Real-time Line-based Flow Visualization

Huaiping Yang[1,2], Wenping Wang[2], Jiaguang Sun[1]
1. Tsinghua University, China
2. The University of Hong Kong, Hong Kong
E-mail: hpyang@csis.hku.hk, wenping@csis.hku.hk, jgsun@tsinghua.edu.cn

Abstract

We present in this paper a method that uses densely distributed streamlines to achieve real-time 2D flow visualization with better visual quality than produced by texture-based methods. We call the method a line-based flow visualization (LBFV) method because it uses a dense set of line segments as basic rendering primitives. Streamlines are commonly used for flow visualization, and they provide more effective visualization of global features of flow data than texture-based methods, due to its integral definition. However, existing streamline-based methods use relatively sparsely distributed streamlines and thus produce lower resolution than texture-based methods. In our method densely distributed streamlines are computed using particle advection, with initial particle positions determined by a novel algorithm, and we use hardware-assisted image blending to render the streamlines efficiently for real-time visualization. Specifically, line segments representing particle advection in the current frame are blended with the image of the previous frame to achieve fast rendering (40-100 fps) as well as the natural decaying effect of the wakes of streamlines. Several intuitive parameters, such as the line width, line color, the total number of particles, and the time step for particle advection, can easily be tuned for producing various visual results. Experimental examples are presented to demonstrate the effectiveness of our method and to compare it with the conventional LIC (Line Integral Convolution) method and the IBFV (image-based flow visualization) method.

Keywords: flow visualization, particle advection, streamlines, image blending.

1. Introduction

Vector fields are frequently encountered in many scientific and engineering disciplines. Faithful and visual representation of vector fields is necessary for accurate and intuitive understanding of the features of vector fields. A challenging problem in vector field visualization is the visualization of flow data, which are typically generated by computational fluid dynamics (CFD) simulations.

Existing streamline-based methods for 2D flow visualization normally use streamlines that are relatively sparsely distributed and carefully placed. We present a novel method for 2D flow visualization that uses densely distributed streamlines generated by a large number of randomly seeded particles; 2000 particles are typically needed for an image of 512×512 pixels. The motion of the particles along field lines is displayed by *streamlets* (i.e. streamline elements), composed of a sequence of connected short line segments. To convey the smooth motion of the particles and their wakes, the displayed image at each frame is generated by α-compositing the rendering of all streamlets at the current frame with the image displayed at the previous frame.

Our method is very efficient; it achieves as high as 100 fps on a commodity PC, thus suits well for real time visualization of flow data. A distinct advantage of the method, owing to its use of streamlets, is that the streamlines of varying lengths can be perceived clearly, thus serving as an effective visual cue for flow velocity.

We call our method line-based flow visualization (LBFV) because line segments are used as basic rendering primitives. Figure 1 compares the snapshots of the visualization results of the same flow field generated by the conventional IBFV (image-based flow visualization) method in [18] and our LBFV method, respectively.

The remainder of this paper is organized as follows. In Section 2 we discuss some related work. Section 3 describes in detail our LBFV method for real time flow visualization. The experimental results are presented and compared with the LIC method and IBFV method in Section 4. We conclude the paper in Section 5 with discussions for future research.

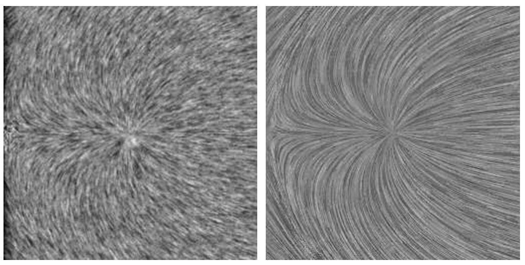
(a) (b)

Figure 1. Flow visualization results produced by IBFV (a) and LBFV (b).

2. Related work

Many methods have been reported in the literature for flow visualization, but we can only review those most relevant results. The reader may consult [12] for a comprehensive overview of flow visualization techniques.

