Flow-based Video Synthesis and Editing

Kiran S. Bhat Steven M. Seitz* Jessica K. Hodgins Pradeep K. Khosla

Carnegie Mellon University *University of Washington

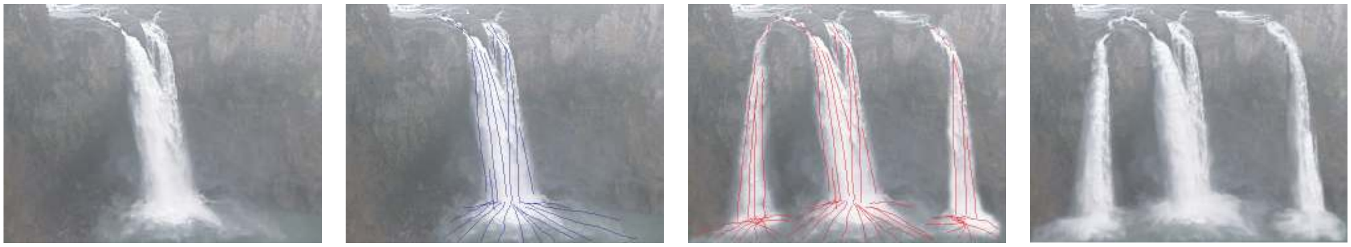

Figure 1: Synthesizing new video by manipulating flow lines. Images from left to right: one frame of input video, flow lines marked by the user, flow lines marked on the edited video and one frame of the edited video.

Abstract

This paper presents a novel algorithm for synthesizing and editing video of natural phenomena that exhibit continuous flow patterns. The algorithm analyzes the motion of textured particles in the input video along user-specified flow lines, and synthesizes seamless video of arbitrary length by enforcing temporal continuity along a second set of user-specified flow lines. The algorithm is simple to implement and use. We used this technique to edit video of waterfalls, rivers, flames, and smoke.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Motion;

Keywords: Texture and Video Synthesis, Particle Systems, Image and Video Processing

1 Introduction

Real footage of natural phenomena has a complexity and beauty that is rarely matched in synthetic footage in spite of many recent advances in simulation, rendering, and post-processing. Leveraging real footage for special effects is difficult, however, because a natural scene may not match the director's intentions and modifying the physical setting may be expensive or impossible. In this paper, we present a technique that allows the user to modify real footage while maintaining its natural appearance and complexity.

The key to our approach is the observation that video of a class of natural phenomena can be approximated by continuous motion of particles along well-defined flow lines. First, we capture the dynamics and texture variation of the particles along user-defined flow lines in the input video. To generate video of arbitrary length, we synthesize particles such that they complete their full paths along each flow line. Playing back these particles along new flow lines allows us to make interesting edits to the original video (Figure 1). The user defines flow lines on both the input and output video and we leverage his or her visual intuition to create a wide range of edits. We demonstrate the power of this approach by modifying scenes of waterfalls, a river, flames, and smoke.

2 Related Work

Creating realistic animations of fluid flow is an active area of research in computer graphics. Physically based simulation techniques have been successfully applied to simulate and control fluids (e.g., [Treuille et al. 2003]). However, these techniques are computationally expensive and are usually tailored for a single type of natural phenomena such as smoke, water, or fire.

Recently, several researchers have attempted to model the textured motion of fluidic phenomena in video and synthesize new (and typically longer) image sequences. Non-parametric models for texture synthesis have been applied to create 3D temporal textures of fluid-like motion (e.g., [Wei and Levoy 2000]). The video textures algorithm creates long videos from short clips by concatenating appropriately chosen subsequences [Schödl et al. 2000]. Video sprites extend video textures to allow for high level control over moving objects in video [Schödl and Essa 2002]. Wang and Zhu [2002] model the motion of texture particles in video using a second order Markov chain. Doretto et al. [2003] use Auto-Regressive filters to model and edit the complex motion of fluids in video. The graph cuts algorithm combines volumes of pixels along minimum error seams to create new sequences that are longer than the original video [Kwatra et al. 2003].

Our synthesis approach is very simple and produces comparable results to the best of these on sequences with continuous flow. Additionally, our technique allows an artist to specify edits intuitively by sketching input and desired flow lines on top of an image.

