High-Quality Animation of 2D Steady Vector Fields

Wilfrid Lefer, *Member, IEEE*, Bruno Jobard, and Claire Chédot

Abstract— **Simulators for dynamic systems are now widely used in various application areas and raise the need for effective and accurate flow visualization techniques. Animation allows to depict direction, orientation, and velocity of a vector field accurately. This paper extends a former proposal for a new approach to produce perfectly cyclic and variable-speed animations for 2D steady vector fields (see [1] and [2]). A complete animation of an arbitrary number of frames is encoded in a single image. The animation can be played using the color table animation technique, which is very effective even on low-end workstations. Cyclic set of textures can be produced as well, and then encoded in a common animation format, or used for texture mapping on 3D objects. As compared to other approaches, the method presented in this paper produces smoother animations and is more effective, both in memory requirements to store the animation, and in computation time.**

Keywords— **Flow Visualization, Textured Representations, Animation, Effective Techniques, Multimodal Visualization.**

I. Introduction

EFFECTIVE and accurate visualization of vector fields has been an active research area for several decades. A good flow visualization system should be able to accurately depict topology, direction, and velocity of the flow at any point. These are essential features of a flow field, although additional information could be useful, such as vorticity or precise location and characterization of critical points. Effective methods have been proposed to visualize 2D steady flow fields, which compute either sparse or dense representations of the flow [3], [4], [5], [6]. Although the topology of the flow is correctly depicted, even for small details, direction and velocity are not adequately visualized. More generally it has been proved difficult to depict both direction and velocity magnitude on a static image. Animation is a suitable solution to this problem but raises new issues, particularly achieving temporal correlation between consecutive frames. Some attempts have been made to compute a sequence of frames with one of the previously cited methods [3], [4], but the animations' quality is not optimal and the solutions proposed to address the problem either raise new issues or are not straightforward to implement (see discussion in section II-B). Moreover, computation time and memory cost are expensive because they are a function of the number of frames that are computed. Our approach does not suffer any of these problems and produces perfectly cyclic, variable speed, and smooth animations.

This paper presents a new technique for efficient computation of flow animations, which can be played in real time, even on low-end workstations. This method is based on an original data structure, called the *Motion Map*, in which we encode all the information necessary to produce an animation of the flow. Once the Motion Map has been computed, a very simple and efficient program, the *Motion Map Viewer*, is able to play the animation in real time, using the color table animation technique. The Motion Map can also be used to encode animations using any standard, such as MPEG or AVI. A Java version of the viewer and precomputed Motion Maps are available on our Web site[1]. Our method reaches the following goals:

- accurate visualization of essential features of the flow, including topology, direction and velocity. The flow is visualized using a textured representation, which allows to depict the finest details. By animating the flow, topology and direction of the vector field are properly visualized. The algorithm for encoding motion in the streamline representation allows to accurately visualize the velocity of the flow at any point. The animations produced by this method are variable-speed and perfectly cyclic, which allows to play the animation continuously without any discontinuity.
- real time animation, even on low-end workstations. Once the Motion Map structure has been computed, the animation of the flow can be played using the color table animation technique, which can be achieved in real time on any machine providing access to the system color map.

W. Lefer is with Université de Pau, France. E-mail: Wilfrid.Lefer@univ-pau.fr.

B. Jobard is with the Swiss Center for Scientific Computing at Lugano, Switzerland. E-mail: bjobard@cscs.ch.

C. Chédot is with Université du Littoral Côte d'Opale at Calais, France. E-mail: chedot@lil.univ-littoral.fr.

[1]see at `http://www-lil.univ-littoral.fr/~lefer/VISC/applets/MMview.english.html`

• computational efficiency. Computing a complete animation of the vector field is achieved in a time comparable to the time necessary to compute a single image of the flow with common techniques, such as LIC or Spot Noise.
• memory compactness. The Motion Map is the size of a static image at the same resolution, thus a complete animation of the flow is encoded at the same memory cost as for a single static image. This is an important feature when remote exploration of fluid flow simulation results is concerned, since the user is able to download a complete animation of the flow with no additional time as compared to a static image. Indeed, we have developed a very simple and compact program, called the Motion Map Viewer, which can be downloaded together with a Motion Map. A Java applet version of this program has been written.

The remainder of this article is organized as follows. Section II gives the necessary background on flow visualization. Section III presents our technique while section IV described the process of computing a Motion Map. Section V addresses the issue of visualizing a vector field together with its context. Section VI shows some applications of this method and section VII concludes.

II. Visualization of 2D steady vector fields

This section covers previous works on visualization, and especially animation, of 2D steady flow fields. Since our method uses the color table animation technique to animate the flow in real time, we give here the necessary background concerning this technique.

A. Flow Visualization

Techniques to visualize 2D steady flow fields include icon plots (vectors, hedgehogs), streamlines, feature extraction and visualization [7], [8], and textured representations [3], [4]. Icon plots typically consist in drawing small arrows at different locations in the image, each arrow length and direction being mapped onto the length and direction of the flow at the corresponding location, respectively. Although effective and easy to implement, this technique suffers from poor spatial resolution because the number of arrows that can be drawn in a given area is limited by the size of each arrow.

A streamline is a line tangential to the vector field at any point. Hence, for a steady flow, a streamline is the path followed by a massless particle that would have been released in the flow at a given position. By covering the image with a set of streamlines, the global structure of the vector field can be visualized. The main issue that has to be addressed concerns the placement of the streamlines, in order to avoid density imbalance in the image. Regions where too many streamlines are drawn would appear dark and would lead to confusing and misleading information, whereas, in regions with low density of streamlines, the information is too sparsely displayed and lacks accuracy. Solutions to the streamline placement problem have been proposed in [5], [6] and [9]. Unlike icon techniques, streamlines require to compute particle paths and are more time consuming. But, assuming a good streamline placement is obtained, the quality of the visualization and the spatial resolution are better.

Computing a texture whose patterns show the structure of the flow is a good way to increase the spatial resolution of the visualization because every pixel is used to convey a piece of information. In Spot Noise, circular spots are dropped into the image at random positions and are advected by the vector field [3]. The advection consists in stretching the spot in the direction of the flow, the amplitude of the stretch being a function of the flow velocity. Thus, advected spots are ellipses of the same area as their original spots. To obtain smoother details in highly turbulent regions, spots can be advected according to a curved support, actually the streamline passing through the spot center [10]. Line Integral Convolution (LIC) consists in convoluting a white noise texture with a one-dimensional filter kernel, the filter support for a given pixel being the streamline passing through the pixel [4], [11], [12], [13]. It amounts to filtering the original image with a low-pass filter, which leads to poorly contrasted images. Methods have been proposed to enhance the contrast of LIC images [14] and a dye advection technique can help identifying flow paths accurately [15]. Kiu and Banks have proposed to use a multi-frequency noise texture in order to depict the velocity magnitude but the accuracy is no longer uniform in the image domain [16]. Computation times for textured approaches are higher than for other techniques but the accuracy of the visualization is quite better. LIC images are generally more accurate than Spot Noise ones, although both methods have comparable features [17]. Several methods based on these techniques have been proposed for animating a flow, which are described in the next section.

B. Animation Techniques

Former approaches for visualizing a flow by means of animation were mostly particle techniques [18], [19]. Massless colored particles are released from virtual sources located in the flow at regularly spaced time steps. For each frame of the animation, the new particles positions are computed and all visible particles are visualized by drawing small icons on the image. Techniques that produce a train effect can be used to ease particle paths identification over time. Particle animation methods are easy to implement. A problem is that only a few particles can be visualized at a time, in order to keep them distinguishable from each other. Also determining ideal source positions is not trivial.

Another approach consists in warping a texture according to the vector field [20]. The domain is triangulated, each triangle vertex having initial texture coordinates so that there is no deformation at the beginning. The motion effect is obtained by changing the texture coordinates from one frame to the next. For each frame, new texture coordinates are computed by integrating the vector field equation backward, for the texture to move forward. A fading up and down mechanism allows to limit texture distortions. This technique does not produce cyclic animations.

Animation of steady flow fields is also possible with Spot Noise. A number of new spots are generated at each frame, the frame at which a spot is generated being called its centre-frame. Each spot is dropped onto every frame in a consecutive series centered at its centre-frame. An integration occurs backward and forward to compute spot positions for all frames on which a spot has to be dropped. The intensity of a spot decreases as the distance of the current frame to its centre-frame increases. Thus, appearance and disappearance of spots occur when their intensity is low, which minimizes their visual effects. By building a cyclic sequence of frames, in which the image next to the last one is the first one, perfectly cyclic animations can be obtained. Some blinking effects appear during the animation though, which decrease visualization quality. This occurs particularly in low-speed and/or turbulent regions, for which the visual amplitude due to the movement of the flow can be less than those of the blinking effect.

Several papers address the issue of animating LIC textures. Periodic filters have been used to create sequences of correlated LIC textures [21]. This method works well for constant-speed animation and produces perfectly cyclic animations. Variable-speed animations can be obtained by varying the frequency of the filter function, but the dynamic range of discernible speeds is drastically limited. To solve this problem, Forsell proposed to modulate the rate of the function phase shift as a function of the local vector magnitude [12], [11]. A set of constant-speed LIC textures is precomputed at various phase shifts of the kernel. The final intensity of a pixel in a given frame is computed by interpolating pixel intensities between the two closest precomputed frames, that is the frames with the closest filter kernel phase values. Stalling and Hege noticed that this approach does not yield cyclic animations and that applying kernel phases to neighboring pixels would lose correlation, thus introducing spatiotemporal aliasing effects. As a consequence, the texture may appear to move in the opposite direction in some situations. To avoid this problem only frames for which filter kernel phases are correlated are kept. Then a frame blending technique is used to produce a periodic sequence of N frames by composing a set of $2N$ LIC textures [13]. A quite different approach to produce animations of LIC textures has been proposed in [22], where one-dimensional LIC textures are computed from a constant flow field and then mapped onto a dense set of streamlines. An animation can be obtained by moving the textures along the streamlines. This requires that the height of the LIC texture be at least the size of the longest streamline. A consequence is that moving the textures can not be achieved by simply updating the color table, hence real time animation can not be ensured.