2D vector fields can be represented by hedgehogs, i.e. arrow plots [11]. This approach, however, is not amenable to easy reconstruction of the flow when the arrows are placed on a regular grid. Streamlines and advected particles are also used to indicate relatively sparsely distributed integration paths, but the start-points of streamlines and particles have to be determined with special care to ensure that important features of flow fields are not missed [18]. Turk and Banks [15] propose an image-guided method to iteratively optimize the placement of streamlines. This method produces good visualization quality but with rather slow convergence. Jobard and Lefer [7] give an efficient method for creating evenly-spaced streamlines of arbitrary density, but the method is not fast enough for real-time animation. Stalling et al. [14] present a technique for interactive 3D vector field visualization using a large number of properly illuminated field lines. Note that none of these streamline-based methods mentioned above accommodate high spatial resolution of visualization.

Using spot textures, van Wijk [17] proposes the first method for visualizing a flow field with high spatial resolution; a spot texture is generated for data visualization by inserting distorted spots with random intensity at random locations in the field. This method has inspired several other texture-based or image-based methods. Cabral and Leedom [3] present a powerful Line Integral Convolution (LIC) method. In the LIC method, a random texture is locally convolved along field lines to yield the final image that depicts the flow structure. The LIC method as originally proposed is rather slow; it takes tens of seconds to compute an image because a large number of points on a streamline (typically 20-50) need to be computed for convolution for each pixel. The fast line integral convolution (FLIC) method is devised by Stalling and Hege in [13]. It employs a simple box filter kernel and minimizes the total number of stream lines to be computed. FLIC reduces computational time by one order of magnitude as compared to the original LIC.

While all the above methods provide only a static visualization of the flow data, much effort has recently been made to develop more efficient techniques for animating the motion of flow data at an interactive rate. Heidrich et al. [6] make the first attempt by accelerating the LIC method with graphics hardware, and achieve 3-4 fps on an SGI Octane workstation. Jobard et al. [8] propose a technique for visualizing unsteady flow fields based on a Lagrangian-Eulerian Advection (LEA) scheme. The algorithm is implemented in a parallel fashion on a 4-processor SGI Onyx2 workstation, and reaches about 4-5 fps. Weiskopf et al. [16] present hardware-accelerated texture advection techniques and use programmable per-pixel operations of an nVIDIA GeForce 3 graphics card to achieve 40 fps for flow motion visualization.

Finally, van Wijk [18] presents an efficient method based on image morphing for the visualization of 2D fluid flow. Due to its comprehensive use of texture and image morphing, this method is referred to as Image-Based Flow Visualization (IBFV). The IBFV method uses the advection and decay of dyes. Each frame of a flow animation is computed as a blend between a warped version of the preceding image and a number of background images. High performance of about 50 fps is achieved using standard graphics hardware, but special care must be taken in choosing the background images to reduce artifacts.

3. Line-Based Flow Visualization

The streamline-based approach offers fine representation of flow field lines but, in the way it is currently used, lacks adequate spatial resolution. On the other hand, the texture-based or image-based approach enjoys high spatial resolution and superior rendering efficiency, but exhibits conspicuous noisy appearance and does not depict field lines clearly. The basic idea of our method is simple: a high-resolution visualization of a flow field with motion animation can be obtained by a large number of moving advected particles displayed by streamlets. The displayed image can be composed using hardware assisted image blending techniques for maximum efficiency. Specifically, the wakes of moving particles can be produced naturally by blending the current particle traces, represented by streamlets, with the image from the preceding frame. In this way high quality images can be obtained by displaying at an interactive rate a large number of densely distributed moving particles – typically 2000 particles are used for an image of 512×512 pixels in our experiments.

In the rest of this section we discuss in detail the three main components of our method: 1) representation and animation of advected particles; 2) image blending; and 3) seeding the initial positions of advected particles.

