# On Animating 2D Velocity Fields

David Kao[1] and Alex Pang[2]

[1] NASA Ames Research Center
[2] Computer Science Department, UCSC
davidkao@nas.nasa.gov, pang@cse.ucsc.edu
www.cse.ucsc.edu/research/avis/texflow.html

**Abstract.** A velocity field, even one that represents a steady state flow, implies a dynamical system. Animated velocity fields is an important tool in understanding such complex phenomena. This paper looks at a number of techniques that animate velocity fields and propose two new alternatives. These are texture advention and streamline cycling. The common theme among these techniques is the use of advection on some texture to generate a realistic animation of the velocity field. Texture synthesis and selection for these methods are presented. Strengths and weaknesses of the techniques are also discussed in conjunctions with several examples.

**Key Words and Phrases:** texture advection, streamlines, droplets, color table animation, LIC, texture mapping, flow field.

## 1 INTRODUCTION

Animation best captures the dynamical properties of velocity fields. We are reminded once again in the recent papers of Stam [7] and Witting [11], where both papers use animation to show the complex nature of a flow field. In these papers, the authors effectively used texture advection as the primary mechanism to depict the results of their flow simulation thereby achieving effects such as swirling and mixing of fluids. While these papers may have met their goals for aesthetic visual effects, the basic technique requires modifications to make it suitable for scientific visualization purposes. Specifically, how does one handle texture advection beyond the bounds of the flow field? Likewise, how does one handle texture advection for critical points such as sources and sinks, etc. in the flow field?

This paper reviews several existing techniques for animating velocity field from the scientific visualization community (Section 2). We then propose two alternative methods (Sections 3 and 4) that draw upon the strengths of existing methods to provide realistic depictions of the behavior of the velocity field.

## 2 RELATED WORK

There are several excellent work on the use of animation to aid in the understanding of flow fields. We review a number of approaches here.

Hin and Post [3] used particle motion animation to depict a turbulent flow field. The flow field is decomposed into a convective and a turbulent component. During each step of the animation, the path of a particle is stochastically perturbed to produce a realistic turbulent flow effect.

Several animation techniques use either color table animation or cycling of texture maps. Van Gelder and Wilhelms [2] used color table animation to animate streamlines in a 3D flow field. Streamlines are rendered opaquely using z-buffer while colored dots "move" along each streamline creating the illusion of motion. Yamrom and Martin [12] used a variation of this approach by first rendering a 3D flow field as hedgehogs. Then a series of 16 1D cycling texture pattern of varying alpha values is used to animate the hedgehogs. No particle integration step is required in their method. Jobard and Lefer [4] introduced the idea of a motion map that stores the path and velocity magnitude of streamlines in a steady flow field. For animation, they create a sequence of color table indices for each streamline. Animation is carried out either by the traditional color table animation or by cycling of the textures, where the color indexes are shifted. For textures, high frequency random textures are generated and used on streamlines producing effects similar to animated line integral convolution (LIC) techniques.

LIC is another popular method for visualizing flow fields. One of the enhancements is animated LIC to better present the dynamic nature of flow fields. Again, a number of work in this area is summarized here. Wegenkittl et al. [10] described oriented LIC (OLIC), a sparser LIC version where orientation and direction of the flow field are encoded with a ramp like convolution kernel. Two different approaches for animating OLIC were suggested: (a) phase shift of convolution kernel in consecutive frames and (b) color table animation.

Another approach with similar effect is with animated spot noise [8] where spots are placed randomly in a 2D flow field and assigned a random phase. During animation, they appear to glow, move a short distance, then fade. In order to handle variable speed animation, Forssell and Cohen [1] proposed that a different convolution kernel is used for each pixel. Specifically, the convolution kernel has a phase shift proportional to the corresponding grid cell's physical velocity magnitude. Shen and Kao [6] extended animated LIC to Unsteady Flow LIC (UFLIC), a time-accurate method capturing unsteady flow fields. Another variation is with Pseudo-LIC (PLIC) [9] where LIC-like images are generated using textured streamlines. Variable speed animation is achieved by varying the cycling frequency of textures.