The problem of animating unsteady flows is quite more difficult and only a few techniques have proved to be effective. These techniques can be obviously applied to steady vector fields as well, although none of them is as effective and accurate as the method presented in this paper because they mainly address the problem of correlating frames computed at successive time steps. The UFLIC method is based on LIC and uses a time-accurate value depositing scheme, which performs a temporal convolution of an initial white noise texture, and a successive feed forward algorithm to maintain a high temporal coherence between successive frames [23]. This approach achieves good spatial and temporal coherence but during the animation it is difficult to understand the motion of the flow. Because each image is obtained by a temporal convolution over a number of time steps, regions with low turbulence give rise to highly contrasted areas with zebra stripes of arbitrary thickness, while highly tur-

bulent regions tend to produce blurred zones. Indeed the zebra stripes obtained in regions of low turbulence are oriented in the direction of the flow, they look like thick streamlines and during the animation their shapes are slowly modified so that they actually seem to move in a direction orthogonal to the direction of the flow. The approach presented in [24] is original because it works with streamlines, which are said not to be suitable for unsteady flows. Instantaneous states of the vector field are successively visualized by means of correlated frames, leading to a smooth animation. Recently the most advanced graphic hardware features have been used to produce smooth animations of unsteady 2D vector fields in real-time [24].

C. Color Table Animation

Color table animation is a simple and effective technique for animating an image [25]. It works only with a false color visual, where each pixel of the image is assigned an entry (also called an index) in a color table. If the color table entries are shifted cyclically, one place at a time, each pixel will be assigned subsequent colors in the color table, one color at a time. To obtain a moving effect, that is to have the sensation that objects in the image are moving in some direction, it is necessary for each pixel to be correlated to the next, in the direction of movement. The movement speed can be controlled by assigning consecutive pixels the same color table index (see Figure 1). What makes this technique really effective is that the image content does not need to be reloaded from one frame to the next but rather only the color table entries are shifted cyclically. Thus, high animation rates can be achieved, even on low performance graphic devices. It should be noted that, since the color table entries are shifted in a cyclical manner, the maximum number of distinct frames in such an animation is given by the physical size (number of entries) of the color table, which is hardware dependant (256 for common graphic boards).

In [26] this technique has been exploited to visualize a 3D flow. A number of seed points, proportional to each cell's mass, are randomly placed in each cell, from which particle paths of a fixed number of segments (1, 3 or 12) are computed. The velocity is considered as constant along a segment. The color table is filled so as to define a small "wave". Color indexes are assigned to segments ends so that shifting the color table entries makes the waves moving along the flow paths. The hardware is used to perform a linear interpolation of colors along each segment. Particle paths

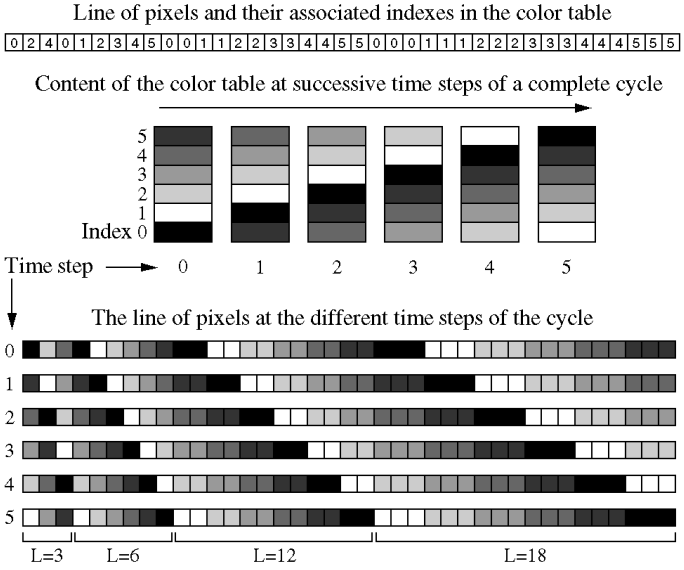

Fig. 1. **The color table animation technique.** Top: a line of pixels with their indexes at time step 0. Middle: content of the color table for 6 consecutive steps of the animation. Bottom: content of the line of pixels for each time step (i.e. its successive contents for a complete cycle of the animation). The speed of movement during the animation can be controlled by assigning consecutive pixels the same color table index. In this example, pixels are correlated along the line in a cyclical manner (sequences of 0, 1, 2, 3, 4, 5). Let us call each set of consecutive pixels with the same value a span. Note that spans on the left are 1 pixel long, span in the middle are 2 pixels long, and spans on the right are 3 pixels long. In this way we are able to produce 3 different speeds when shifting the color table entries. The flow will seem to speed up as it moves from the left to the right and the speed on the right part of the pixel line will be 3 times the speed on the left part. The speed of the flow is proportional to the number of pixels in the current span. The color index $Index_{P_i}$ to assign to pixel P_i is given by:

$$\begin{cases} I_{P_0} = 0 \\ I_{P_i} = I_{P_{i-1}} + k_i \\ Index_{P_i} = [I_{P_i}] \, mod \, N \end{cases}$$

where I_{P_i} is an increasing series in $\mathbb{R}^+$, k_i is a positive real increment, *mod* is the modulo function, and N is the number of indexes. Using a modulo leads to perfectly cyclic animations. The speed of the flow is determined by k_i. Variable-speed animation can be obtained by varying k_i. For instance the values of the 39 pixels of the figure above have been obtained with $k_i = 2 \quad \forall i \in \{1, \ldots, 3\}$, $k_i = 1 \quad \forall i \in \{4, \ldots, 9\}$, $k_i = 1/2 \quad \forall i \in \{10, \ldots, 21\}$, and $k_i = 1/3 \quad \forall i \in \{22, \ldots, 38\}$. Here the speed of the flow has been made piecewise constant so as to be able to map a complete cycle of the color table onto each constant-speed segment of the line. Let us remark that the length L of each segment is given by $L = N/k_i$. Here the minimum value of L is $L_{min} = 6/2 = 3$ and the maximum value is $L_{max} = 6/(1/3) = 18$.

are rendered as opaque using Z-buffer.

In [1] we introduced a new approach, which uses the color table animation technique to produce perfectly cyclic, variable speed animations of an arbitrary number of frames. We extent our method to deal with multimodal animation, where an additional scalar field is visualized during the animation [2]. The main features of this approach are detailed in the remaining sections and we give new elements for the formalization and the assessment of this method.

III. The Motion Map

Our approach is based on the computation of a so-called *Motion Map*, which contains all the motion information about the flow field and from which perfectly cyclic animations can be produced. This section describes the principle of the method, how motion information is encoded in the Motion Map, and explains the parameters that control visualization quality.

A. Method Overview

The principle of the method consists in computing a dense set of streamlines, which are colored so that patterns appear on them. The animation is obtained by making the patterns moving along the streamlines. An important feature of this approach is that the movement of the patterns is obtained by shifting the entries of the color table cyclically. It relies on the fact that for steady flow fields, flow properties are constant over time. Hence the streamline placement, once computed, remains valid for all frames. Thus, we just have to concentrate on coloring the streamlines appropriately.

The algorithm starts by filling the image with a dense coverage of streamlines so that all pixels are covered. A pattern is defined as an ordered sequence of fixed colors. For instance we can use a gradation of colors. A single pattern will be use for all the image. Each streamline is colored by a sequence of consecutive patterns. The animation is obtained by moving the patterns along the streamlines. The system works in false color mode, each pixel being assigned an index in a color table. The color table is filled with the ordered colors of the pattern. Shifting the color table entries will make the flow moving in the right direction, using the technique described in section II-C. Pixel values are computed as a function of the flow velocities, in order to achieve variable-speed animations. Once the image has been computed it contains all the information necessary to visualize direction, orientation and velocity of the vector field accurately. We call such an image a *Motion Map*. By defining values of consecutive pixels on a streamline as values of a continuous and periodic function, perfectly cyclic animations can be obtained.

B. Pattern Parameterization and Image Quality

Each pattern will be curved according to the shape of the streamline onto which it is mapped. Pattern identification relies on a good correlation between colors of consecutive pixels, the transition from a color to the next being smooth enough so that pixels appear correlated in the direction of the flow and not correlated in other directions. Two additional factors have a strong impact on the quality of the visualization, that is the ability to track patterns in the flow: contrast and size of the pattern. The contrast can be defined by the range of colors used in the pattern, i.e. the color table content. It is obvious that the more contrasted the image, the easier pattern identification is. We can for instance choose a regular gray ramp from black to white. Let us remark that, based on our observations, generally only 6 or 8 distinct colors are enough to obtain a good correlation along streamlines, so that flow paths can be easily identified. Let L be the length of a pattern in pixels. L can be expressed as a number of sample points as well. Figure 2 illustrates the visual results obtained with different values of L for a constant speed vector field. Let us note that the criteria for evaluating the quality of such images depend on whether static images or animated sequences are concerned. Thus, image on the left of Figure 2 depicts more accurately the flow features in a static image whereas those on the right is more suitable for animation.