3.1. Particle advection

The particles are initially randomly seeded with a uniform distribution; the addition of new particles during the animation is a bit more complicated and will be discussed in Section 3.3. Let $x_i(t)$ denote the current position of the i^{th} particle p_i at time t. Then its next position $x_i(t + \Delta t)$ can be computed by solving the following equation:

$$\frac{dx}{dt} = v(x). \tag{1}$$

The following three integration methods are commonly used for particle advection in flow visualization [12]: 1) the standard Euler method; 2) the second-order Runge-Kutta method; and 3) the fourth-order Runge-Kutta method. We choose the second-order Runge-Kutta method as an acceptable compromise between computational efficiency and accuracy. The second-order Runge-Kutta method can be expressed as

$$\begin{cases} x^* & = x(t) + \Delta t \cdot v(x(t)) \\ x(t+\Delta t) & = x(t) + \Delta t \cdot [v(x(t)) + v(x^*)]/2. \end{cases} \quad (2)$$

To ensure high approximation accuracy, Δt should be small enough to satisfy

$$\Delta t \cdot max(|v(x)|) \leq \Delta x, \quad (3)$$

where Δx is set to be the length of 10 pixel units in our implementation. When this is satisfied, a line segment connecting the point $x_i(t+\Delta t)$ and the point $x_i(t)$ for the particle p_i is used as the particle advection vector, or the streamlet; otherwise, the streamlet from $x_i(t)$ to $x_i(t+\Delta t)$ will be considered too long, and therefore will be represented by a sequence of connected short line segments. This treatment preserves the smoothness and accuracy of the path lines even when the animation time step Δt is large.

3.2. Image blending

Image blending is used for flow visualization in several existing methods [8] [10] [18]. These methods usually use texture advection techniques and blend successive images at successive frames for the smooth representation of flow motion. In our LBFV method, image blending is done between an image containing a large number of streamlets and the previous frame to produce the long wakes of streamlines for the decaying effect of moving particles.

Suppose that a single particle p_i starts at an initial position $x_{i,0}$ and moves through a sequence of intermediate positions, denoted by circles in Figure 2, to arrive at the current position $x_{i,k}$ at the k^{th} time step; the case of $k = 10$ is illustrated in Figure 2. The line segment $x_{i,9}x_{i,10}$ is the streamlet of p_i at the current frame, and the sequence of streamlets from $x_{i,0}$ to $x_{i,9}$ are captured in the image at the previous frame. Then the blending of these two images produces the effect shown in the rightmost image of Figure 2. Due to the repeated application of α-blending, the path-line of the advected particle p_i becomes less conspicuous as it goes back in time. The wake of the particle p_i, i.e. the clearly visible part of the path-line, can be controlled by using different α values in α-blending. (See Section 4.2.)

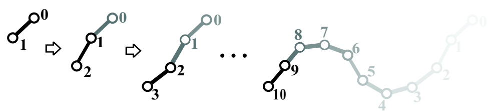

Figure 2. Blending effect for a single particle.

The current frame of streamlets can be rendered quickly into a texture using the OpenGL function glCopyTexImage2D(). The blending operation can then be performed efficiently in OpenGL using a standard graphics card.

3.3. Color of streamlines

The colors used for rendering the streamlets play a critical role in enhancing visualization quality. If the same color is applied to every streamlet, as done in other existing streamline based methods, then all streamlines, which are normally in close proximity of each other, will fill out an image and make all streamlines indistinguishable, since almost all pixels will end up having the same color.

We use the following color scheme for rendering streamlines. We first specify two basic colors, denoted *color1* and *color2*. A particle is assigned a color that is randomly selected from *color1* and *color2*, and this color will then be used for the entire life of the particle. Figure 3 shows an example that applies this color scheme to a steady flow. (See the video clip at http://cgcad.thss.tsinghua.edu.cn/∼yhp/fig3.mpg). Figure 7 provides another example of an unsteady flow.

