2 frames/sec.

TABLE II
TIME REQUIRED TO GENERATE STYLIZED VIDEO USING BSGP (FRAMES PER SECOND).

VI. EXPERIMENTAL RESULTS

We tested our system with various videos which we obtained commercially [27].

Our source video, shown in Figure 9(a), has a resolution of 740×480. Figure 9(b) shows the results obtained using the flow-based particle simulation with stroke texture mapping.

Figure 10 compares the results of applying our flow-based stylization to a still image with those obtained by bilateral filtering. To apply our method to a still image, we first extracted the 2D flow field from the image using the same scheme that we use in 3D, and then applied the stylization. To extend the comparison, we prepared abstractions using bilateral filtering based on both isometric and anisotropic kernels. To generate the anisotropic kernel-based results, we used the flow-based bilateral (FBL) filter [15]. Because a general bilateral filter uses an isotropic kernel for smoothing, it does not enhance the directional features of the image very well (see Figure 10(b)). However, the FBL filter can generate abstracted results which preserve the directional features of an image in a similar way to our technique (see Figure 10(c)

and (d)). However our method is better at enhancing features both spatially and temporally. In addition, of course, it can be extended to video applications whereas the FBL filter cannot.

Figure 11 compares the results of stroke-based painterly rendering [4] and our flow-based painterly rendering method. Since the stroke texture of previous work is relatively large and it has a fixed shape, the results of stylization lose the directional details. However, our flow-based method can preserve these details, and it causes the more natural stylization. Furthermore, our flow-based method can perform in real-time, whereas stroke-based method requires dozens of seconds per frame to generate the stylized results.

Temporal stability is a key issue in video stylization, especially to avoid undesirable flickering in uniform areas. We compared the stability of our 3D flow based-approach with that of a 2D flow-based approach which applies a particle simulation to each frame, and then evaluates its temporal stability [28]. We processed 100 frames of video, taken with a static camera, and kept the stylization parameters fixed to ensure a fair comparison. Figure 12 shows how the intensity of the pixels varies. This result presented the ability of our 3D flow-based approach to produce a temporally coherent stylization.

VII. Conclusions

We have presented a method for extracting 3D flow from video and demonstrated its use in video stylization. By dealing with spatial and temporal flow together, our 3D flow field helps to preserve and enhance features within and across frames, at a low computational cost. As discussed in Section 1, the proposed 3D flow should be applicable to other types of stroke-based rendering as well, including stippling, mosaics, engraving, pen-and-ink illustration, many of which have not yet been attempted on video.

It should be noted that, although our 3D flow improves the temporal coherence of video stylization, it does not always capture the full detail of object motion. If an object moves a lot between frames, flow-based trilinear interpolation may produce a noticeable amount of motion blur. This could be undesirable in certain applications, although we find it often looks acceptable (sometimes even amusing) in a stylized video. In case a more rigorous motion estimation is desired, optical flow would be a better (albeit more expensive) choice.

References

[1] P. Litwinowicz, "Processing images and video for an impressionist effect," in *Proceedings of ACM SIGGRAPH '97*, 1997, pp. 407–414.

[2] A. Hertzmann and K. Perlin, "Painterly rendering for video and interaction," in *Proceedings of ACM Symposium on Non-photorealistic Animation and Rendering*, 2000, pp. 7–12.

[3] A. W. Klein, P.-P. Sloan, A. Finkelstein, and M. F. Cohen, "Stylized video cubes," in *Proc. Symposium on Computer Animation*, 2002, pp. 15–22.

[4] J. Hays and I. Essa, "Image and video based painterly animation," in *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, 2004, pp. 113–120.

[5] J. Wang, Y. Xu, H.-Y. Shum, and M. F. Cohen, "Video tooning," in *Proceedings of SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, 2004, pp. 574–583.

[6] H. Winnemöller, S. C. Olsen, and B. Gooch, "Real-time video abstraction," in *Proceedings of ACM SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, 2006, pp. 1221–1226.

[7] A. Bousseau, F. Neyret, J. Thollot, and D. Salesin, "Video watercolorization using bidirectional texture advection," in *Proceedings of ACM SIGGRAPH '07: ACM SIGGRAPH 2007 Papers*, 2007, p. 104.