3 Approach

Some natural phenomena such as waterfalls and streams have time-varying appearance but roughly *stationary* temporal dynamics [Doretto et al. 2003]. For example, the velocity at a single fixed point on a waterfall is roughly constant over time. Consequently, these phenomena can be described in terms of particles moving

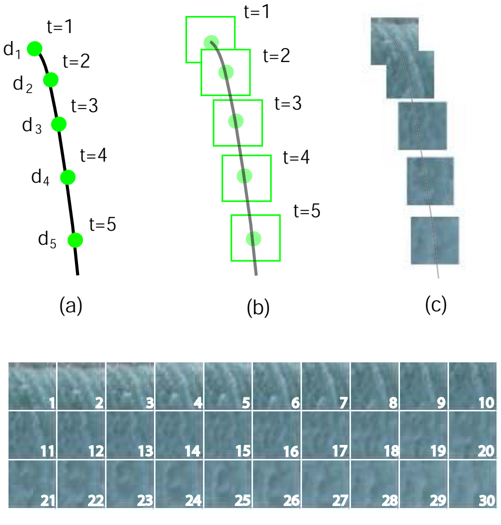

Figure 2: A particle-based representation for video. (a) A particle moving along its flow line. (b) Particle texture moving along the same flow line over time. (c) Texture variation of a *real* particle from the Niagara sequence (Figure 6). For clarity, we show the particle texture every 6th frame as it moves along the flow line. The particle velocity increases as the particle moves downward as would be expected due to gravity. (d) A filmstrip (left-right, top-bottom) showing the particle texture for each frame as the particle travels downward along the flow line. The texture of two adjacent cells is similar, which facilitates tracking. However, the texture varies significantly between the beginning and end of the flow line.

along fixed flow lines (possibly curved) that particles follow from the point at which they enter the image to the point where they leave or become invisible. For instance, the flow lines in the waterfall in Figure 1 are mostly vertical. Each particle also has an associated texture (a patch of pixels), which changes as the particle moves along the flow line. Our video texture synthesis technique produces seamless, infinite sequences by modelling the motion and texture of particles along user-specified flow lines. We first describe the way in which the particles move in video, and then describe how they are rendered using texture patches.

Particle Dynamics: To begin, consider the case of a single flow line in the image, as shown in Figure 2(a). Any particle that begins at the start of the flow line d_1 will pass through a series of positions $d_1, d_2, \ldots, d_n$ during its trajectory. The particle's velocity along the flow line may be time-varying; thus the positions d_i need not be evenly spaced. The particle's texture may vary as it moves along the flow line (Figure 2(b,c,d)).

We represent the temporal evolution of particles along this flow line as follows. Define a matrix $M(d,t) = (p,f)$, where p refers to a specific particle, and f specifies the frame in the input sequence where that particle appears at d. Figure 3(a) plots $M(d,t)$ for the input sequence, where the number in each cell corresponds to f and the color to p. The first column of this matrix shows the particles and their positions on the flow line in frame 1 of the input (hence these entries have $f = 1$). The red numbers, for example, show the path of a single particle during the course of the input sequence.

When the sequence is looped, there is a discontinuity (vertical black line) between frames 5 and 6 because particles abruptly move to different locations or disappear altogether. This discontinuity appears simultaneously for every pixel in the image making it a very noticeable artifact.

We can reduce this discontinuity by a simple change to the entries of $M(d,t)$. The modified matrix is obtained by repeating the first K diagonals of the original matrix. For example, the shaded entries of Figure 3(a) are repeated to produced Figure 3(b). While the modified matrix is composed of a subset of the same entries as in (a), it enforces temporal continuity because each particle completes its full path along the flow line. The vertical discontinuity in Figure 3(a) is replaced with a ladder-shaped discontinuity pattern in Figure 3(b), with one spatial discontinuity in each column, corresponding to an abutting pair of patches in the output sequence that were not adjacent in the input sequence. These spatial discontinuities are difficult to detect, however, because the abutting pair of patches move together in the output sequence. This simple procedure lets us create a matrix of arbitrary width that preserves temporal continuity for all particles along the flow lines.