The predominant idea of animating flow fields seem to be either with color table animation or texture animation. In fact, one of the earlier work is by Max et al. [5] where 3D textures coordinates were advected in a climate simulation model. Two 3D textures were linearly combined over time and then passed through a transparency filter so that texture distortions and overlapping textures were minimized.

The work presented in this paper extends animation of flow fields in two ways. First, we examine the texture advection approach used by Max and modify it to better support boundary conditions and critical points in the flow. Second, we look at the texture cycling along a streamline and show how a continuous animation of a steady flow can be achieved with a finite number of integration steps.

## 3 TEXTURE ADVECTION

The idea behind this approach is to modify the texture image such that it appears to get distorted by some underlying flow field. This is best carried out by advecting the texture coordinates of the image. That is, each texel is treated as a particle and integrated backwards in time for some number of integration steps L. Backward integration is used instead of forward integration. For each texel, we calculate which texel(s) along a streamline contributed to it. As time progresses, texels contributing to a particular texel would have come from a farther distance (integration step) away. By incrementally increasing L and saving the output image at each time step, we can generate an animated flow field showing how the input texture image is distorted by the flow field.

### 3.1 Boundary Conditions

Although the method above seems easy and straightforward, a couple of issues arises when the integration reaches the boundary or a critical point. We will discuss the boundary case situation first. Since the particle is advected backward in time, depending on the original seed location and the flow direction, it is most likely that after some time period, a particle will reach the grid boundary. This prompts the question: what texel value do we assign when a particle reaches a boundary? A simple and naive solution would be to terminate the particle at the boundary and use the color of the texel at the boundary where the particle terminated. We refer to this approach as the constant color boundary method. The outcome of this strategy will be that after some time frames, the boundary color is propagated into the image. Since the boundary color is constant, we will see a corresponding constant color streaks coming from the boundaries of the image (See Figure 1, particularly the regions near the upper left and lower right). Eventually, the image will consist of many constant texture color regions. While this may be undesirable for most situations, it does have some visualization value. Specifically, it identifies regions in the flow that are not affected by the boundaries at all and the regions of flows that become time-invariant. To avoid "freezing" the image with constant texture regions, these boundary texels must somehow be changed or made dependent on integration time.

We now consider the periodic boundary condition. If the input texture is periodic, then we displace the particle when it reaches the boundary i.e. if $x < 0$ then $x+ = max\_x$ and if $x > max\_x$ then $x\% = max\_x$. This check is also performed for the $y$ coordinate. Assume that the input texture is periodic, imposing periodic boundary might, at first glance, appear to solve the problem. However, this alone is not sufficient. The problem really is that once a particle goes outside the boundary of the flow, there's no information available for carrying out the backward integration. Hence, it would always be drawing from the same texel value. Figure 2 shows the effects of imposing a periodic boundary condition. To combat this problem, the following approach is used. A procedurally defined 3D texture is used such that when a particle exits the boundary of the flow, integration is stopped. When an animation calls for time steps beyond this point, texels values along the depth of the 3D textures are used. The depth is determine by the number of time steps left when the particle reaches the boundary. The color of the texture to use is a function of this depth value. By guaranteeing that the boundary

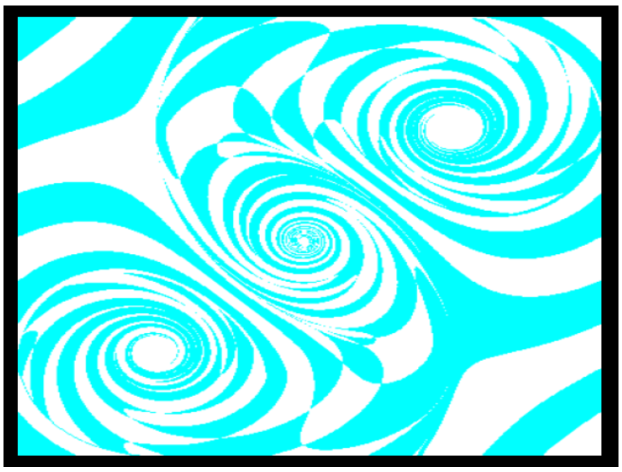

**Fig. 1.** Effect of using constant colors once integration step reaches the boundary. Notice how the colors at the boundaries get streaked out over the image.