[8] H. Zhao, X. Jin, J. Shen, X. Mao, and J. Feng, "Flow-based image abstraction," *The Visual Computer*, vol. 24, no. 7, pp. 727–734, 2008.

[9] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin, "Orientable textures for image-based pen-and-ink illustration," in *Proceedings of ACM SIGGRAPH '97*, 1997, pp. 401–406.

[10] V. Ostromoukhov, "Digital facial engraving," in *Proc. ACM SIGGRAPH*, 1999, pp. 417–424.

[11] A. Hausner, "Simulating decorative mosaic," in *Proc. ACM SIGGRAPH*, 2001, pp. 573–578.

[12] D. Kim, M. Son, Y. Lee, H. Kang, and S. Lee, "Feature-guided image stippling," *Computer Graphics Forum*, vol. 27, no. 4, pp. 1209–1216, 2008.

[13] H. Kang, S. Lee, and C. Chui, "Coherent line drawing," in *Proceedings of ACM Symposium on Non-photorealistic Animation and Rendering*, 2007, pp. 43–50.

[14] J. E. Kyprianidis and J. Dollner, "Image abstraction by structure adaptive filtering," in *Proc. EG UK Theory and Practice of Computer Graphics*, 2008, pp. 51–58. [Online]. Available: http://www.kyprianidis.com/go/tpcg-2008

[15] H. Kang, S. Lee, and C. K. Chui, "Flow-based image abstraction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 1, pp. 62–76, 2009.

Fig. 9. Video stylization: (a) is an input video; (b) is the result of a painterly video after flow-guided stroke texture mapping.

[16] S. C. Olsen, B. A. Maxwell, and B. Gooch, "Interactive vector fields for painterly rendering," in *GI '05: Proceedings of the 2005 conference on Graphics interface*. School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society, 2005, pp. 241–247.

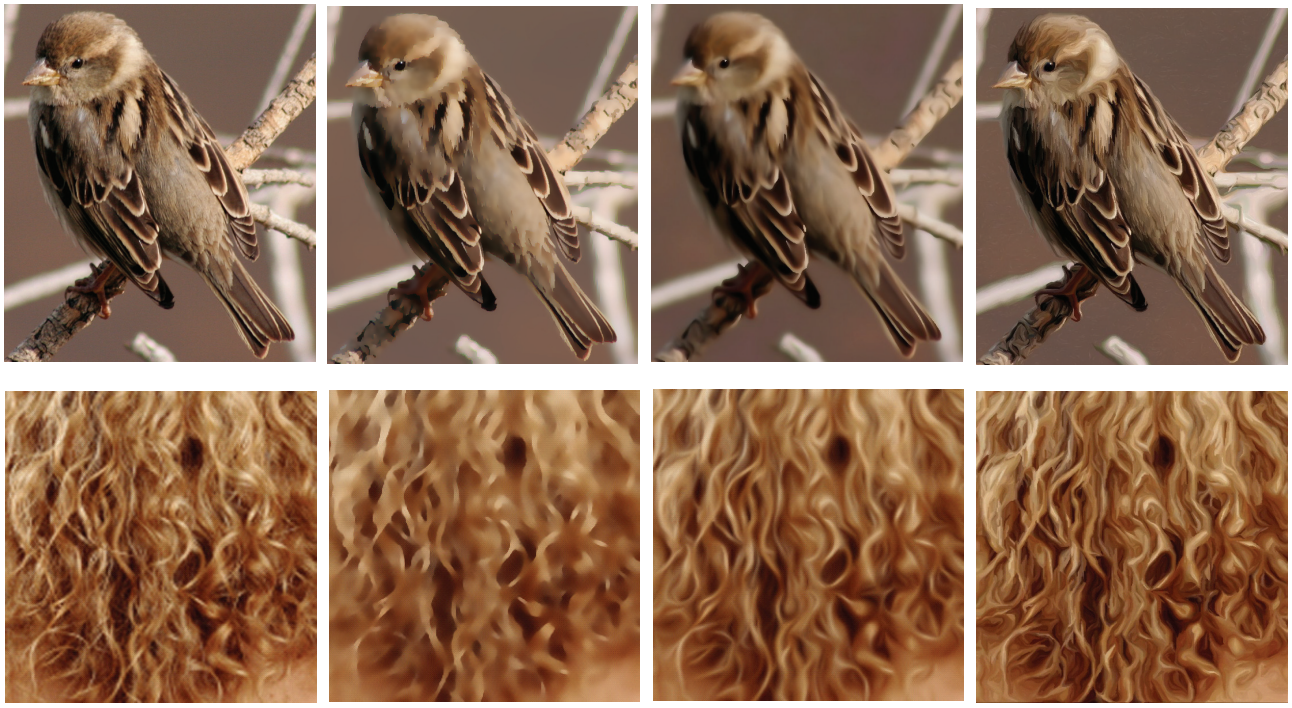
[17] A. Agarwala, "Snaketoonz : A semi-automatic approach to creating cel animation from video," in *Proceedings of Non-Photorealistic Animation and Rendering*, 2002, pp. 139–146.