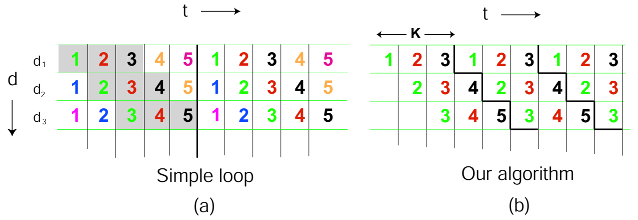

Figure 3: (a) Plot of $M(d,t)$ showing the particles on a single flow line over time in the input sequence. Numbers specify frames, and colors specify particles. Note that there is a discontinuity for each particle along the flow line between time $t = 5$ and $t = 6$ when the input sequence is looped. (b) In contrast, our synthesis algorithm maintains temporal continuity along flow lines. Although there is a discontinuity along the diagonal stepped line, it is less noticeable because all particles complete their paths along the flow line.

We implement the procedure in Figure 3(b) by sequentially generating particles at d_1 from a subset of K input particles (shaded in Figure 3(a)), and moving them along the flow line over time. First, an artist sketches a dense set of flow lines over the input video. Then, we compute and store the particle velocities and textures along these flow lines. The particle velocities are computed as follows. For a given particle location in the current frame, we search *along the flow line* for a corresponding location in the next frame that best matches the texture of the current particle.[1] The rectangular patch of pixels around the *best match* location becomes the particle texture for the next frame. All the synthesis parameters, including the number and spacing of flow lines and the size of the particle texture (Figure 2(c,d)) are controlled by the artist.

Particle Rendering: Once the particle positions along the flow lines in each new image are determined, the system blends their textures to produce the rendered result. Recall that we store the particle texture as it evolves along the flow line in the input sequence. Rendering simply involves drawing the patch of texture around each particle, and using feathering to blend the overlapping regions from different patches. The feathering method assigns a weight to each pixel in the overlapping region that is inversely proportional to its distance to the edges of the overlapping region.

Generality: Our approach works best for input sequences that have stationary temporal dynamics. Waterfalls and rivers are examples of continuous phenomena where the velocity (magnitude and direction) at any fixed point in the image is roughly constant over time. Phenomena like flames rising upward or smoke from a chimney (without wind) can also be modelled with a simple extension to our method for defining particle dynamics. For such input se-

[1] This procedure performs a constrained form of optical flow along the flow line, e.g., [Trucco and Verri 1998].

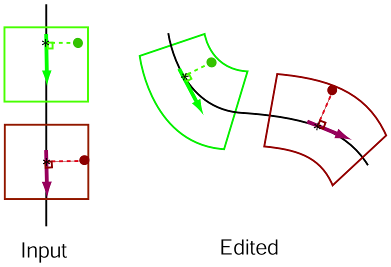

Figure 4: Schematic showing the transformations used to transfer texture from an input flow line to an edited flow line. For example, the green and red pixels in the edited flow line are obtained from the input particle texture using the tangent information at corresponding points.

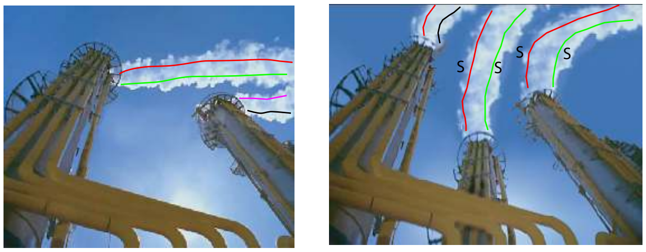

Figure 5: Result of editing a smoke sequence by manipulating flow lines. Corresponding flow lines have the same color in this picture. The flow lines from the second and third chimney in the edited sequence shown with letter S are scaled because the edited flow lines are of different length than the input.

quences, we allow the velocity magnitude at each fixed point on a flow line to vary over time. Unlike Figure 2(a), particles that begin at the start of the flow line d_1 at different times t will pass through a *different* series of positions $d_1(t), d_2(t+1), \ldots, d_n(t+n-1)$ along the *same* trajectory. This modified procedure uses the steps for dynamics and rendering described in the previous paragraphs, however, particles with different frame indices f may now have different velocity magnitudes along the flow line. Although this extension is an approximation to input video with non-stationary dynamics, it seems to work well in practice for video of flames and smoke where the overall direction of flow remains almost constant.

