The Motion Map: Efficient Computation of Steady Flow Animations*

Bruno Jobard Wilfrid Lefer

Laboratoire d'Informatique du Littoral†
Calais, France

Abstract

This paper presents a new approach for animating 2D steady flow fields. It is based on an original data structure called the Motion Map. The Motion Map contains not only a dense representation of the flow field but also all the motion information required to animate the flow. An important feature of this method is that it allows, in a natural way, cyclical variable-speed animations. As far as efficiency is concerned, the advantage of this method is that computing the Motion Map does not take more time than computing a single still image of the flow and the Motion Map has to be computed only once. Another advantage is that the memory requirements for a cyclical animation of an arbitrary number of frames amounts to the memory cost of a single still image.

1 Introduction

Effective visualization of flow fields is a difficult issue because direction and velocity are difficult to render. Efficient methods have been proposed to compute dense images of 2D flow fields [11][1][7] but direction and velocity of the flow are difficult to render in a still image and areas of high vorticity are generally not adequately rendered. Animating the field helps to better understand its topology, especially where the flow is highly turbulent. Velocity variations within the flow could also be depicted by way of animation.

Methods proposed to compute such an animation generally consist in computing a cycle of frames which are then played in order to produce the animation. Problems arise when perfectly cyclic and variable-speed animations have to be computed. The drawbacks of such methods rely both in the computation cost, since all the images have to be computed separately, and in the memory requirements which are related to the number of frames included in the sequence (although animation compression techniques can be used).

This paper presents a new approach for computing cyclical and variable-speed animations of 2D steady flow fields. It is based on an original data structure called the Motion Map. We show that computing the Motion Map amounts to compute a single still image of the flow field and that memory requirements to store an animation of an arbitrary number of frames amounts to the memory cost of a single still image.

The remainder of the paper is organized as follow. Section 2 is a state of the art on related work. Section 3 gives an overview of our method while section 4 explains how to include motion information in the streamline representation. Section 5 describes the Motion Map structure and how it is built and section 6 shows how the computed animation sequence is played. Section 7 concludes the paper.

*This work has been supported by the Council of the Nord-Pas-de-Calais Region

†Laboratoire d'informatique du Littoral, BP 719, 62228 Calais Cedex. http://www-lil.univ-littoral.fr/∼lefer/visc.html

2 Related Work

Max and Becker proposed a texture advection technique which warps a texture alongside the flow [8]. A grid is embedded in the field, each grid point beeing assigned a texture coordinate. The motion effect is obtained by changing the texture coordinates from a frame to the next. A fading up and down mechanism allows to limit texture distortions.

A number of papers have been published on animating line integral convolution (LIC) textures [1][2][3][10]. A LIC texture is generated by convoluting an input texture with a streamline-oriented one dimensional filter kernel. Each pixel of the final texture is computed separately. To create cyclical animated sequences of LIC textures, periodic filters have been used [4]. This method is suitable for constant-speed animation. Making the frequency of the filter function varying could achieve variable-speed animation, but the dynamic range of discernable speeds is drastically limited. To solve this problem, Forsell has proposed to modulate the rate of the function phase shift as a function of the local vector magnitude [2][3]. A set of constant-speed LIC textures is pre-computed 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 pre-computed frames (that is the frames whose filter kernel phase values bound the value of the current frame). Detlev and Stalling noticed this approach does not yield periodic sequences any more and that applying kernel phases to neighboring pixels would lose any correlation [10], introducing spatio-temporal aliasing effects. As a consequence, the texture may appear to move in the opposite direction in some cases. To avoid this, to achieve cyclical variable speed animation they only use frames where filter kernel phases are correlated. Then a frame blending method is used to produce a periodic sequence of N frames by composing a set of $2N$ LIC textures.

Van Gelder and Wilhelms used the color table animation technique to animate 3D flow fields [5]. Particle paths are integrated and drawn in 3D as tubes in which colored particles seem to move. The tubes have a length equal to the integration step distance and are compounded with a fixed number of segments (1, 3 or 12). Within each of these segments the velocity is treated as constant so that a linear interpolation of color indexes can be done. The tubes are randomly "seeded" proportionally to the cells' weight and rendered as opaque using Z-buffer.