[18] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz, "Keyframe-based tracking for rotoscoping and animation," in *Proceedings of ACM SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*. ACM Press, 2004, pp. 584–591.

[19] C. Xu and J. L. Prince, "Snakes, shapes, and gradient vector flow," *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 359–369, 1998.

[20] D. Tschumperlé, "Curvature-preserving regularization of multi-valued images using pde's," in *Proceedings of ECCV*, 2006, pp. 295–307.

[21] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," in *Proceedings of ACM SIGGRAPH '98*, 1998, pp. 453–460.



(a) (b) (c) (d)

Fig. 10. Comparison between the results of bilateral filtering and our flow-based method: (a) is an input image; (b) is an abstraction obtained by is isotropic bilateral filtering; (c) is an abstraction obtained by flow-based bilateral filtering [15]; (d) is an abstraction obtained using our method.

[22] ——, "Paint by relaxation," in *CGI '01: Computer Graphics International 2001*, 2001, pp. 47–54.
[23] B. Gooch, G. Coombe, and P. Shirley, "Artistic vision: painterly rendering using computer vision techniques," in *NPAR '02: Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering*, 2002, pp. 83–90.
[24] D. DeCarlo and A. Santella, "Stylization and abstraction of photographs," in *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 2002, pp. 769–776.
[25] M. Grundland, C. Gibbs, and N. A. Dodgson, "Stylized multiresolution image representation," *Journal of Electronic Imaging*, vol. 17, no. 1, pp. 1–17, 2008.
[26] Q. Hou, K. Zhou, and B. Guo, "Bsgp: bulk-synchronous gpu programming," in *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, 2008, pp. 1–12.
[27] Stock, "Freestockfootage.com," 2009, http://www.freestockfootage.com/.
[28] S. Paris, "Edge-preserving smoothing and mean-shift segmentation of video streams," in *ECCV '08: Proceedings of the 10th European Conference on Computer Vision*, 2008, pp. 460–473.