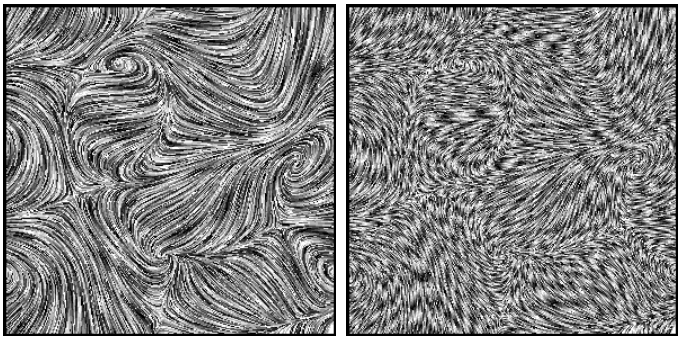

Fig. 2. **Effect of the pattern length.** A constant speed vector field is visualized using two different values of L. Left: $L = 50$. Right: $L = 10$.

In case of variable-speed vector fields, using the technique described on section II-C leads to different

values of L (see Figure 1), with small (resp. great) values of L for regions of low (resp. high) velocities. Thus, we have to determine a minimal value of L, let it be L_{min}, for the slowest regions, and a maximum value, let it be L_{max}, for the fastest ones (see Figure 3). For a given vector field, we set L_{min} and L_{max}, independently of the vector field values, and then the velocity range of the vector field is mapped linearly between L_{min} and L_{max}. Not any value of L_{min} and L_{max} is valid though. Obtaining good quality animations imposes limits to these values. The lower limit for L_{min} is the minimal number of sample points for the movement of the pattern to be perfectly followed and understood, which means direction and velocity of the vector field should be clearly depicted. A minimum of 3 pixels is actually necessary to clear up the ambiguity on the direction of the flow. There is also an upper limit for L_{max} in order to avoid for a single pattern to be so long that it could be difficult for the human eye to capture it at whole and hence to track it during the animation. This limit is expressed in pixels and can be defined by the user based on a visual assessment.

C. Motion Encoding along Streamlines

Each streamline is computed as a series of consecutive evenly spaced sample points (see section IV for details). The motion encoding algorithm aims at obtaining a good correlation between consecutive pixels on a streamline so that the flow will seem to move smoothly along the streamline. Because an approximation occurs during rasterization, color table entries are assigned to the streamline sample points rather than to the pixels covered by the streamline, thus the problem amounts to correlating consecutive sample points.

Once a first sample point has been determined, subsequent sample points are correlated to their predecessors using a formula similar to those given in Figure 1. In order to make the speed of the animation proportional to the flow velocity at any point, k_i is replaced by a linear function of the velocity. The first sample point is assigned a random value rather than 0, so as to avoid the formation of undesirable patterns in the image. The formula used to obtain the index of the current sample point is given by:

$$\begin{cases} S_{P_0} = random(N) \\ S_{P_i} = S_{P_{i-1}} + k(V_{P_i}) \\ Index_{P_i} = [S_{P_i}] \bmod N \end{cases} \quad (1)$$

where $random(N)$ is a function that returns an integer in the interval $[0, N-1]$, P_i is the current sample point, S_{P_i} is an increasing series in $\mathbb{R}^+$, V_{P_i} is the flow velocity at P_i, $k(V_{P_i})$ is the instantaneous contribution of the velocity at P_i, and $Index_{P_i}$ is the color table index for P_i. The next section gives two different expressions of function k, in order to produce animations with various features.

D. Expression of function k

According to equation 1, $k(V_{P_i})$ is a function that gives the increment value of S when moving from P_{i-1} to P_i, as a function of V_{P_i}. In order to obtain a linear mapping of velocities, k should be linear, that is:

$$k(v) = a \times v + b \quad (2)$$

As stated previously, $k(v) = N/L$ (see Figure 1), hence to determine a and b we can use the limit conditions:

$$\begin{cases} k(V_{min}) = N/L_{min} \\ k(V_{max}) = N/L_{max} \end{cases} \quad (3)$$

which yields:

$$k(v) = \frac{N}{L_{max}} \left(\frac{v - V_{min}}{V_{max} - V_{min}} \right) + \frac{N}{L_{min}} \left(\frac{V_{max} - v}{V_{max} - V_{min}} \right) \quad (4)$$

Figure 3 shows the large and continuous dynamic range allowed with this index allocation method for a horizontal flow of increasing speed. This range is quite larger than those obtained by varying the length of the convolution filter in the LIC method, which means that we are able to depict more distinct velocities in the animation.

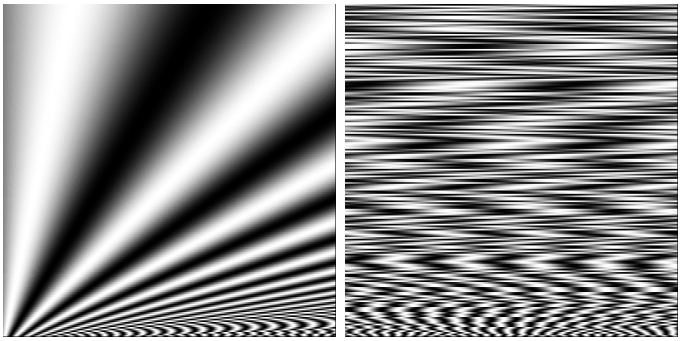

Fig. 3. **Dynamic range of the Motion Map.** Animation of a horizontal shear flow: velocities increase linearly from bottom to top, the ratio between velocities for bottom and top lines being 25. For both images: $L_{min} = 20$, $L_{max} = 25$. Left: $S_{P_0} = 0$. Right: $S_{P_0} = random(N)$.

Although k is linear in v the ratio between velocities in the vector field is not kept in the animation. For instance if $V_{min} \to 0$ then $V_{max}/V_{min} \to \infty$. In order to obtain a constant ratio between a velocity in the vector field and its corresponding speed in the animation, it is necessary to state that:

$$\frac{L_{max}}{L_{min}} = \frac{V_{max}}{V_{min}} \tag{5}$$

If we replace V_{min} in equation (3), we obtain the new limit conditions:

$$\begin{cases} k(V_{min}) = \frac{NV_{max}}{L_{max}V_{min}} \\ k(V_{max}) = N/L_{max} \end{cases} \tag{6}$$

which leads to a new expression of function k, which preserves the velocity ratios:

$$k(v) = \frac{N}{L_{max}} \left(\frac{v - V_{min}}{V_{max} - V_{min}} \right) + \frac{NV_{max}}{L_{max}V_{min}} \left(\frac{V_{max} - v}{V_{max} - V_{min}} \right) \tag{7}$$

Since the physical limit for L_{min} is 3 (see section III-B), it means that we must have $k(V_{min}) \leq N/3$, and hence:

$$V_{min} \geq \frac{3V_{max}}{L_{max}} = V_{threshold} \tag{8}$$

Practically, since a vector field generally has critical points, where the velocity is 0, we use $v = Max(v, V_{min})$ in equation (7). It means that the flow will go faster than it should do in regions of low velocity, but the movement will keep slow. An advantage is that it allows to depict direction and orientation properly everywhere, whereas a too slow movement would not allow to understand the behavior of the flow. Figure 4 shows a frame of a variable speed animation with its corresponding velocity map.

In case of a constant-speed vector field ($V_{min} = V_{max}$), we use a simpler function:

$$k(v) = 1 \tag{9}$$

The next section gives details on the different stages of the Motion Map computation, such as selection of seed points and streamline integration.

IV. Computation of the Motion Map

The Motion Map is a two-dimensional array of color table indexes of the same resolution as the final image. Streamlines are integrated in the vector field space and rasterized in the Motion Map.

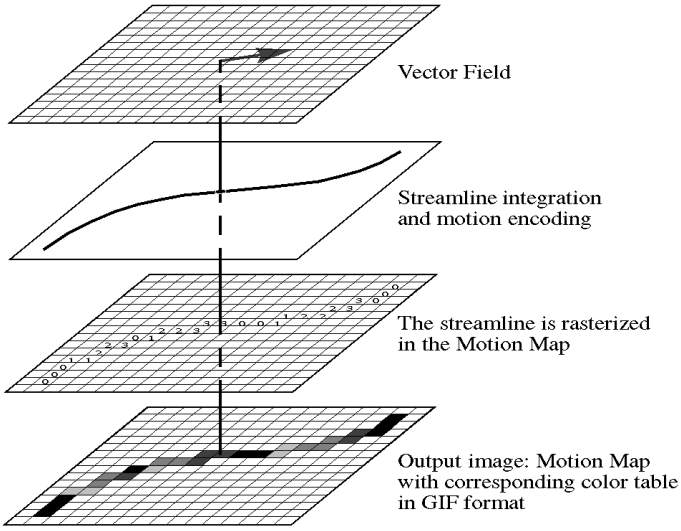

Fig. 5. Computation of the Motion Map.

A. Algorithm

The process of computing the Motion Map is illustrated on Figure 5. The first step consists of computing a streamline, which is constructed as a list of evenly spaced sample points. The motion information is encoded while building the streamline, that is a color table index is assigned to every newly created sample point. The second step consists in rasterizing the streamline, that is we determine all the Motion Map cells covered by the streamline and we assign them an index. These two steps are repeated in a loop until the Motion Map is entirely filled. Actually a streamline is rasterized as its sample points are computed so as to stop the integration when the streamline reaches an already covered cell of the Motion Map (see section IV-C). The last step consists in encoding the Motion Map together with the content of the color table in a common format. We use the GIF format so that the Motion Map can be visualized as a static image using any common image viewer or Web browser. The main issues that have to be addressed are the placement of seed points, the streamline construction and the detection of intersections between streamlines. Solutions to these problems are given in the following sections.