The method presented in the remaining sections allows creating 2D textures in a more natural way than LIC-based approaches. Real-time animation is achieve by way of either color table animation or cyclical textures animation.

3 Algorithm Overview

The basic principle of our method consists in computing a dense coverage of the image with a set of streamlines. But instead of convoluting a white noise texture as it is usually performed when computing a still image, streamlines are colored in such a way all the motion information, in particular direction and magnitude of the flow, is encoded within the streamlines, further allowing us to play an effective animation of the flow. Our approach relies on an important property of steady flow fields: the topology and magnitude distribution of the flow is constant over time. Thus the streamline placement, once computed, remains valid for all frames. We exploit this property and propose a new concept and data structure called the *Motion Map* to store a cyclical variable-speed animation of the flow field of an arbitrary number of frames. The method to compute the Motion Map proceeds as follow. The first part of the method consists in computing a dense coverage of the image by a set of streamlines. The second part uses a method based on the color table animation technique in order to color the streamlines and thus adding the motion information to the Motion Map. Actually both parts are executed subsequently each time a new streamline is computed. Due to the random election of new starting points for streamline integration (usually called *seed points*), the algorithm has to be stopped when a certain coverage percentage of the whole image has been obtained. Remaining sections explain the different part of the method.

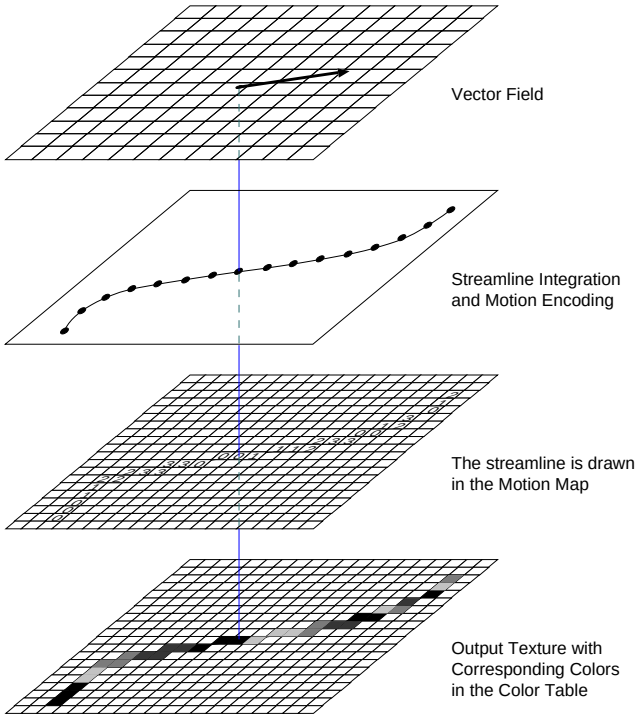

Figure 1: **From vector field to output frame.** A streamline is integrated in the vector field, motion is added to it and then the streamline is drawn in a Motion Map. To create the output texture the Motion Map indexes are replaced with the associated colors.

4 Encoding Motion within the Streamlines Representation

The method uses the color table animation technique [9]. In this technique, each pixel of the image is assigned an entry (also called an index) in the color table. If the color table entries are cyclically shifted together, one place at a time, a motion effect appears on the image. But a smooth animation requires each pixel to be well correlated to the next. The main advantage if this technique is that the image content has not to be updated while playing the animation but only the color table, allowing real time display on low performance graphic devices. Last, let us remark that, due to the cyclical shift on the color table, the maximum number of distinct frames in the animation sequence is given by the size (number of entries) of the color table.

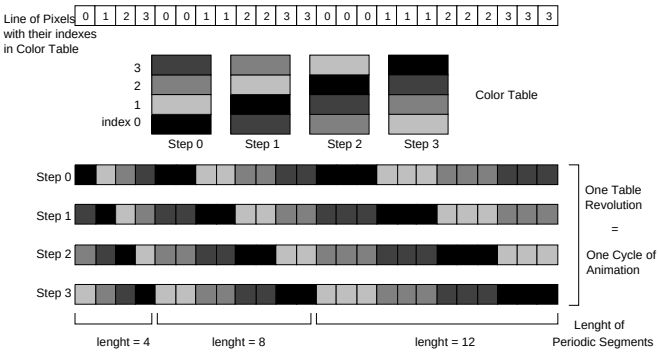

Figure 2: Variable speed in the color table animation technique.