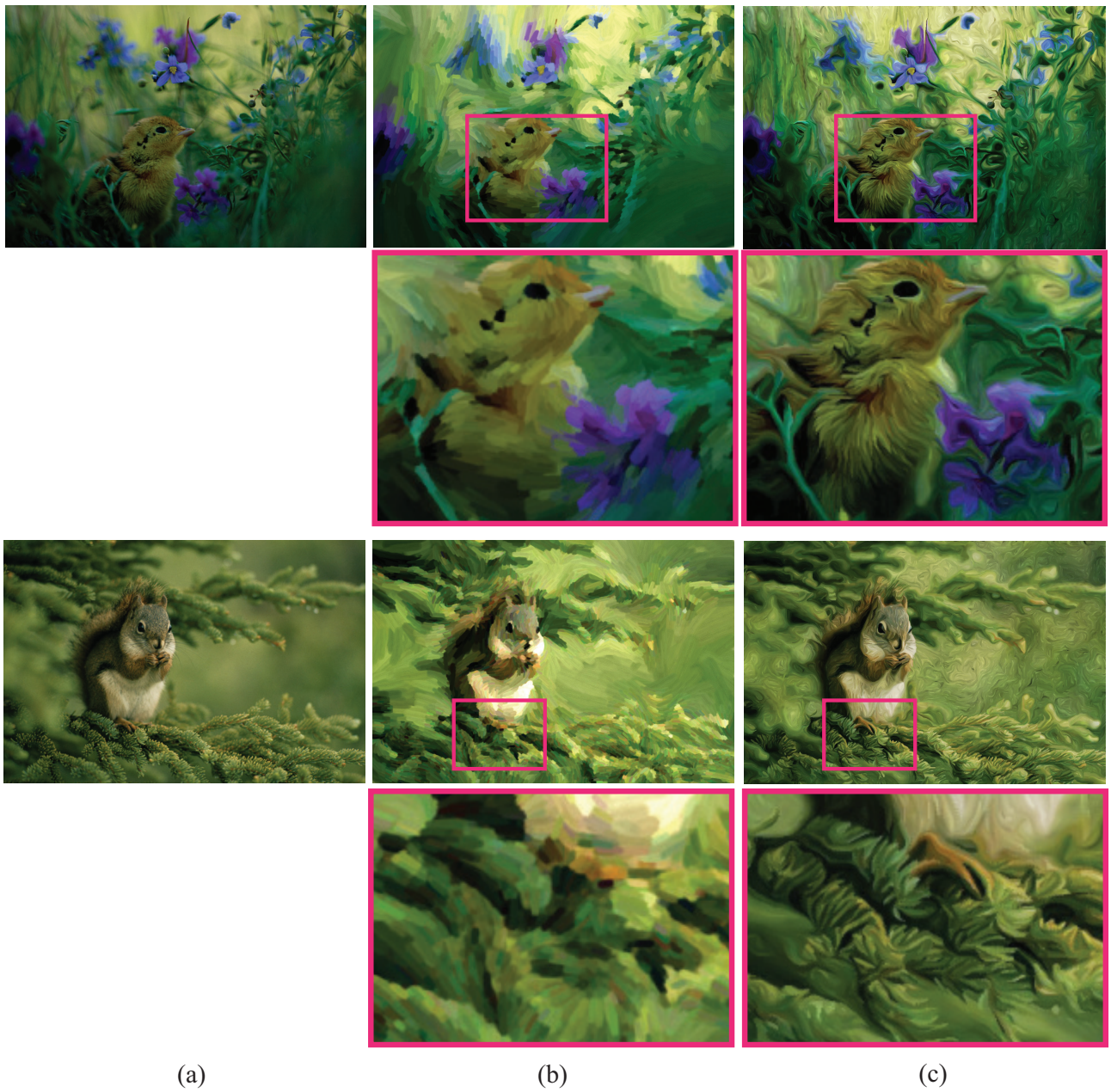


Fig. 11. Comparison between the stroke-based painterly rendering and our results: (a) stroke-based painterly rendering result from [4]; (b) is a painterly rendering created from a flow-guided stroke texture map.

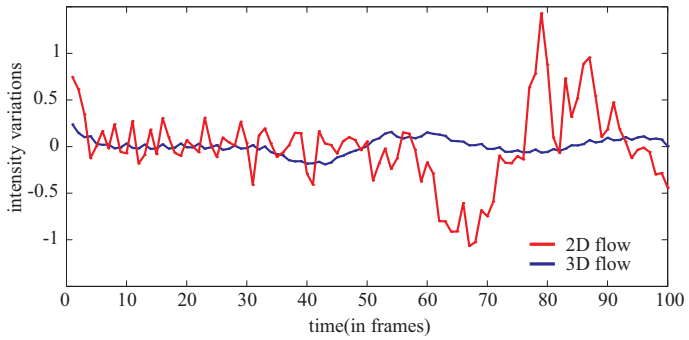


Fig. 12. Comparison of temporal stability between 3D and 2D flow- based approaches: temporal coherency is better conserved by the 3D flow field, since those are fewer intensity variations in the stylized video produced using the 3D flow field.