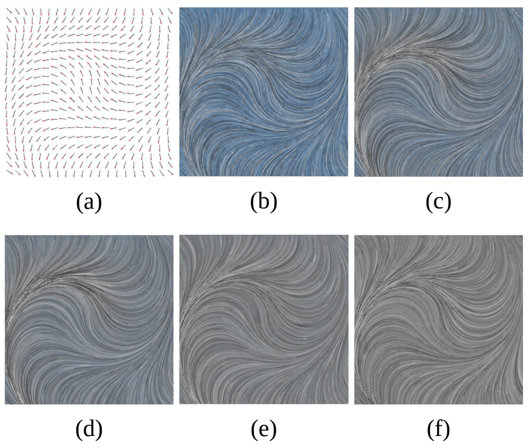

Figure 3. A sequence of images generated for a steady flow (106.4 fps, 2000 particles used). The flow field is indicated with arrow plots in (a). The two basic colors used are black and white, and the initial background color is blue.

3.4. Seed point selection

Streamline placement is a main concern in all visualization methods that use a sparsely distribution of streamlines, since there is a need to maintain certain spacing between the streamlines. Previous work on streamline placement can be found in [7] [14] [15]. Our use of a dense set of moving streamlines poses some requirements different from those by existing streamline-based methods. We require to have all pixels covered by streamlets in order to provide a visualization of high spatial resolution. Meanwhile, each pixel should not be covered by too many streamlets at the same time. These requirements are met in our method by a novel algorithm for seeding the initial locations of particles dynamically during visualization animation.

Our particle seeding algorithm is based on the following observation. Without knowing the features of flow data, i.e. sink and source, we start with a uniform distribution of all seed points of particles. Then, for each frame during flow animation, we check and record for each pixel the number of times it is passed through by streamlets; this number is called the *visit count* of the pixel. At each frame, we deem that pixels having small visit counts are not adequately covered. Therefore these pixels will receive particle seeds with a higher probability than those pixels with high visit counts. The aim of such a strategy is to eventually make all pixels attain the same visit counts.

To speed up the calculation, we divide the whole field/image into a number of cells c_i of uniform size $\Delta c \times \Delta c$, where Δc is set to 4 pixels in our implementation. That is, we deal with cells rather than pixels.

The main steps of our seeding algorithm are as follows:

1. The whole field is divided into uniform cells of size $\Delta c \times \Delta c$. Usually, with $\Delta c = 4$, 128×128 cells are generated for a 512×512 image.

2. Let s_i record the visit count of the cell c_i. Initialize all s_i to be zero.

3. Generate the initial positions of particles with uniform distribution over all cells.

4. For a new frame $t + \Delta t$, generate the streamlets for all advected particles with a time step Δt. Record the total number D of particles that are disappearing at this frame due to absorption into a sink or moving out of image boundary. Increase the visit count s_i of each cell c_i by $d_i(t)$, the number of streamlets that intersect c_i at the current frame. Then subtract $d_i(t - n\Delta t)$ from s_i; therefore, s_i records only the number of visits by streamlets to the cell c_i in the past n frames. Note that $d_i(t - n\Delta t)$ when $(t - n\Delta t) < 0$. A typical value of n used in our experiments is between 8 and 15.

5. Add D new particle seeds into the image. The cell with the smallest visit count are more likely to receives new particles. The detail of this prioritized allocation is explained below. Repeat the steps 4 and 5 until the end of visualization animation.

The following method, often referred to as the *Russian roulette method* [4], is used to select the cell for a new seed point. First choose one cell c_{i_0} from all cells c_i, $i = 1, 2, \ldots, n_c$ ($n_c = 128 \times 128$). Considering that the selection probability E_i of a cell c_i should be low if its visit count s_i is high, the following probability is defined

$$E_i = \frac{1/s_i}{\sum_{i=1}^{n_c}(1/s_i)}, i = 1, 2, \ldots, n_c, . \qquad (4)$$

and $E_0 = 0$. Then, for each particle to be seeded, we generate a uniformly distributed random variable $\xi \in [0, 1]$ and make the following decision: if $\xi \in [\sum_{i=0}^{i_0-1} E_i, \sum_{i=0}^{i_0} E_i)$, then c_{i_0} is selected (see Figure 4). This decision process can

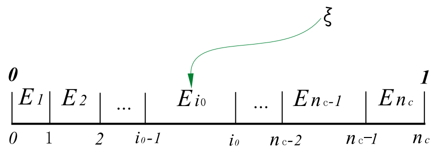

Figure 4. Using Russian roulette to select the cell.

be done quickly using a binary search [5], which has a time complexity of $O(\log n_c)$.