It is possible to define different speeds in the animation by assigning consecutive pixels the same color table index. For instance figure 2 shows a line of pixels and the associated color table. Spans of identical pixels are longer on the right of the line. When color table entries will be shifted to the right, the flow will seem to speed up as it moves to the right. The speed of the flow is proportional to the number of pixels in the span.

The aim of our motion encoding algorithm is to compute such a sequence of color table indexes as a function of the flow velocities for each computed streamline. Each streamline is computed as a serie of consecutive evenly spaced sample points (see section 5 for details on the streamline computation method). Color table entries are assigned to the streamline sample points rather than to the pixels covered by the streamline because an approximation occurs during rasterization and therefore computing the motion before approximation will give more accurate results.

The formula used to compute color table indexes as a function of the flow velocities is given by Equation 1:

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

Where N is the number of distinct colors in the color table (i.e. the number of frames in the cyclical animation), P_i is the current sample point, (S_{P_i}) is an increasing serie in $\mathbb{R}^+$, $k(\|V_{P_i}\|)$ is the instantaneous contribution of the velocity at P_i and $Index_{P_i}$ is the color table index. When processing a streamline we start by the first sample point and then subsequent sample points are processed in order. $k(V_{P_i})$ is a linear 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 allow us to parameterize the visualization we introduce two new variables: L_{min} is the minimum number of sample points on which a complete revolution of the color table could be mapped and R is the ratio between the minimum and maximum speeds in the final image.L_{min} has been introduced in order to obtain a good correlation between consecutive pixels and to avoid anti-aliasing artifacts. Figure 3 illustrates the visual results obtained with different values of L_{min} with a constant speed vector field. We obtain the better visual results with $5 \leq L_{min} \leq \frac{1}{4} image\_size$. R has been introduced in order to set the speed range in the final animation. Indeed it could be interesting, in some cases, to increase (or decrease) the speed range to better show the flow field properties. To obtain the actual velocity range (that is to keep the original proportion between the different velocities in the flow field), we just have to set $R = V_{min}/V_{max}$. For instance to set the ratio between the maximum and minimum speed in the final image to 3, we just have to set $R = 1/3$.
Since k is a linear function of the velocity, we can write:

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

To determine a and b we used the limit conditions:

$$\begin{cases} k(V_{min}) = \frac{1}{L_{min}} \\ k(V_{max}) = R \times k(V_{min}) \end{cases}$$

Which yields:

$$\begin{cases} a = \frac{R-1}{L_{min} \times (V_{max} - V_{min})} \\ b = \frac{V_{max} - R \times V_{min}}{L_{min} \times (V_{max} - V_{min})} \end{cases}$$

Let us notice that for a constant speed animation $R = 1$ and thus $a = 0$ and $b = \frac{1}{L_{min}}$.

With this method, all the sample points on a streamline (and i.e. all pixels covered by the streamline) are strongly correlated together. Figure 4 shows the large and continuous dynamic range allowed with our index allocation method with a horizontal flow of increasing speed. Figure 5 shows a frame of a variable speed animation with its corresponding velocity map.