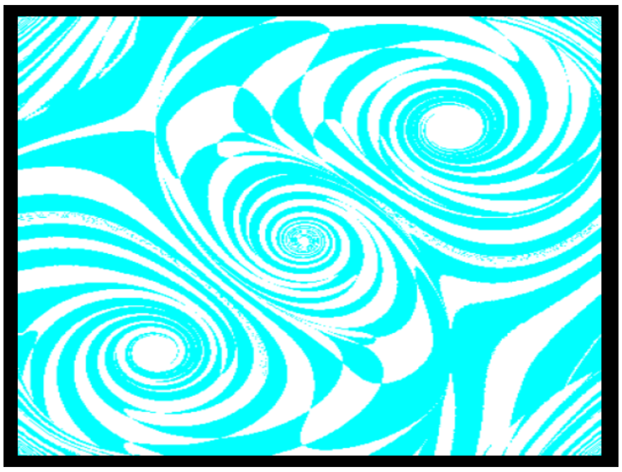

**Fig. 2.** Effect of using periodic boundary conditions. Similar problem as constant boundary conditions but new patterns seem to feed in from the boundaries although they are not very coherent.

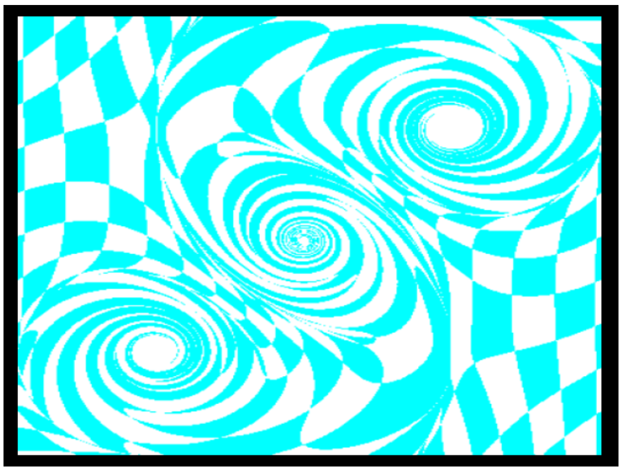

**Fig. 3.** Effect of using a 3D checkerboard texture pattern. Coherent checkerboard pattern appear to feed in from the boundaries into the image.

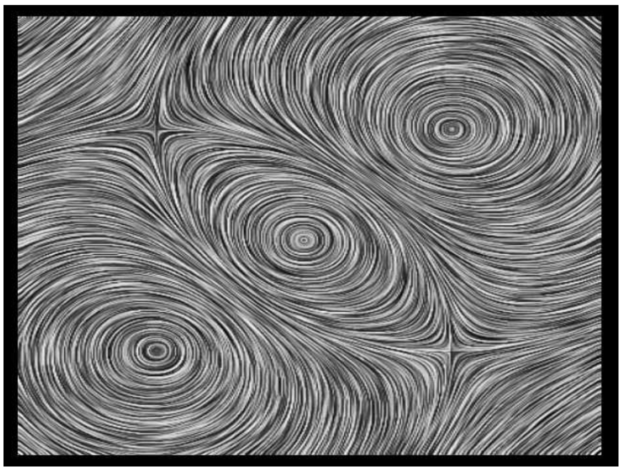

**Fig. 4.** A LIC image that depicts the dynamic vortices flow field used in the previous figures.

textures are different as one goes through the depth of the 3D texture, a dynamic continuous feed effect is achieved. Figure 3 illustrates this approach when a 3D checkerboard texture pattern is used. Note that adjusting the scales and frequency of the checkerboard along the depth of the 3D texture affects the quality of the resulting animation.