The stochastic algorithm given above has proven effective. An example is given in Figure 5 to show the improvement of the visualization result by using this method as compared with a uniform distribution of the particle seeds.

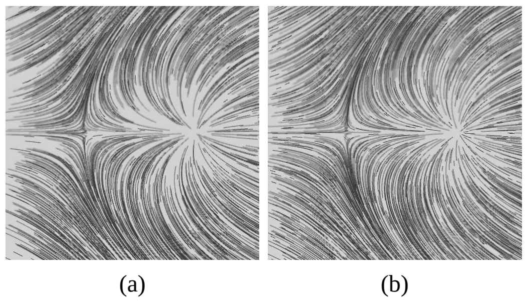

Figure 5. Visualization effect resulting from the uniform distribution of particle seeds is shown in (a). The improved visualization by using our particle seeding algorithm in Section 3.4 is shown in (b). The flow field used is shown in Figure 8(a).

4. Implementation and Experimental Results

We implemented LBFV in Visual C++ 6.0 using the standard OpenGL library. All the experiments were run on a PC with a Pentium 4 1.8GHz CPU and an ATI Mobility Radeon 7500 graphics card. All images have 512×512 resolution, for which 128×128 cells are used for seeding the particles.

4.1. Performance

The LBFV method achieves a frame rate ranging between 40 fps and 100 fps, with the rate dependent on the number of particles that are active in the field. Note that 2000 particles normally suffice to produce satisfactory visualization. Table 1 lists the processing time and frame rates with different numbers of particles used.

4.2. User control

Our algorithm can easily be controlled by adjusting several intuitive parameters, such as line width, line color, opacity or alpha values for blending, and so on. Figure 6

Table 1. Time performance of LBFV.

Num of particles	Rendering time per frame	fps
$2,000$	$9.4ms$	106.4
$4,000$	$12.9ms$	77.5
10,000	$22.7ms$	44.1

shows two visualizations where different line colors are used.

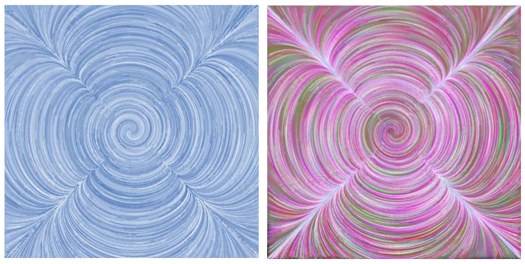

Figure 6. Flow visualization with different color choices in LBFV. Two colors (white and blue) are used in (a); Five different colors are used in (b).

The opacity or alpha values can also be adjusted to produce different blending effects. The blending function is

$$C_n = \alpha_L \cdot C_L + (1 - \alpha_L) \cdot C_T, \tag{5}$$

where α_L is the alpha value of the streamlets in the current frame, C_L and C_T are the colors of current streamlets and the previous frame, respectively, and C_n is the final computed color. The wake of a streamline becomes less conspicuous when a larger value of α_L is used.

4.3. Flow animation

Our method can animate flow visualizations naturally, thanks to the particle advection and image blending method used, as discussed in Section 3.1 and 3.2. Figure 3 shows a sequence of animated images for a steady flow. For an unsteady flow, we get path-lines instead of streamlines using LBFV. Usually a larger value of α_L should be used in order to reduce the influence of earlier path-line segments on the current image. Hence, in this case more particles should be used to maintain the high spatial resolution of visualization. Typically 2500 particles are enough to produce satisfactory visualization for an unsteady flow of the size 512×512 pixel. A visualization of an unsteady flow is shown in Figure 7, with 2500 particles. (The flow data for Figure 7 is from the supplemental source code provided by van Wijk in [18].)