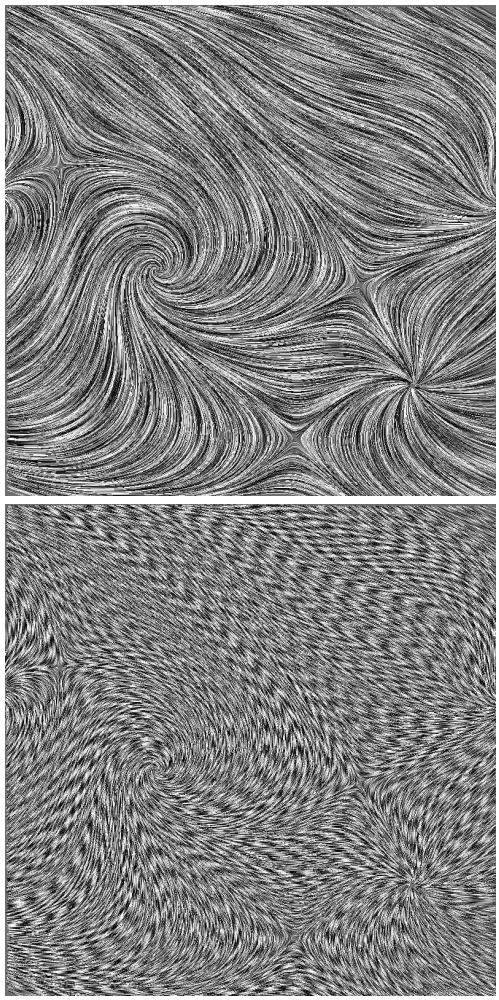

Figure 3: **Influence of $\mathbf{L_{min}}$.** (top) $L_{min} = 50$, (bottom) $L_{min} = 10$. Criteria for flow field textures evaluation differ depending on whether still images or animated sequences are concerned. Thus the top texture depicts more accurately the flow features in a still image whereas the bottom texture is more suitable for animation. Both images has been computed with $R = 1$, that is a constant speed over the field.

5 The Motion Map

Now we introduce the data structure which enables us to generate, at low cost, the N frames cyclical animation of the flow. We call *Motion Map* a bidimensional array of the same size the final image and in which streamlines will be drawn. The two next sections explain the seed point selection and streamline computation algorithms which are used to compute the Motion Map.

5.1 Seed Point Selection

Several seed point selection techniques have been proposed, especially for the computation of LIC images, such as selecting points in the scanline order, dividing images into blocks or using Sobol quasi random sequences [12]. We use the following algorithm. Each time a new streamline has to be computed, a cell is randomly elected as a new candidate for starting the future streamline. If the location is already covered by another streamline, we choose a direction (among up, down, left and right) and the next cell in this direction become the new candidate. If it is not free, the next cell in the same direction is tested and so on until either a free location is found or the boundary of the Motion Map has been reached. In the second case, a new cell is randomly elected and the selection process is executed one more time. When a valid seed point has been found, its location is no longer free and a global counter of the number of free cells in the Motion Map is decreased.

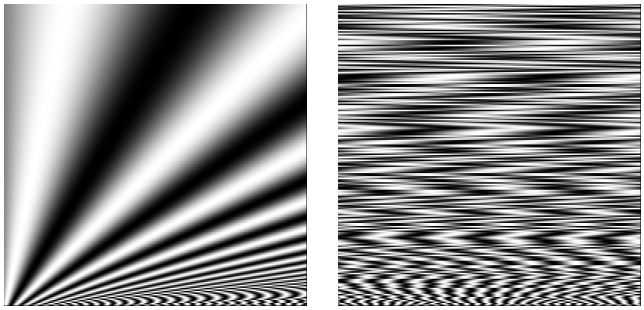

Figure 4: **Dynamic range.** Animation of a horizontal shear flow. In both images we set the ratio between maximum and minimum speed to 25 ($R = 1/25$) and L_{min} to 20. Left: $S_{P_0} = 0$, right: $S_{P_0} = random(N)$.