B. Seed Point Selection

The computation of a streamline starts by determining the first sample point, called a seed point, from which the integration of subsequent sample points occurs. Several seed point selection techniques have been proposed, especially for the computation of LIC images. For instance it is possible to process seed

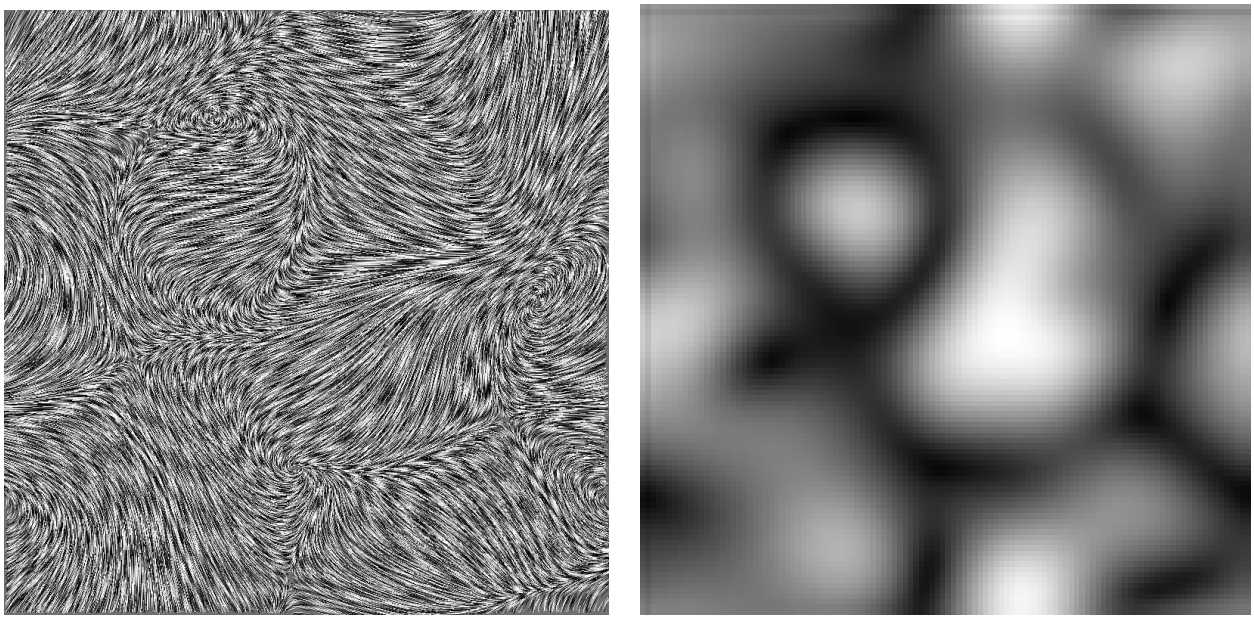

Fig. 4. **Variable speed animation.** Left: frame of a variable speed animation ($L_{min} = 5$, $L_{max} = 30$). Right: magnitude distribution: areas of high magnitude (in white) correspond to long patterns in the image on the left.

point selection in scanline order, to divide the image in blocks to ensure a uniform distribution of seed points, or to use Sobol quasi random sequences [27]. We propose a new and more effective algorithm. Firstly, seed points positions always correspond to cell centers in the Motion Map. Therefore the process of seed point election amounts to choosing a free cell. For efficiency reasons (cache management) the Motion Map is stored as a one-dimensional array, cells being stored in scanline order. Each time a new streamline has to be computed, a cell is randomly elected as a new candidate for starting it. If it is already covered by another streamline, a direction is chosen (between forward and backward) and we scan the one-dimensional array in this direction until either a free cell has been found or we have reached the initial position. The scan is performed modulo the size of the array. The algorithm for choosing a new seed point follows:

```
Choose a new cell randomly
If this cell is already covered Then
  Choose a direction randomly between forward and backward
  Do
    Move one cell in that direction modulo Motion Map size
  While current cell is covered And is not the initial cell
EndIf
If current cell is covered Then
  return NONE
Else
  return current cell
EndIf
```

C. Streamline Computation

Once a valid seed point has been found, we process the integration forward and backward from that point, in order to compute the positions of all sample points for the current streamline. Basically each integration step yields a new sample point. There is a wide family of integrators that can be used to integrate flow paths [13]. We use an integrator that produces evenly-spaced sample points, such as DOPRI5 [28]. Thus we are able to determine the best trade-off between accuracy of the integration and computation time, which are functions of the number of computed sample points.

The construction of a streamline stops when we reach a singularity of the flow (source or sink), or the border of the computational domain has been reached, or the current cell has already been set (intersection). For each new sample point we determine the cells covered by the segment joining the last sample point and the current one (see section IV-E) so that we are able to perform the intersection test.

D. Increasing Pixel Correlation

In order to make the animation as smooth as possible it is necessary to achieve a high degree of correlation between pixels in the final image. We can define the degree of correlation in the Motion Map as the proportion of correlated pixel pairs, that is pairs of consecutive pixels of a streamline. With this def-

inition, it becomes obvious that it is better to cover the Motion Map with longer streamlines as much as possible. For this reason we do not stop the streamline construction as we hit another streamline for the first time but rather we continue the integration some steps forward. The reason to proceed like this is due to the fact the distance between two adjacent streamlines is obviously not constant but rather depends on the local structure of the flow. Thus, two streamlines with a distance of several cells between them at a place could go closer to each other so that their separating distance falls under one cell, and then go farther again. Our experiences showed that streamlines are often stopped while they should have been extended several integration steps forward. To increase the correlation between pixels the algorithm proceeds as follows. As we start computing a new streamline, a counter is set to 0. When we detect an intersection with an existing streamline, that is the current streamline crosses an already covered cell, we just compute the new index but nothing is performed in the Motion Map, and the counter is incremented. When the counter has reached a certain value, the streamline construction is stopped. With this method, there are some pixels along a streamline that are not correlated to the others, but they are correlated to the pixels of another streamline intersecting this cell, whose shape at this location resembles the shape of the current streamline.

Of course as more intersections are allowed, the more expensive the computation, because more integration steps are performed that do not cover any cell. Figure 6 shows the results we obtained by varying the number of intersections, both in terms of degree of correlation and in terms of computation time.

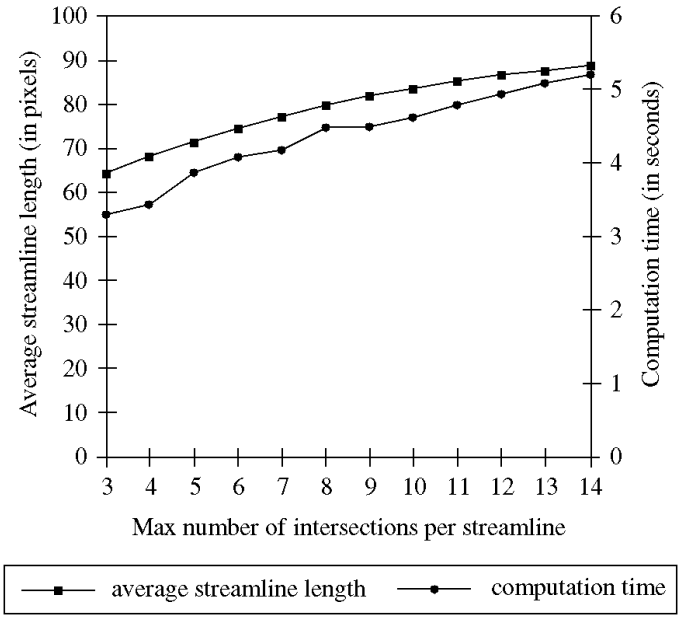

Fig. 6. **Degree of correlation and computation time as a function of the number of allowed intersections.** For all these experiments, the resolution of the image was 512×512. We can see that, when the number of allowed intersections increases, the average streamline length increases, which means that the degree of correlation is higher, but the computation time increases too. A good trade off has to be found between quality of the animation, which is a function of the degree of correlation, and the computation time.

E. Streamline Rasterization

A streamline is rasterized as a polyline using a modified Bresenham's algorithm [29]. For accuracy reasons, the distance between two consecutive sample points should be related to the resolution of the computational grid, typically half a cell of the grid. If the resolution of the Motion Map is lower than those of the vector field, several segments will project onto the same cell, which means that we would have processed several integration steps needlessly. To avoid this problem, only segments covering at least 2 cells are actually drawn. This is done by computing the distance between the end of the last rasterized segment and the current sample point. If the resolution of the Motion Map is higher than those of the vector field, each segment joining two consecutive sample points yields a rasterized segment covering several cells of the Motion Map, and thus the streamline will be approximated by a polyline. To minimize the effect of this approximation, the distance between consecutive sample points is never greater than the width of a Motion Map cell. Though, due to Bresenham's algorithm, several Motion Map cells may be covered by a single segment. In this case all the covered cells get the same value, which is actually the value assigned to the sample point at the left end of the segment (see Figure 7 left). Another solution to this problem has been proposed in [13], where the integration step was related to the resolution of the vector field only and a Hermitte interpolation process was applied to create intermediate sample points before rasterization.