### 3.2 Critical Points

The problem of what texture pattern to feed into the image also occurs when a particle reaches a critical point that is source or repelling spiral i.e. where the flow emanates from the critical point. Using the standard advection of texture coordinates with constant color boundary condition will produce an image like Figure 5. What is happening here is that particles around the region of influence of the source are backward integrated into the same point – the source critical point. So, whatever texel value happen to be at that point gets propagated out through the source region. A more realistic effect would have been to see new texture patterns flowing out of the source point. Again, we use the 3D texture approach so that if we end up at the same location during a backward integration step, (i.e. at a source or repelling spiral), the texel value assigned is the next one in depth. This is illustrated in Figure 6.

This texture advection is image based, and the particle advection is performed for each texel. Basically, there is a one to one mapping of the output image and the input image. For each texel in the output image, there is a corresponding texel in the input image.

The computational time of this approach grows linearly as the number of time steps increases. During the initial time step, no integration is performed to show the initial input image. At the second time step, the texels are advected to the next time step. Hence, at time n, the particles would need to be advected by n time steps. However, it is easy to improve the computational time by saving the current positions of the particles at the current time step and then continue the advection to the next time step.

This approach does not require large memory overhead because the integration are performed texel by texel. The only information saved are the resulting output image obtained from the particle integration.

## 4 STREAMLINE CYCLING

As pointed out, a weakness of the texture advection approach, even with the 3D textures for feeding new textures at boundaries and source and repelling spiral critical points, is the lack of coherency of new textures. Another potential drawback is the sensitivity of the visual effect to the 3D texture used. That is, different textures can potentially produce different visual effects even on the same flow field. This section discusses another flow field animation approach. This approach was motivated by the question of how to feed textures into the image when a particle reaches a boundary or a source or repelling spiral critical point. Instead of creating new textures, why not reuse the existing textures in the streamline? The basic idea is to first intersect streamlines with the underlying texture in the image, those textures are then cycled for each streamline. Because the texture features (e.g. a region of the texture such as a part of a checkerboard

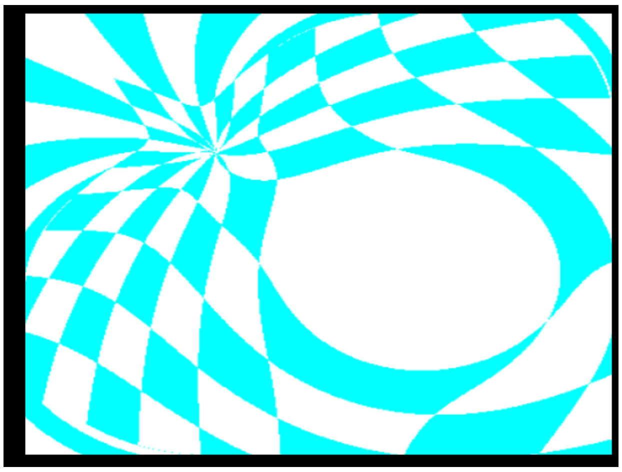

**Fig. 5.** Texel value at the critical source point (lower right) is replicated over time. Eventually the region of influence of the source point becomes homogeneously white.