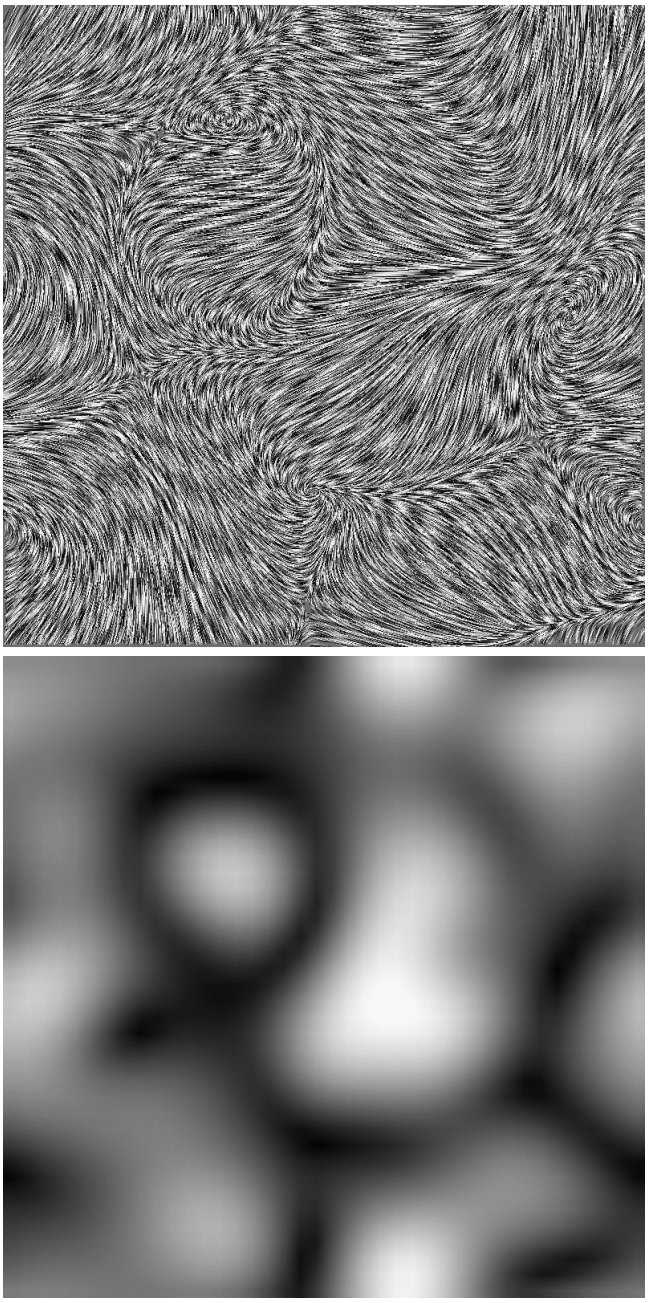

Figure 5: **Variable Speed**. Top image: frame of a variable speed animation (R=6 and $L_{min} = 5$). Bottom image: magnitude distribution, areas of low magnitude (dark) corresponds to small "waves" in the top image.

5.2 Computing Streamlines and Filling the Motion Map

In order to obtain a smooth animation, the consecutive sample points within a streamline have to be evenly spaced. Many integrators can produce such evenly spaced sequences of sample points [10]. The constant distance between two consecutive sample points is set to be equal to one or two pixel.
The streamline integration is stopped when it reaches an edge of the Motion Map or a singularity of the flow (source or sink) or becomes too close to another streamline. To validate this last stop condition, a simple and effective test consists to verify if the current sample point hits a non-free cell in the Motion Map. Best results has been obtained when we allow a streamline to hit several non-free cells before stopping the integration process. As a consequence, we obtain longer streamlines, which yields more correlation between subsequent pixels.
Once a streamline is computed and motion added to it, as described in section 4, the streamline is drawn as a polyline in the Motion Map using Bresenham's algorithm [6]. The "color" of each segment is the index which was assigned to the first sample point of each segment (see Figure 6).

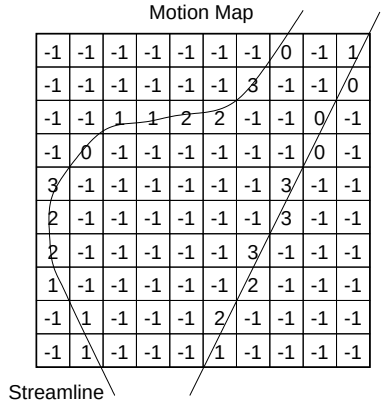

Figure 6: Drawing streamlines in a Motion Map.

As the Motion Map coverage increases, the number of free cells covered by each new streamline tends to decrease, making the convergence toward the complete filling of the Motion Map slowing down (see Figure 7). Fortunately it is not really a problem because the last free cells are randomly isolated over the map. Furthermore, affecting an arbitrary index to those free cells can just introduce spatio-temporal aliasing since they are not correlated with their neighborhood. For these reason the algorithm terminates when the number of free cells in the Motion Map falls under a certain threshold. In fact, before the beginning of the method, all the Motion Map is initialized with neutral value (-1 for instance) corresponding to a non-animated index of the color table. We allow generally between 3% and 5% of unset pixels in the final image. Also we noticed a quicker convergence is reached when the number of allowed hits of non-free cells during streamline integration is greater (see Figure 7).