The modification of Bresenham's algorithm concerns the way in which already covered pixels are processed. In the former Bresenham's algorithm, the final color of a pixel is the color of the last segment that has been drawn on it [29]. The problem here is that streamlines computed at the end of the process are generally shorter than those computed at the be-

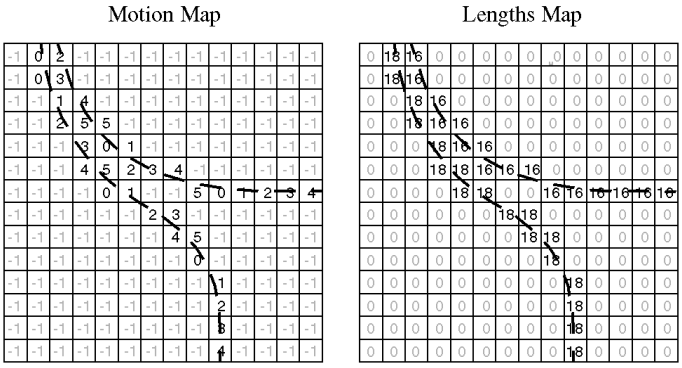

Fig. 7. **Drawing streamlines in the Motion Map.** The Lengths Map is used to determine, in case of intersection, whether the current cell content should be set with the value related to the current streamline or the old value should be kept. The cell gets the value related to the longest streamline. The Lengths Map is a good measure of the degree of correlation in the image.

ginning because they are computed when more cells have already been filled, hence they more likely hit other existing streamlines. Thus, pixels tend to be correlated to the shortest streamline that covers them. With our algorithm a cell gets the value of the longest streamline that passes through it. The goal of this modification is to increase the average length of the streamlines in order to improve the degree of correlation in the image. To achieve it we compute a so-called lengths map, which is a two-dimensional array of the same resolution as the Motion Map (see Figure 7). A cell of the lengths map contains the length of the longest streamline that has been rasterized in the corresponding Motion Map cell. In order to get a better view of the degree of correlation for a complete image, we have visualized the lengths map by mapping its values on a gray ramp. Figure 8 compares two images of the same vector field obtained with the former Bresenham's algorithm and our algorithm.

F. Motion Map Output

Once the Motion Map has been computed, we are able to generate an animation of the flow. There are two different ways to produce a perfectly cyclic animation derived from the Motion Map. The first one uses the color table animation technique presented in section II-C. We have developed a simple program, the *Motion Map Viewer*, which takes as input a GIF image supposed to be a Motion Map, and animates the flow by shifting the color table entries.

The second way to use the Motion Map consists in producing a set of textured frames, called a cyclic set of textures. We build N images of the vector field, each image being obtained by replacing the indexes in the Motion Map by their corresponding RGB colors in the color table. Color table indexes are shifted cyclically from a frame to the next, thus producing N different frames. Since those frames are correlated together, playing them in sequence yields a smooth animation of the flow. Such a sequence of frames can be used for texture mapping onto 3D objects or they can be stored using any animation format, such as animated GIF or MPEG.

V. Visualizing a Flow and its Context Together

Visualizing direction, orientation and velocity of the flow is sometimes not sufficient to understand the behavior of a complex dynamic system. It may require to visualize additional information together with the flow. We consider the problem where a scalar field has to be visualized during the animation and we propose two different methods to solve this issue. The first one uses transparency to visualize the scalar data underneath the flow whereas the second one extends the Motion Map technique to deal with multimodal visualization. In the remaining of this paper we suppose the scalar field be represented as a color image of the same resolution as the Motion Map, each scalar value being mapped onto a color. We call such an image a *Contextual Map*.

A. Using Texture Transparency

When the Motion Map is used to produce a cyclic set of textures, the number of available colors is no longer limited by the size of the color table because we use RGB values instead of color table indexes (see section IV-F). Hence it is possible to use as many different colors as needed. The technique consists in computing the color of a given pixel for a given frame as a function of its index in the Motion Map and the color of the corresponding pixel in the Contextual Map. From the Motion Map we compute a cyclic set of N luminance textures (consider the color table as a gray ramp from black to white). During the animation, the textures are alpha-blended with the Contextual Map, one next to the other. This can be achieved in real time only on machines with hardware texture mapping facility. The animation of the flow around the globe (see Figure 13) has been produced with this technique.

This method works well for displaying contextual information during the animation of the flow, but it

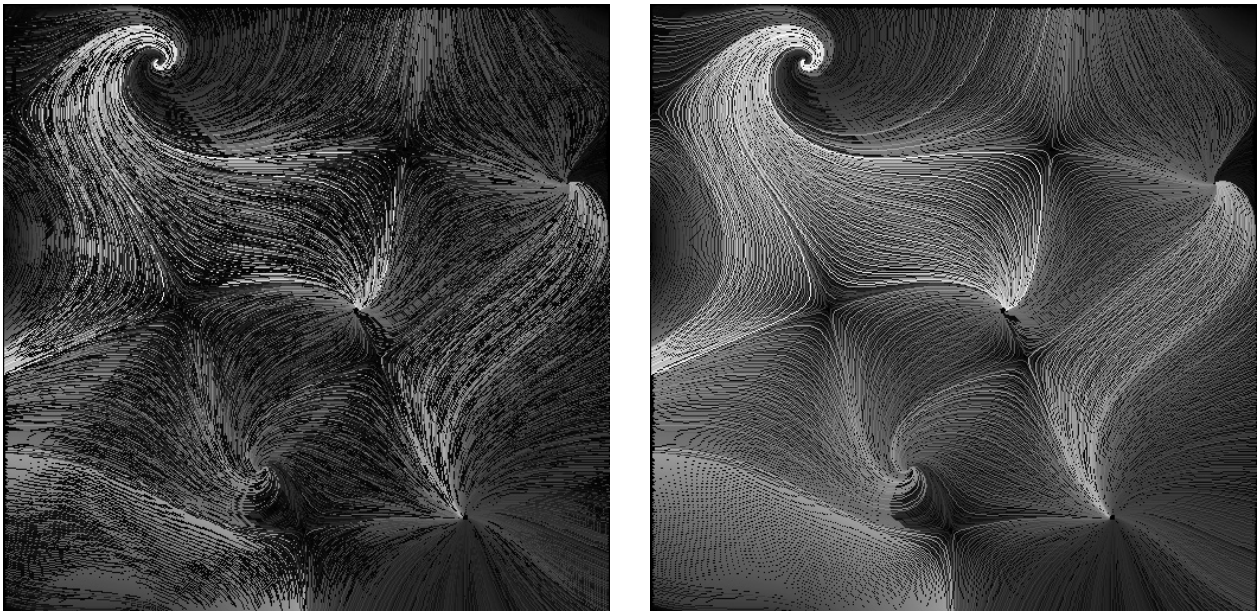

Fig. 8. **Visualization of the pixel correlation.** Two different rasterization algorithms have been used to compute two Motion Maps for the same vector field. Left: original Bresenham's algorithm. Right: our algorithm. The lengths maps are visualized by mapping their values to a gray ramp from black (lower correlation) to white (higher correlation). The image on the right shows a quite better correlation between pixels. On the left image, long (bright) streamlines are often cut by shorter (darker) ones. The average of values of both lengths maps are 44.4 and 63.9, respectively.

is only suitable when the Motion Map is used to produce a cyclic set of textures. In the case of real time animation using the color table animation technique, we propose a more sophisticated method.

B. Using Color Table Animation

In order to animate the flow using the color table animation technique, we propose to compute a so-called *Multimodal Motion Map* so that the scalar data will be displayed underneath the flow. The Multimodal Motion Map is computed by combining the Motion Map with the Contextual Map, as described below.

B.1 Principle

A Multimodal Motion Map is a Motion Map for which the color table has been extended and divided into a number of distinct sections, each section acting as an independent motion map color table. Sections may have different sizes, although they are generally the same size. Each pixel in the final image is assigned a section. During the animation each section is shifted independently, that is each pixel will be assigned all colors of its own section subsequently. The idea of the Multimodal Motion Map is to compute a different section, that is a different set of colors, for every color in the Contextual Map. For instance if we have a binary image with blue and red pixels, we will build two sections: a blue ramp and a red ramp. Thus, during the animation, pixels in blue (resp. red) in the Contextual Map will take different blue (resp. red) tones subsequently. The number of colors used in a Multimodal Motion Map is limited by the physical size of a color map. Remember that only 6 or 8 different colors are generally enough to define a suitable pattern in the Motion Map. Thus, with a standard color map of 256 colors, we can compute a 8-frames cyclic animation with 32 different colors, i.e. 32 different values for the scalar data. The Contextual Map has generally more than 32 distinct colors though. Thus, the first step of the Multimodal Motion Map computation aims at reducing the number of colors of the Contextual Map.

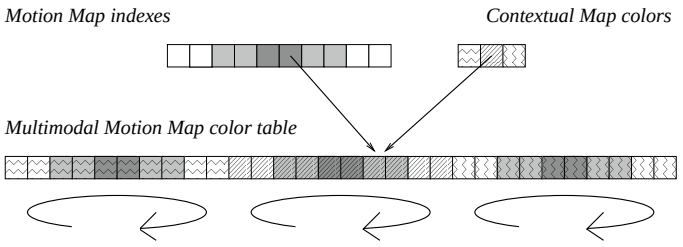

Fig. 9. **Principle of the method.** Contextual Map colors are patterned instead of being painted in order to show the merging with the Motion Map grey tints on this sheet.

The principle of the method is illustrated on Figure 9. It consists in extracting a number of base colors (let us call them *tones*) from the Contextual Map and generating a color ramp (let us call these colors *shades*) for each tone. Each color ramp is used to fill a different section of the Multimodal Motion Map color table. The main treatment consists in selecting the right set of tones and computing shades so that the total number of colors fits the desired size of the color map. The algorithm is given below:

Set number of tones and number of colors per tone
Convert the Contextual Map so that it has only the desired number of tones (quantization)
Compute a color ramp for each tone
Merge quantized Contextual Map and Motion Map into the Multimodal Motion Map

In order to visualize the contextual information as accurately as possible, the selection of tones and shades should be the most representative one with respect to the Contextual Map. We use Heckbert's median-cut algorithm to quantize the Contextual Map [30]. This method selects an optimal set of t colors to code the image, maps the existing colors to the new ones, and returns the quantized image.