4 Editing

The synthesis algorithm described in the previous section can be extended to support editing by synthesizing texture over a new set of flow lines. We warp the texture along the input flow line to synthesize texture over the edited flow line, using the tangent information between corresponding points of the two curves (Figure 4). A simple scheme for assigning corresponding points between the two flow lines is to choose points of equal arclengths (from the start of the flow line) along the two curves. An alternative is to add a term to the warping function that scales the arclengths of the input to be equal to the edited flow lines. This modification produces edited textures that are scaled and warped versions of the texture from the input flow line. The animator chooses between these two schemes to create different effects in the edited sequence (Figure 5).

In our editing framework, the animator first draws a dense set of input flow lines. Then, she draws a *sparse* set of flow lines corresponding to the desired edited sequence, and specifies the correspondence between the input and edited flow lines. To create a dense set of edited flow lines, the system interpolates the flow lines spatially, as shown in Figure 6. We use dynamic programming to find a globally optimal set of correspondences between the interpolated flow lines and the input flow lines. Let Y be the dense set of input flow lines and X be the set of edited flow lines. For every pair $(x-1, x)$ of neighboring edited splines, we compute the *patch overlap error* $E(x-1, y', x, y)$ for all possible pairs (y', y) of input splines (Figure 7). We evaluate the overlap error term by synthesizing the flow line $x-1$ using texture from y' and the neighboring flow line x using texture from y. The error is defined as the sum-squared error in color in the overlapping regions of the two textures, averaged over a fixed number of frames (30 in our experiments). To compute the correspondences, we minimize the following cost function:

$$C(x,y) = \min_{y'}[C(x-1,y') + E(x-1,y',x,y)] \qquad (1)$$

where $x \in X$, $y \in Y$ and $y' \in Y$. Although the dynamic programming approach is computationally intensive, it produces a set of interpolated flow lines whose colors match well.

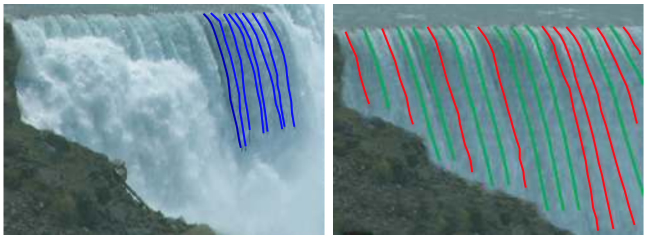

Figure 6: The left figure shows an input waterfall sequence and the right shows an edited waterfall created using our algorithm. The animator draws the flow lines in the input image, shown in blue. She then draws a sparse set of flow lines in the edited image in red and specifies the correspondences between the edited and input flow lines. The system generates a set of interpolated flow lines (shown in green) and computes their correspondences with the input flow lines automatically using dynamic programming.

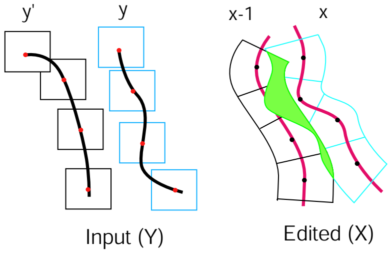

Figure 7: For a set of neighboring edited flow lines $(x-1, x)$ and their corresponding input flow lines (y', y), the patch overlap error computes a least square difference between the colors of overlapping regions, shown in green.

5 Experimental Results

We applied our algorithm to footage of waterfalls, a river, smoke, and flames. For most of these examples, we create a long temporal sequence from a short (2 second) clip of video and show a number of interesting edits (see accompanying video). We used texture patches of 24×24 pixels for all sequences except flame, where the patch size was 100×100 pixels. All the sequences took a few seconds per frame to render in Matlab, for frame sizes of 320×240 pixels.

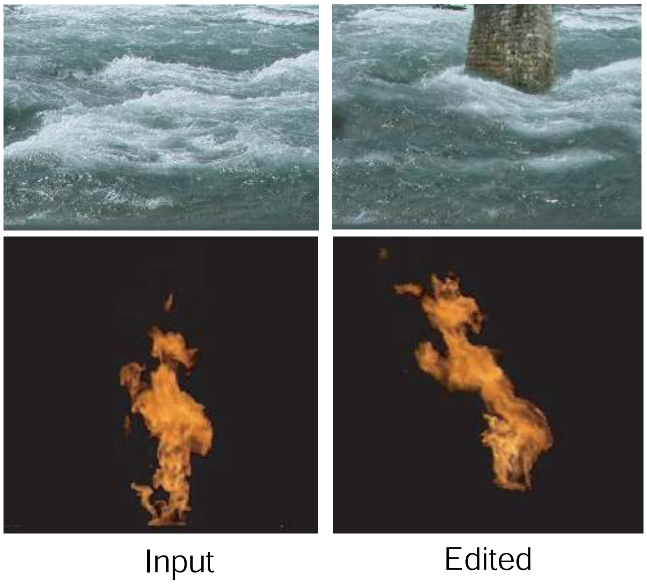

Figure 8: Editing results for a river and a flame, where a trunk is inserted into a river and a flame is bent to the left.