Another remark concerns the length of the streamlines which tends to decrease as the coverage of the Motion Map increases. In order to maintain a good correlation along each streamline, the new streamlines are drawn "behind" previously computed ones.

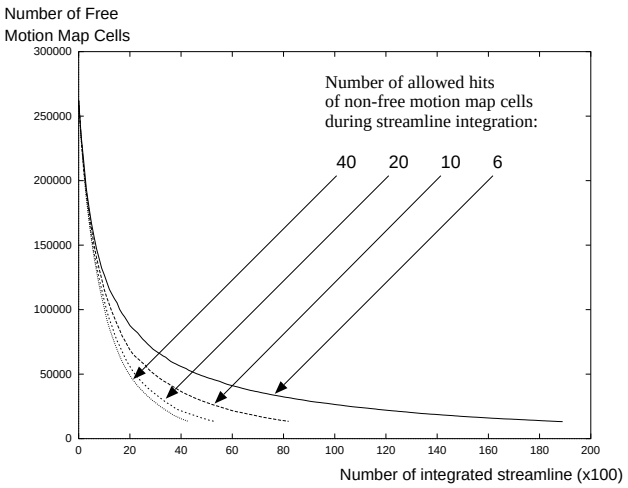

Figure 7: Convergence toward the complete filling of the motion map. The four plots was obtained with different numbers of allowed hits during streamline integration. These measures were made with a 512×512 image and the algorithm was stopped at a coverage percentage of 95%.

6 Playing the Animation

Once we have built the Motion Map, we have to generate an animation. In the two next sections, we presents two ways to achieve cyclical animation derived from our motion map. The first one uses the simple but effective *color table animation*, the second exposes how to obtain high quality *cyclical set of textures*.

6.1 Color Table Animation

Color table animation of the flow field is a direct and effective way to visualize the flow using the Motion Map. The extra work required at this stage is to fill the color table.

As described in [5], in addition to the directional information we can visualize a scalar quantity, such as a density or an energy. To do it we divide the color table in several distinct sections, each section beeing assigned a base hue and lighting within each section ranging from dark to bright. To find the index of a given pixel, we apply the same algorithm as explained earlier assuming the color table is one section large and then we add an offset which is determined by the value of the scalar variable at the same point. In this case the colormap cannot be shifted globally but an independant shift must be performed on each section. This is not a problem however, it just involves a more general shifting algorithm which takes into account the number of sections in the color table (1 if no additional variable has to be visualized). With a standard color map of 256 colors, we can compute a 16-frames cyclical animation with 16 different possible values for the additional scalar field (using 16 levels of brightness, the brightness coding the motion).

The Motion Map can be saved together with the color table as a GIF file for instance, thus enabling common viewers to display a still image of the flow field. Assuming we have a viewer able to cyclically shifting the color table indexes, the Motion Map is the more efficient way to store an animation of an arbitrary number of frames since the size of the animation is equal to the size of a still image.

6.2 Cyclical Set of Textures

The Motion Map can also be exploited to produce a cyclical set of textures which could be used to make a movie or to be mapped on complex objects. Cycle of textures is also more suitable than color table animation for creating color animation without limiting the number of available colors. Now we explain how to derive N cyclical textures from the Motion Map and how to use the color hue to visualize an independent scalar variable.
Let M be the Motion Map, with $M(x,y) = i, i \in \{0 \dots N-1\}$ and let $f(i) : \{0; N-1\} \longrightarrow [0;1]$ be a discrete function which associates an intensity to each index in the Motion Map. This function will give the shape of the wave which seems to move on the streamline. Let C be a color map where $C_{r,g,b}(x,y) = (r,g,b)$ is the RGB color corresponding to the independent value at the location (x,y) in the Motion Map. The colored texture is obtained by multiplying colors with intensities:

$$T_{r,g,b}(x,y) = \begin{cases} f(M(x,y)) * C_{r,g,b}(x,y) & \forall M(x,y) \geq 0 \\ D_{r,g,b}(x,y) & \forall M(x,y) < 0 \end{cases}$$

Where $D(x,y)$ is the *default color* function which gives a RGB value for the locations where any steamline passed through (generally, $D_{r,g,b} = C_{r,g,b}$).