B.2 Computation of Shading Sequences

The usual RGB color space provides a convenient additive definition of colors. Nonetheless, the notion of lightness is not directly represented, which prevents from shading colors with ease. The HLS color space better fits our needs. It is a double hexacone, as show on Figure 10. The angle around the vertical axis of the hexacone represents the hue (H), lightness (L) is represented on the vertical axis (0 for black and 1 for white), and saturation (S) is measured radially from those vertical axis (0 to 1). The main asset of the system is its structure since shading a tone can be achieved by directly varying a single component: lightness. However, the shape of the system requires to take care when varying L. The HLS hexacone is not uniform (colors spaced at equal perceptual distances are not equally spaced in L) [31]. Indeed, the lightness range decreases as the saturation increases. The range is null when S reaches its maximal value S_{max}. In order to allow the shading for each tone, we may extend the variation to the two other components.

Several thresholds must be set to ensure acceptable shades for a tone, which we call T_h, T_l and T_s. They define a sub-space within which the shading occurs. Since it is not uniform, the lightness range magnitude L_{range} is set in respect to the value of S: the magnitude is chosen inversely proportional to the value of S. In addition, to prevent very small magnitude (remember we stated on section III-B that patterns should be as contrasted as possible), we compel L_{range} to be a fixed minimal range if T_s is at a distance inferior to a certain threshold from S_{max}.

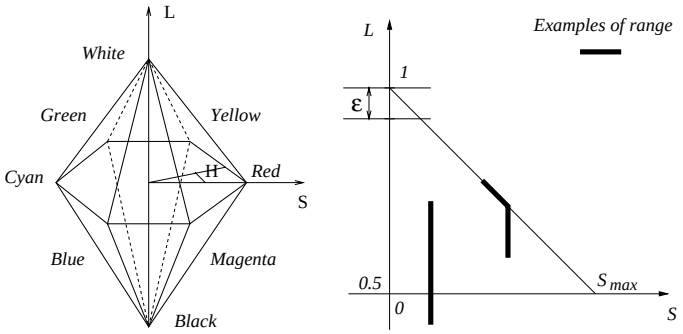

Fig. 10. Left: the HLS color space. Right: an upper section in profile.

With this magnitude definition, the range is centered on the tone color. The value of T_l does not always permit the variation of the lightness for the given range. Therefore, when a shade reaches a maximal (resp. minimal) possible value, we decrease S and keep L as high (resp. low) as possible. This comes down to choosing the shades on the hexacone sides (see Figure 10). An exception must be done if T_l is less than ϵ far from 0 or 1 (i.e. almost white or black tones) because neither L nor S can much vary. In this case, we vary H over the whole hexacone. The shades are then slightly perceptible because lightness is extreme (very high or very low).

B.3 Maps Combination

The combination of the Contextual Map and the Motion Map consists in generating the color map with the computed shades and computing the new indexes for the Multimodal Motion Map. The first step is trivial, the RGB values of the shades computed at the previous step are written in the color map, section by section. When all sections are the same size, the formula to compute the color index v of a pixel in the Multimodal Motion Map, as a function of a tone index i in the quantized Contextual Map and an index j in the Motion Map is:

$$v = s \times i + j$$

where s is the size of a section of the color map.

Although the formula to compute v differs, the principle works for sections of different widths as well. In

particular, not uniform section widths allow the treatment of fixed colors (for which section width is set to 1). Fixed colors correspond to regions that should not undergo the animation (borders, obstacles, text, captions, ...).

B.4 Real Time Animation

The *Motion Map Viewer* has been extended to implement a multi-sections shifting algorithm, the context-free case described in section IV being treated as a particular case in which there is only one section. The information on the structure of the Motion-Map is encoded in the image itself. The content of the color map is part of GIF, the number and sizes of the sections are stored in the comment part of the header section of the GIF format.

VI. Results and Discussion

In this section we compare our approach with Spot Noise and LIC. Sections VI-A and VI-B demonstrate the advantages of the method described in this paper by considering the qualitative and quantitative aspects, respectively, while section VI-C shows applications of this technique to wind datasets.

A. Qualitative assessment

Here we focus on the visual quality of the animations produced by our method, as compared to other approaches. With the Motion Map, we are able to produce variable-speed and perfectly cyclic animations of an arbitrary number of frames, the only limit being the actual size of the color table. Consecutive frames are perfectly correlated together, which yields to really smooth animations. An important feature of this approach is that several representations of a vector field can be obtained with a single Motion Map, just by changing the color table content. The content of the color table determines the appearance of the patterns which are moving during the animation. Particularly, using a sparse function allows to produce particle animations, with two important advantages over traditional particle systems: there is no need for computing particle paths in real time since moving the particles is achieved by shifting the color table entries, and particles are uniformly distributed over the domain at any time. Figure 11 shows three animations obtained with the same Motion Map. With the Motion Map technique, an additional scalar field can be visualized during the animation. This is a very important property because it allows precise localization of the flow features. For instance, if a satellite image of the physical domain is visualized underneath the flow, one can localize the region affected by a cyclone. A drawback of this approach is the necessary quantization of the scalar image. There are two arguments in favour of our approach though. Firstly, the loss of image quality is largely offset by the gain provided by animation. Second, we can assume that there is no need for high accuracy in the scalar field visualization during the animation. Hence a rough visualization of the scalar data is generally suitable in this context.

Now let us point on the advantages of this method over Spot Noise and LIC. It has been established that both techniques are similar in many respects [17]. For both methods, spatial correlation occurs along portions of streamlines of approximately the same length, which corresponds to the length of the spots and the size of the convolution filter kernel, respectively. This length is usually said to be about 20 pixels. A problem is that correlating the pixels leads to a loss of contrast because the longer the spots (resp. the convolution mask), the lower the contrast. Moreover, in their more successfully completed versions, as described in [32] and [33], respectively, both Spot Noise and LIC use a blending technique to achieve temporal correlation between consecutive frames, which yields also to a loss of image contrast. Indeed contours of the patterns that move in the flow appear as blurred, thus decreasing the spatial resolution. The consequence is that the finest details are not clearly visualized. With our technique we have a full control on the contrast, by determining the content of the color table, and no blending is performed, each pixel being computed separately. Last, we obtain a large and continuous dynamic range of velocities for variable-speed animations, as shown on Figure 3, whereas the dynamic range of animated LIC is narrow and available velocities are roughly quantized when cyclic animation is achieved [11], [13]. An interesting issue to investigate concerns the number of distinct velocities that the human eye is able to perceive in such a context.

Up to now we have considered using the Motion Map technique for animating textured images. Actually it can be used to animate sparse streamline-based images as well. Not covered pixels are assigned the background color and the flow is moving only along computed streamlines. Such a feature is not available with Spot Noise and LIC because they are not based on streamlines.

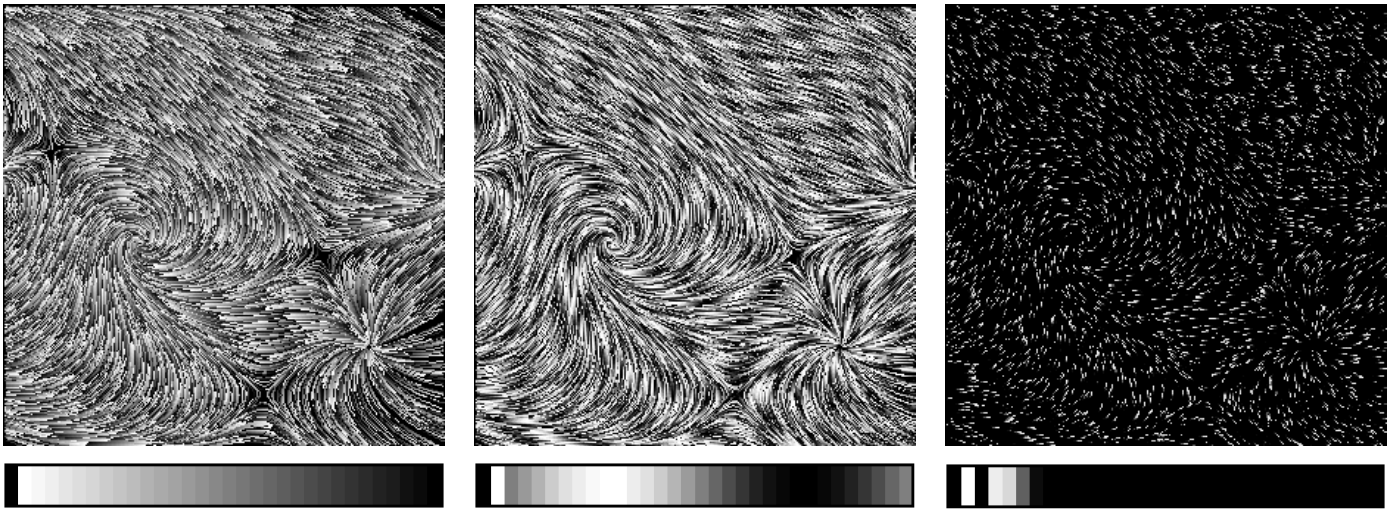

Fig. 11. **Various possible color table contents for a single Motion Map.** The images above have been obtained with a single Motion Map with different contents for the color table, which allows to produce various animation effects. Left: sawtooth function. Middle: sinusoidal function. Right: "sparse" function.