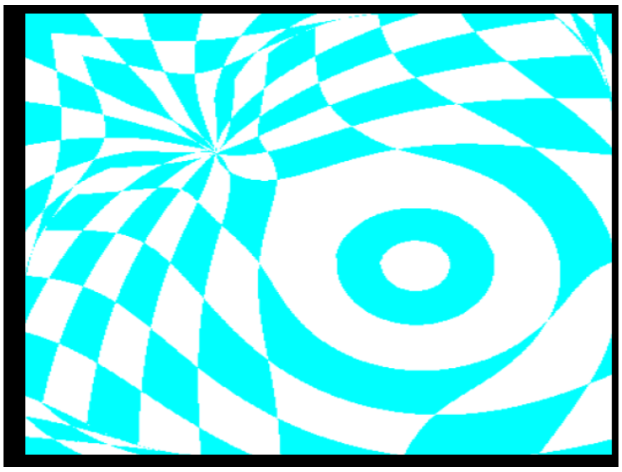

**Fig. 6.** Effect of using a 3D checkerboard texture pattern. New patterns seem to continuously appear from the source.

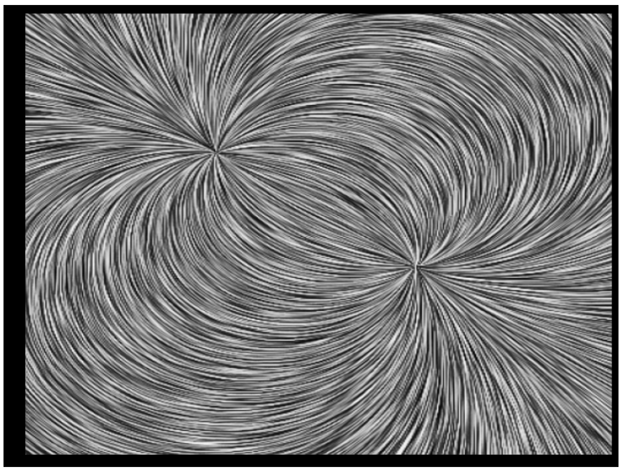

**Fig. 7.** A LIC image that depicts the source and sink critical points in the flow which was used as as input for the previous two figures.

texture) span many streamlines, the movement of the textures are very prominent. Using this approach, flow about the source and sink critical points are clearly depicted.

Here is how the algorithm works. For each texel, the particle is integrated forward and backward in time. The particle is terminated when it (a) reaches the boundary, (b) reaches a source or repelling spiral critical point, or (c) has exceeded the maximum integration steps allowed. While the particle is being integrated along the flow field, the color intensity (RGB) of the texel that it traversed through is saved. For animation, the saved color intensities along the streamline is cycled by a fixed amount in each time step, thus producing an animated image. This is illustrated in Figures 8 and 9 using different texture images. For simplicity, if the streamline has A, B, C, D, and E color intensities and we shift the streamline by one position each time, then we will get B, C, D, E, A in the second time step and C, D, E, A, B in the third time step, and so on. The streamline integration and shifting are performed in texel space. We save the morphed image from each time step to a file and then playback the saved images for animation.

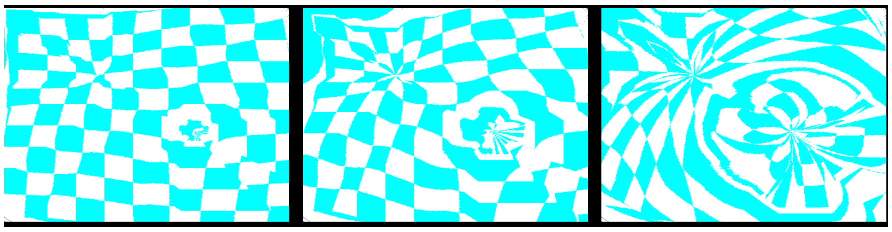

**Fig. 8.** A sequence of time steps (at time steps 5, 10 and 30) from using streamline cycling. The flow about the source is clearly shown.

In our implementation, we seed the streamlines every other 3 texels. Furthermore, the seeds are jittered to avoid artifacts from uniform sampling. We found this seed density tends to give us near full coverage of the image. For each streamline, the texel positions (i,j) and the texel color (R,G,B) values are stored. The maximum length of the streamline is set to 1,000. For the test data that we used, only a small fraction of the streamlines reach this maximum. Hence, the maximum storage requirement per stream line is 1,000*(4+3)= 7KB, where (i,j) is 2 shorts and (R,G,B) is 3 bytes. For a 400x400 image which has seeding at every other 3 texels, it would require approximately 133x133x7KB, or 123.8MB. Note that this is the size of the entire animation as well because we just need to cycle through the texture for each streamline to do the animation.