The N frames $T^0, ..., T^{N-1}$ of the cyclical animation are obtained by cyclically shifting the index values:

$$T^t_{r,g,b}(x,y) = \begin{cases} f(\,(M(x,y)+t) \ mod \ N) \\ \quad \times C_{r,g,b}(x,y) \\ D_{r,g,b}(x,y) \end{cases} \begin{bmatrix} t \in \{0;..;N-1\} \\ \forall M(x,y) \geq 0 \\ \forall M(x,y) < 0 \end{bmatrix}$$

Let notice that the number of available colors to represent the independent scalar value is not limited. Moreever we have noticed that the motion effect keeps very effective, even if color frames are reduced to black and white textures (black pixels for values below a given threshold, white pixels for values above).

6.3 Discussion and Results

The advantages of the Motion Map for the generation of cyclical animations are both quantitative and qualitative. Quantitatively, the memory requirements to store the Motion Map are not greater than for storing a still image of the flow (by comparison, LIC-based animations require to store all frames separately). This means that, instead of storing a still image of the flow, we are now able to store an animation at no extra cost but increasing the information contained in the data since direction and magnitude of the flow are better rendered in an animated sequence. Concerning computation requirements, Table 1 gives the computation times for two different textures. These values have been obtained on a 150 MHz MIPS R4400-32Mo based system. The algorithm terminated when a coverage of 95% was reached. This table shows that this method is not computationally expensive as compared to LIC-based approaches where all frames have to be computed independently.
Qualitatively, the method achieves a good correlation between consecutive pixels along a streamline, the last one beeing also correlated with the first one. Since the method uses the same streamlines for all frames, the frames are correlated together too.

Some animation examples and executable files can be found at the address: http:www-lil.univ-littoral.fr/∼jobard/MotionMap

Image		Computation times
Fig 3:	512×512	7 seconds
Fig 5:	900×900	14 seconds

Table 1: Computation times needed to create a cyclical variable-speed animation of an arbitrary number of frames.

7 Conclusion

We have presented a new approach for computing a cyclical and variable-speed animation of a steady flow field. It is based on an original data structure called the *Motion Map*. We showed that, both qualitatively and quantitatively, our method gives good results, especially when compared to LIC-based approaches. When accessing remote datasets through the net, the Motion Map could be downloaded at no additional cost as compared to a still image of the flow and thus does not require higher bandwidth. Thus, as far as remote access to steady flow fields is concerned, one can imagine to replace usual still thumbnails included in HTML pages (or in certain dataset file headers) by animated thumbnails which would be animated by a plugin or a helper able to shift the color table entries. The main drawback of our method is that it is limited to 2D steady flows and future works will address the issues of animating unsteady flows and 3D flows.

References

[1] Brian Cabral and Leith Leedom. Imaging vector fields using line integral convolution. *Computer Graphics*, pages 263–272, aug 1993.

[2] Lisa K. Forsell. Visualizing flow over curvilinear grid surfaces using line integral convolution. In *Proc. of Visualization '94*, pages 240–247. IEEE Press, Los Alamitos, CA, oct 1994.

[3] 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*, 1(2):133–141, jun 1995.

[4] William T. Freeman, Edward H. Adelson, and David J. Heeger. Motion without movement. *Computer Graphics*, 25(4):27–30, jul 1991.

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

[6] Donald Hearn and M. Pauline Baker. *Computer Graphics - Second Edition*. Prentice Hall International Editions, 1994.

[7] Bruno Jobard and Wilfrid Lefer. Creating evenly-spaced streamlines of arbitrary density. In Wilfrid Lefer and Michel Grave, editors, *Visualization in Scientific Computing*. Springer Wien, 1997.

[8] Nelson Max and Barry Becker. Flow visualization using moving textures. In *Proceedings of ICASE/LaRC Symposium on Visualizing Time-Varying Data*, 1995.

[9] Richard Shoup. Color table animation. *Computer Graphics*, 13:8–13, aug 1979.

[10] Detlev Stalling and Hand-Christian Hege. Fast and resolution independent line integral convolution. *Computer Graphics*, pages 249–256, aug 1995.

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

[12] Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. Parallel line integral convolution. In *Proceedings of First Eurographics Workshop on Parallel Graphics and Visualization*, pages 111–125. αlpha Books, sep 1996.