B. Quantitative assessment

Here we focus on the computation time and memory requirements of our method, as compared to Spot Noise and LIC. The memory necessary to store a Motion Map is the same as for a single static image of the vector field. It means that we are able to store an animation of the flow at the same cost as a static image, whereas the information contained in an animation is strongly increased because direction and magnitude of the flow are better visualized. By comparison Spot Noise and LIC require to store all frames separately.

The Motion Map Viewer is able to play the animation directly from the Motion Map using the color table animation technique, thus real time animation of the flow can be obtained even on low-end workstations. When a direct manipulation of the color table is impossible, for instance in a Web browser, we have developed a Java applet version of the Viewer, which is able to produce a cyclic set of textures and to encode them in an animated GIF, so that it can be played by any common Web browser. An interesting application is online visualization of vector fields. By publishing a Motion Map together with the applet Viewer, an animation of the flow can be downloaded at the cost of a static image, and visualized through any Web browser.

The Motion Maps showed on Figures 2 left and 4 left have a resolution of 512×512 and 900×900, respectively. The computation times obtained on a 150MHz MIPS R4400-32Mo SGI workstation are 7 and 14 seconds, respectively. As a comparison, a single LIC image of 512×512 with a filter kernel of 40 pixels needs approximately 6 seconds to be computed on the same machine [13], and computing an animation with LIC requires that all frames be computed independently, which yields $6 \times N$ seconds for a N frames animations. The comparison goes for Spot Noise as well.

C. Applications

We have used the Motion Map technique to visualize the results of a wind simulation over Europe. The input data are a vector field and a map of Europe obtained from remote sensing data (see Figure 12). The Motion Map uses 16 indexes. The original Contextual Map was composed of 216 distinct colors, which have been reduced to 6 by the quantization process. The color map used for the final animation has 96 entries, divided in 6 sections of 16 colors each.

L_{range} can also be parameterized in order to obtain special visualization effects. For instance we can set L_{range} as a function of the flow vorticity, so that regions of high vorticity will be more contrasted than the others, which would emphasize them during the animation.

Producing a cyclic set of textures can be useful for visualizing a vector field at the vicinity of a 3D object. Figure 13 shows a visualization of the results of a wind simulation over the Earth.

VII. Conclusion

Animation is necessary to visualize direction and velocity of a vector field properly. Understanding the

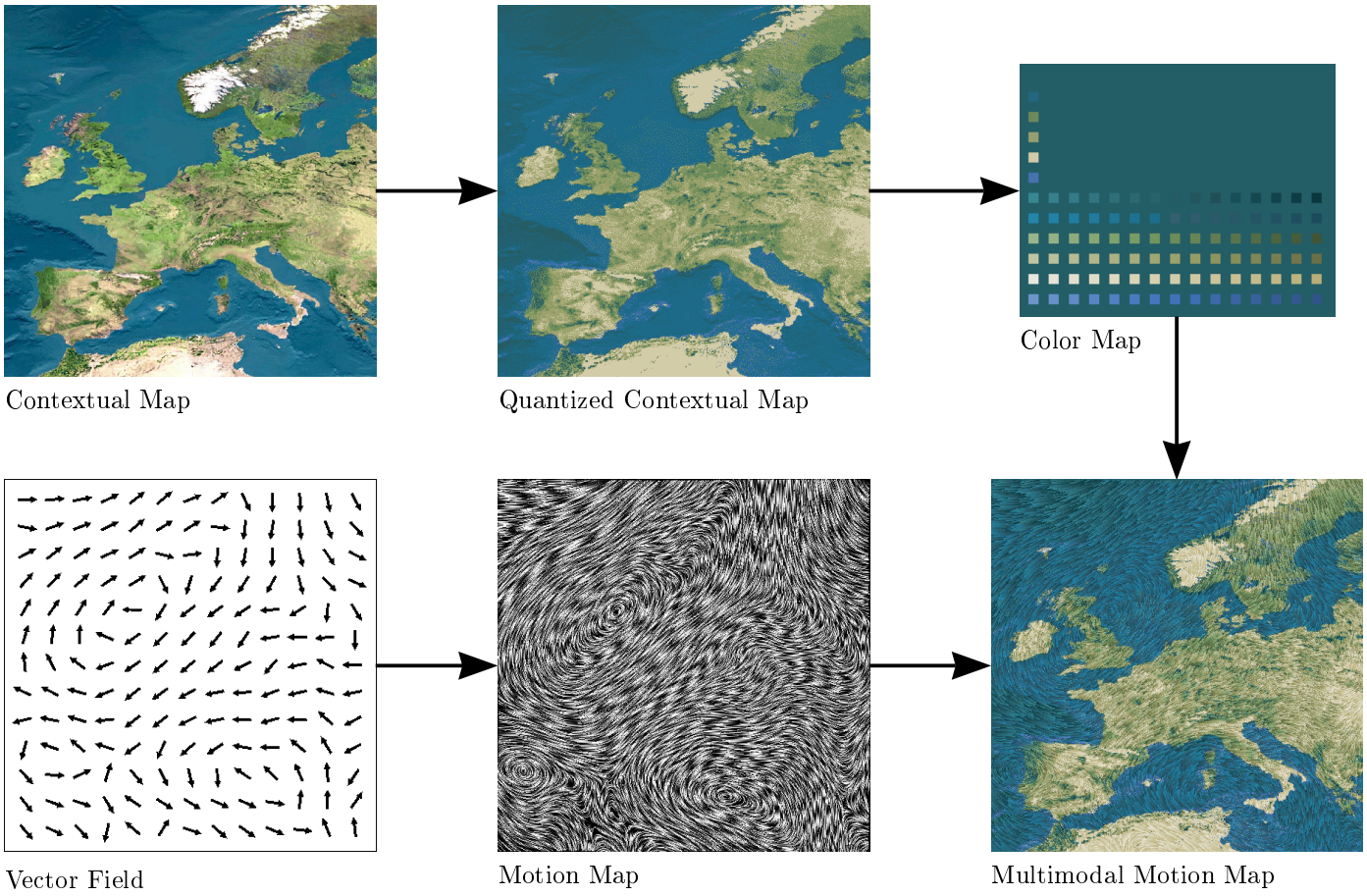

Fig. 12. **Visualization of wind simulation over Europe.** A vector field resulting from a wind simulation is animated over a satellite view of Europe. The Contextual Map is quantized to select the best set of tones while a Motion Map is computed from the vector field. Then our algorithm computes a color table composed of sections of shades corresponding to the colors of the Quantized Motion Map, and a Multimodal Motion Map is computed, which uses that color table. The Multimodal Motion Map is encoded in GIF and can be animated in real time with the Motion Map Viewer.

behavior of a complex dynamic system requires generally not only to visualize its dynamic features but also additional information, such as scalar data.

The Motion Map technique is a suitable solution to the issue of animating steady 2D vector fields. As far as computation time and memory requirements are concerned, this technique proves to be as effective as other techniques when they are used to compute a single image. Moreover an additional scalar field can be visualized throughout the animation. Animations can be generated in a really efficient way by using the color table animation technique, which allows to obtain real time animation of complex vector fields on any low-end workstation. The compactness of the Motion Map, together with the applet Viewer, makes it possible to explore large vector data sets remotely, and to visualize them through any Web browser.

Today only steady 2D vector fields can be visualized with the Motion Map technique. The principle of using the color table animation technique prevents from using the Motion Map for the visualization of unsteady vector fields. But there is probably some potential investigation to do in the direction of visualizing 3D flow fields using a variant of this technique. Actually computing a 3D Motion Map is not a problem but rather only a generalization of the algorithm to the 3D case. The main issue is to visualize such an animated 3D texture effectively.

Acknowledgements

We would like to thank Jean-Marc Pierson, who developed the applet viewer, Robert van Liere and Willem de Leeuw for the vector field data set for wind simulation over Europe, and the Canadian Meteorological Centre for the data set on wind predictions over the Earth.

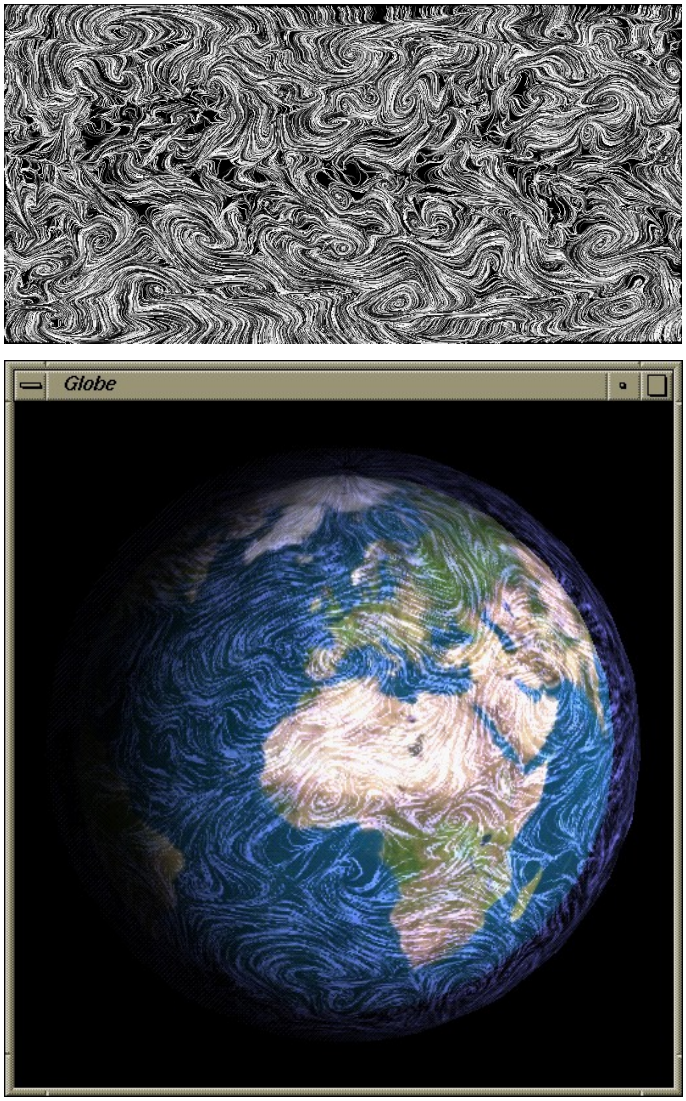

Fig. 13. **Visualization of wind predictions over the Globe.** Top: the Motion Map computed from the original vector data. Bottom: a cyclic set of correlated textures is generated from the Motion Map and visualized at the vicinity of a 3D model of the Earth, onto which a satellite image has been mapped. The Motion Map is visualized using transparency, as described in section V-A.