Niagara Falls: This sequence has several distinct texture regions, with clear water on the top, foamy water on the left and spray on the bottom (Figure 6). As shown in Figure 2(d), the texture changes dramatically along flow lines. Because our algorithm does not model transparency, we observe some visual artifacts at the interface between water and spray in the synthesized infinite sequences. We edited the Niagara sequence to change the landscape and remove the foamy portion of the waterfall. The resulting edited sequence and the corresponding flow lines are shown in Figure 6.

Waterfall: In this example, we added two extra channels to a waterfall sequence (Figure 1). We mark a set of vertical flow lines to capture the water moving down and another set of radial flow lines to model the collisions at the bottom of the fall. The interface between the waterfall and the lake at the bottom causes interesting foam patterns, which are captured in the temporally extended and edited sequences.

Stream: Here, we edit a stream sequence to simulate the effect of water colliding against a trunk (Figure 8). We edit the water flow lines to curve around the trunk and use the foamy texture (where water collides with rocks in the input video) to create the effect of water colliding against the trunk.

Smoke: This example shows an edit of smoke from a chimney that exhibits non-stationary motion along constant flow lines (Figure 5). We add an extra chimney and change the direction of the smoke.

Flame: We model the input flame video using particles moving on a single vertical flow line. Figure 8 shows one frame of an edited sequence, where we simulate the effect of wind blowing to the left by specifying an edited flow line that curves left.

6 Discussion

This paper describes a simple algorithm for creating and editing arbitrarily long videos from short input clips for a variety of natural phenomena. We applied this technique on image sequences of waterfalls, rivers, flames, and smoke. For these phenomena, our synthesis results are comparable to the best existing video texture synthesis techniques. Our system also provides an intuitive interface for editing video. Most of the editing examples took a few iterations of interaction (refining the edited flow lines) to achieve the desired result.

Although our approach has been successfully used to edit a number of sequences, it is limited to input sequences with nearly stationary flow patterns. Better results would likely be obtained by using graph cuts instead of feathering to blend overlapping particle texture patches, possibly enabling application to more structured scenes such as cars moving on a highway. Extending the algorithm to handle transparency is an important and interesting avenue for future work. In this work, we leveraged the artist's intuition about the flow of particles in the scene to specify flow lines in the input and desired sequences. Devising robust techniques to compute good flow lines automatically would be a useful improvement, as would devising ways to animate the flow lines temporally to create a broader set of interesting edits. We do not desire, however, to create a fully automatic technique but instead wish to create a technique that leverages the artist's visual judgment by allowing her the greatest control over the resulting footage.

References

DORETTO, G., CHIUSO, A., SOATTO, S., AND WU, Y. 2003. Dynamic textures. *International Journal of Computer Vision 51*, 2, 91–109.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003, 22*, 3, 277–286.

SCHÖDL, A., AND ESSA, I. A. 2002. Controlled animation of video sprites. In *ACM SIGGRAPH Symposium on Computer Animation*, 121–128.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. In *Proceedings of ACM SIGGRAPH 2000*, 489–498.

TREUILLE, A., MCNAMARA, A., POPOVIĆ, Z., AND STAM, J. 2003. Keyframe control of smoke simulations. *ACM Transactions on Graphics, SIGGRAPH 2003, 22*, 3, 716–723.

TRUCCO, E., AND VERRI, A. 1998. *Introductory Techniques for 3-D Computer Vision*. Prentice-Hall, Inc, New Jersey, ch. 7, 146–148.

WANG, Y., AND ZHU, S.-C. 2002. A generative model for textured motion: Analysis and synthesis. In *Proc. European Conference on Computer Vision (ECCV)*, 582–598.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of ACM SIGGRAPH 2000*, 479–488.