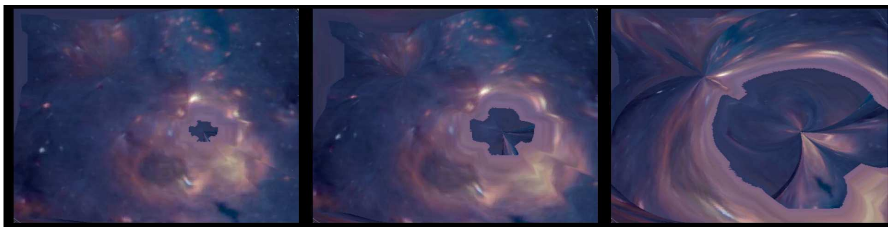

**Fig. 9.** Streamline cycling using the same sequence of time steps (at time step 5, 10 and 30) as Figure 8 but using a different input image. The texture image is that of galaxies. Arbitrary texture images may be used.

In this approach we did not deal with the situation when multiple streamlines traverse through the same texel. The main reason why this situation is not considered is due to the extra memory requirement that is need for normalization – store the sum of the texel color from the streamlines that pass through and the count of the number of streamlines that pass through each texel. Furthermore, additional computation is required to average the texel colors. A related consequence is that because streamlines are calculated independently, textures being cycled on each streamline may not look coherent particularly when coming out of a critical source point. This is an area for further improvement.

There are several advantages to the streamline texture cycling approach. One nice feature is that it is pretty much automated, i.e. the user does not need to specify any parameters besides the input flow field, an input image, and the number of time steps. Based on these information, the program produces an animation that morphs the input image over time to depict the flow field. Another advantage is that unlike the texture advection approach, integration does not continue indefinitely with the animation time. Because we are cycling the textures on the streamlines, the longest integration required would be that of the longest streamline in the flow field. Finally, another advantage is that arbitrary texture images may be used. (The video illustrates this method with a checkerboard and a galaxy image as textures). When specially designed textures are used as in [4, 9], animated LIC-like images can be produced. Likewise, the same technique can be used to produce animations similar to animated droplets (streamlets) where ink droplets are smeared to reveal flow orientation and animated to show the flow direction. The droplets are approximated by small disks with varying intensities. Animation of these droplets is done by either phase shifting of the convolution kernel in consecutive frames or by color table animation. The same effect can be easily simulated using streamline cycling. The key is to start with an input texture that consists of scattered droplets. By cycling through the texture along each streamline, the droplets get smeared and oriented properly and move in the direction of the flow when animated. Figure 10 illustrates how this approach works.

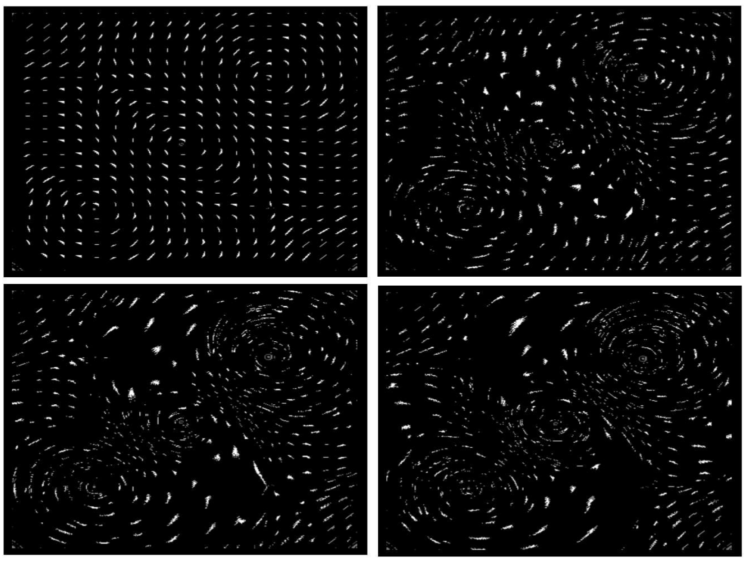

**Fig. 10.** Animated droplets can be achieved using the streamline cycling approach together with an initial texture image of random spots. This sequence shows several four frames from the data set used in Figures 1 to Figure 4.