References

[1] Bruno Jobard and Wilfrid Lefer, "The motion map: Efficient computation of steady flow animations," in *Proc. of IEEE Visualization '97, Phoenix, Arizona, Oct 19-24, 1997*, Roni Yagel and Hans Hagen, Eds. oct 1997, pp. 323–328, IEEE Press, Los Alamitos, CA.

[2] Claire Chédot and Wilfrid Lefer, "Multi-modal flow animation with the motion map," in *Proc. of IEEE Visualization '98 Late Breaking Hot Topics, Research Triangle Park, North Carolina, Oct 19-24, 1998*, Amitabh Varshney and Craig Wittenbrink, Eds. oct 1998, IEEE Press, Los Alamitos, CA.

[3] Jarke J. van Wijk, "Spot noise: Texture synthesis for data visualization," *Computer Graphics*, vol. 25, no. 4, pp. 309–318, jul 1991, Proc. of SIGGRAPH '91.

[4] Brian Cabral and Leith Leedom, "Imaging vector fields using line integral convolution," *Computer Graphics*, pp. 263–272, aug 1993, Proc. of SIGGRAPH '93.

[5] Greg Turk and David Banks, "Image-guided streamline placement," *Computer Graphics*, pp. 453–460, aug 1996, Proc. of SIGGRAPH '96.

[6] Bruno Jobard and Wilfrid Lefer, "Creating evenly-spaced streamlines of arbitrary density," in *Visualization in Scientific Computing*, Wilfrid Lefer and Michel Grave, Eds. 1997, Focus on Computer Graphics, pp. 43–55, Springer-Wien-New-York, Proc. of Eighth Eurographics Workshop on Visualization in Scientific Computing, Boulogne sur Mer, Apr 28-30, 1997.

[7] J. Helman and L. Hesselink, "Visualizing vector field topology in fluid flows," *IEEE Computer Graphics and Applications*, vol. 11, no. 3, pp. 36–46, may 1991.

[8] Theo van Walsum, Fritz H. Post, Deborah Silver, and Frank J. Post, "Feature extraction and iconic visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, no. 2, pp. 111–119, jun 1996.

[9] Xiaoyang Mao, Yuji Hatanaka, Hidenori Higashida, and Atsumi Imamiya, "Image-guided streamline placement on curvilinear grid surfaces," in *Proc. of Visualization '98*, David Ebert, Hans Hagen, and Holly Rushmeier, Eds. oct 1998, pp. 135–142, IEEE Press, Los Alamitos, CA.

[10] Wim de Leeuv and Jarke van Wijk, "Enhanced spot noise for vector field visualization," in *Proc. of Visualization '95*, Gregory M. Nielson and Deborah Silver, Eds. oct 1995, pp. 233–239, IEEE Press, Los Alamitos, CA.

[11] Lisa K. Forsell and Scott D. Cohen, "Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 2, pp. 133–141, jun 1995.

[12] Lisa K. Forsell, "Visualizing flow over curvilinear grid surfaces using line integral convolution," in *Proc. of Visualization '94*, Daniel Bergeron and Arie E. Kaufman, Eds. oct 1994, pp. 240–247, IEEE Press, Los Alamitos, CA.

[13] Detlev Stalling and Hand-Christian Hege, "Fast and resolution independent line integral convolution," *Computer Graphics*, pp. 249–256, aug 1995, Proc. of SIGGRAPH '95.

[14] A. Okada and D. Lane, "Enhanced line integral convolution with flow feature detection," Tech. Rep. NAS-96-007, NAS, jun 1996.

[15] H-W. Shen, Chris Johnson, and Kwan-Liu Ma, "Visualizing vector fields using line integral convolution and dye advection," in *Proceedings of the Symposium on Volume Visualization '96*, 1996, pp. 63–70.

[16] M-H. Kiu and D.C. Banks, "Multy-frequency noise for lic," in *Proc. of Visualization '96*, Roni Yagel and Gregory M. Nielson, Eds. oct 1996, pp. 121–126, IEEE Press, Los Alamitos, CA.

[17] Wim de Leeuv and Robert van Liere, "Comparing lic and spot noise," in *Proc. of Visualization '98*, Hans Hagen and Holly Rushmeier, Eds. oct 1998, pp. 359–365, IEEE Press, Los Alamitos, CA.

[18] L. Yaeger, C. Upson, and R. Myers, "Combining physical and visual simulation - creation of the planet jupiter for the film 2̃010;" *Computer Graphics*, vol. 20, no. 4, pp. 85–93, jul 1986, Proc. of SIGGRAPH '86.

[19] Jarke J. van Wijk, "A raster graphics approach to flow

visualization," in *Proc. of Eurographics'00*, David Duce and Carlo Vandoni, Eds., 1990, pp. 251–259.

[20] Nelson Max and Barry Becker, "Flow visualization using moving textures," in *Proc. of ICASE/LaRC Symposium on Visualizing Time-Varying Data, Williamsburg, Virginia*, David C. Banks, Tom W. Crockett, and Ken Stacy, Eds., sep 1995, pp. 77–87, NASA Conference publication 3321.

[21] William T. Freeman, Edward H. Adelson, and David J. Heeger, "Motion without movement," *Computer Graphics*, vol. 25, no. 4, pp. 27–30, jul 1991, Proc. of SIGGRAPH '91.

[22] D. Verma, D. Kao, and A. Pang, "Bridging the gap between streamlines and lic," in *Proc. of Visualization '99*, David Ebert, Markus Gross, and Bernd Hamann, Eds. oct 1999, pp. 341–348, IEEE Press, Los Alamitos, CA.

[23] H-W. Shen and D. Kao, "Uflic: a line integral convolution algorithm for visualizing unsteady flows," in *Proc. of IEEE Visualization '97, Phoenix, Arizona, Oct 19-24, 1997*, Roni Yagel and Hans Hagen, Eds. oct 1997, pp. 317–322, IEEE Press, Los Alamitos, CA.

[24] Bruno Jobard and Wilfrid Lefer, "Unsteady flow visualization by animating evenly spaced streamlines," *Computer Graphics Forum*, vol. 19, no. 3, pp. 31–39, aug 2000.

[25] Richard Shoup, "Color table animation," *Computer Graphics*, vol. 13, pp. 8–13, aug 1979, Proc. of SIGGRAPH '79.

[26] Allen Van Gelder and Jane Wilhelms, "Interactive visualization of flow fields," in *Proc. of Workshop on Volume Visualization*. 1992, pp. 47–54, ACM.

[27] Malte Zöckler, Detlev Stalling, and Hans-Christian Hege, "Parallel line integral convolution," in *Proc. of First Eurographics Workshop on Parallel Graphics and Visualization*, Alan Chalmers, Ed. sep 1996, pp. 111–125, αlpha Books.

[28] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I - Nonstiff Problems*, Springer Verlag, 1993.

[29] Jack E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965.

[30] Paul Heckbert, "Color image quantization for frame buffer display," *Computer Graphics*, vol. 16, pp. 297–307, 1982, Proc. of SIGGRAPH '82.

[31] James Foley, Adries van Dam, Steven Feiner, and J. Hughes, *Computer Graphics - Principles and Practice*, Addison-Wesley, second edition, 1990.

[32] Wim de Leeuv, *Presentation and Exploration of Flow Data*, Ph.D. thesis, Technical University of Delft, The Netherlands, 1997.

[33] Hand-Christian Hege and Detlev Stalling, "Lic: Acceleration, animation and zoom," in *Texture Synthesis with Line Integral Convolution (course notes of SIGGRAPH'97)*, 1997, pp. 17–49.

Wilfrid Lefer is a Professor at the University of Pau, France. He received a PhD in computer science from the University of Caen, France, in 1992, and an Habilitation à Diriger les Recherches from the University of Littoral Côte d'Opale, France, in 2000. Lefer has founded the Visualization group at the University of Littoral Côte d'Opale, France. He organized and chaired the Eight Eurographics Workshop on Visualization in Scientific Computing, in Boulogne sur Mer (France), in 1997, and the First Workshop on Distributed Visualization Systems, at Research Triangle Park, North Carolina, in 1998. He is an expert evaluator for the Fifth Work Programme of the European Union. His research interests include flow visualization, and distributed and Web-based visualization systems. He is a member of the ACM, the IEEE Computer Society, and the Eurographics Association.

Bruno Jobard received a PhD in computer science from the University of Littoral Côte d'Opale, France in 2000. He was a postdoctoral fellow at Florida State University, from August 1999 to March 2001. He holds the position of researcher at the Swiss Center for Scientific Computing. His research interets concern flow visualization algorithms.

Claire Chédot received a PhD in computer science from the University of Provence at Marseille, France, in 1997. She is currently an assistant professor at the University of Littoral Côte d'Opale, France. Her research interests include flow visualization and multivariate visualization.