4.4. Comparison with LIC and IBFV

LIC and IBFV are both texture based methods, and they both need one or more noise textures as input. LIC is a

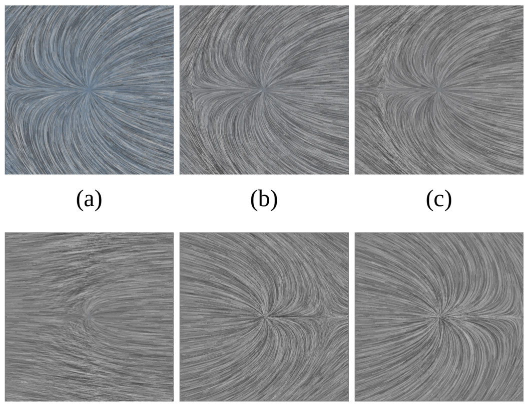

Figure 7. A sequence of images from an animation of an unsteady flow generated by our LBFV method (96.8 fps, 2500 particles used). See also the accompanying video clip at http://cgcad.thss.tsinghua.edu.cn/~yhp/fig7.mpg.

pixel-based approach, where each pixel is computed individually by sampling along a streamline and convolving with a random noise texture [3]. High image quality can be achieved by LIC because distinct streamlines can be perceived clearly. LIC can also be accelerated by graphics hardware to generate interactive animation at 3 to 4 fps on an SGI Octane workstation) [6]. Unlike LIC, IBFV is a texture-patch-based approach, where a number of distorted and texture-mapped patches are used to warp the previous image [18]. IBFV is more time efficient for flow animation than LIC, but the image quality is not as good as LIC.

Our method, LBFV, does not need any noise textures input. Instead we use randomly selected colors of particles to distinguish different streamlines. The blending of successive frames is similar to the process of convolving backward along streamlines in LIC, but done more efficiently. This blending strategy is also used in IBFV. The high spatial resolution of LBFV comes from the use of a large number of densely distributed streamlets, rather than the noise textures used in IBFV. Thus long and distinct streamlines can be seen more clearly in LBFV than in IBFV. The superior efficiency of LBFV comes from the fact that line segments, even a large number of them, can be rendered quickly using a standard graphics card. The LIC method yields better image quality than LBFV (see Figure 8), but it is not as fast as LBFV.

Figure 8 shows the visualization results generated by the LIC method [3], the IBFV method [18] and our LBFV method, respectively, for the same flow field shown in Figure 8(a). It can be seen that LBFV enjoys the favorable properties of both LIC and IBFV, i.e. the fine structure and visual quality of the streamline-based approach and the high efficiency of image-based approach. (The frame rates of IBFV and LBFV for generating the visualizations in Figure 8 are 49.3 fps and 106.4 fps, respectively.)

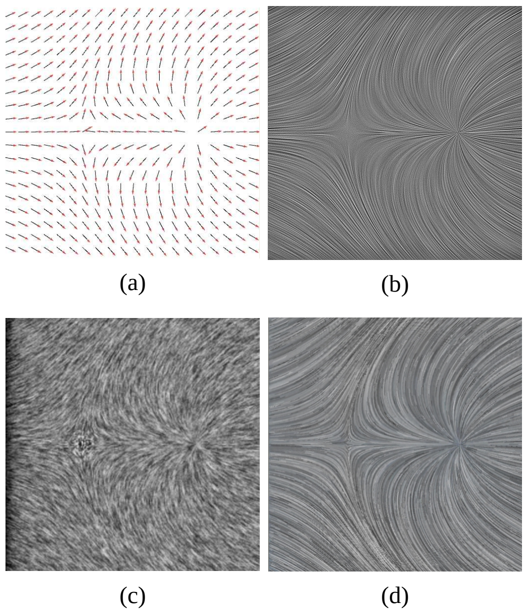

Figure 8. Different visualization results from LIC (b), IBFV (c), and LBFV (d) for the same flow field in (a).