We found the streamline cycling approach to be most appropriate for depicting the flow about the source and sink critical points in the flow field. The resulting animation gives a striking appearance of the texture been swallowed near the sink critical point and the bursting of texture from the source critical points. This dynamic depiction of the flow is not possible with any static flow depiction including those generated from the LIC algorithm.

## 5 SUMMARY

We have presented how two easy to implement and use flow field animation techniques can be modified to become useful for scientific visualization applications. In particular, the methods presented provided mechanisms for feeding new textures into the animation both at boundary regions as well as critical points such as sources and repelling spirals in the flow field. Conventional particle trace animation techniques usually segment the particle traces into fragments and then move the fragments along the particle path. Our techniques differ from the conventional techniques in that we morph the given texture image to help reveal the dynamics in the flow and we propose solutions on how to feed textures from the flow boundaries and the critical points. There are still several areas for improvements. One area is to extend our approach for 3D velocity flow fields. This would involve using a 3D input texture pattern and it would be best generated procedurally. Currently, our 3D texture boundary approach only generate checkerboard like patterns for feeding into the flow field. We also plan to include other texture patterns.

## ACKNOWLEDGMENTS

We would like to thank David Kenwright for discussions on cross flow coordinates. We would also like to thank the members of the Advanced Visualization and Interactive Systems laboratory at UC, Santa Cruz for their feedback and suggestions. This project is supported by NASA grant NCC2-5281, LLNL Agreement No. B347879 under DOE Contract No. W-7405-ENG-48, DARPA grant N66001-97-8900, and NSF grant ACI-9619020.

## References

[1] L. K. Forssell and S. D. Cohen. Using line integral convolution for flow visualization: curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):133–141, June 1995.

[2] Allen Van Gelder and Jane Wilhelms. Interactive visualization of flow fields. In *Proceedings of Workshop on Volume Visualization*, pages 47 – 54. ACM, 1992.

[3] A.J.S. Hin and F.H. Post. Visualization of turbulent flow with particles. In *Proceedings of Visualization 93*, pages 46–52. IEEE, 1993.

[4] B. Jobard and W. Lefer. The motion map: Efficient computation of steady flow animations. In *Proceedings of Visualization 97*, pages 323 – 328. IEEE, October 1997.

[5] N. Max, R. Crawfis, and D. Williams. Visualizing wind velocities by advecting cloud textures. In *Proceedings: Visualization '92*, pages 179 – 184. IEEE Computer Society, 1992.

[6] Han-Wei Shen and David L. Kao. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):98–108, April-June 1998.

[7] Jos Stam. Stable fluids. In *Computer Graphics*, pages 121–128, Los Angeles, Ca., August 1999. ACM Siggraph Conference Proceedings.

[8] J. J. van Wijk. Spot Noise: Texture synthesis for data visualization. *Computer Graphics*, 25(4):309 – 318, 1991.

[9] Vivek Verma, David Kao, and Alex Pang. PLIC: Bridging the gap between streamlines and LIC. In *Proceedings of Visualization '99*, pages 341 – 348, October 1999.

[10] R. Wegenkittl, E. Groller, and W. Purgathofer. Animating flow fields: rendering of oriented line integral convolution. In *Computer Animation '97*, pages 15–21. IEEE Computer Society Press, June 1997.

[11] Patrick Witting. Computational fluid dynamics in a traditional animation environment. In *Computer Graphics*, pages 129–136, Los Angeles, Ca., August 1999. ACM Siggraph Conference Proceedings.

[12] B. Yamrom and K.M. Martin. Vector field animation with texture maps. *IEEE Computer Graphics and Applications*, 15(2):22–24, March 1995.