5. Conclusions

We have presented a line-based fluid visualization method for visualizing a 2D flow field. By using a set of densely distributed moving streamlines, our method is capable of producing real-time visualization of a flow field with clearly perceived field lines.

We have experimented with the application of our method to 3D flow visualization. Since the image blending technique is not directly applicable in 3D case, we use OpenGL's accumulation buffer to blend the successive frames with a fixed viewpoint, which is clearly a restrictive and inconvenient requirement for 3D flow visualization. Moreover, further research is needed to enhance the moving effects of 3D streamlets.

References

[1] A. Arvo and D. Kirk, Particle transport and image synthesis, *Computer Graphics*, 1990, 24(4), pp. 53-66.

[2] H. Aref, R. Charles, and T. Elvins, Scientific visualization of flow, In *Frontiers of Scientific Visualization*, John Wisley and Sons, New York, 1994.

[3] B. Cabral and C. Leedom, Imaging vector fields using line integral convolution, In *Proceedings of ACM SIGGRAPH'93*, 1993, pp. 263-272.

[4] F. H. Clark, Methods and data for reactor shield calculations, In *Advances in Nuclear Science and Technology*, 1971, No. 5, pp. 95-183.

[5] R. F. Gilberg and B. A. Forouzan, *Data Structures: A pseudocode approach with C*, Pws Pub Co, 1998.

[6] W. Heidrich, R. Westermann, H. P. Seidel, and T. Ertl, Applications of pixel textures in visualization and realistic image synthesis, In *ACM Symposium on Interactive 3D Graphics*, 1999, pp. 127-134.

[7] B. Jobard and W. Lefer, Creating evenly-spaced streamlines of arbitrary density, *Visualization in Scientific Computing*, In W. Lefer and M. Grave, editors, Springer Verlag, 1997, pp. 43-56.

[8] B. Jobard, G. Erlebacher, and M. Hussaini, Lagrangian-eulerian advection for unsteady flow visualization, In *Proceedings IEEE Visualization'01*, 2001, pp. 53-60.

[9] H. W. Jensen, F. Suykens, and P. Christensen, A practical guide to global illumination using Photon mapping, *SIGGRAPH' 01 Course 38*, Los Angeles, August 2000.

[10] N. Max and B. Becker, Flow visualization using moving textures, In *Proceedings of the ICASW/LaRC Symposium on Visualizing Time-Varing Data*, 1995, pp. 77-87.

[11] W. Merzkirch, *Flow visualization*, Academic Press, Orlando, 1987.

[12] F. Post, B. Vrolijk, H. Hauser, R. Laramee, and H. Doleisch, Feature extraction and visualization of flow fields, In *the State-of-the-Art Proceedings of EUROGRAPHICS'02*, 2002, pp. 69-100.

[13] D. Stalling and H. C. Hege, Fast and resolution independent line integral convolution, In *Proceedings of ACM SIGGRAPH'95*, 1995, pp. 249-256.

[14] D. Stalling, M. Zöckler, and H. C. Hege, Fast display of illuminated field lines, *IEEE Transactions on Visualization and Computer Graphics*, 1997, 2(2), pp. 118-128.

[15] G. Turk and D. Banks, Image-guided streamline placement, In *Proceedings of ACM SIGGRAPH'96*, 1996, pp. 453-460.

[16] D. Weiskopf, M. Hopf, and T. Ertl, Hardware-accelerated visualization of time-varying 2D and 3D vector fields by texture advection via programmable per-pixel operations, In *Vison, Modeling, and Visualization VMV '01 Conference Proceedings*, 2001, pp. 439-446.

[17] Jarke J. van Wijk, Spot noise: Texture synthesis for data visualization, *Computer Graphics*, 1991, 25(4), pp. 309-318.

[18] Jarke J. van Wijk, Image based flow visualization, *ACM Transactions on Graphics*, 2002, 21(3), pp. 745-754.