Creating Evenly-Spaced Streamlines of Arbitrary Density[⋆]

Bruno Jobard and Wilfrid Lefer

Laboratoire d'Informatique du Littoral, Calais, France
{jobard,lefer}@lil.univ-littoral.fr

Abstract. This paper presents a new evenly-spaced streamlines placement algorithm to visualize 2D steady flows. The main technical contribution of this work is to propose a single method to compute a wide variety of flow field images, ranging from texture-like to hand-drawing styles. Indeed the control of the density of the field is very easy since the user only needs to set the separating distance between adjacent streamlines, which is related to the overall density of the image. We show that our method produces images of a quality at least as good as other methods but that it is computationally less expensive and offers a better control on the rendering process.

Introduction

The problem of visualizing vector fields has been widely addressed in the past years because it has numerous applications. The main issue is to visualize properly the direction and magnitude of the flow. Spatial resolution techniques such as arrow plots, streamlines or particles traces suffer from their spatial resolution that limits drastically their usefulness, in particular in the presence of a turbulent flow. Moreover the effectiveness of the streamline and particle methods depends critically on the placement of the forming seed points. Texture-like methods, such as Spot Noise [10] and LIC [1] produce dense field images showing the flow features in fine-grain detail. Another issue is to compute sparse flow fields, laying stress on the visual appearance of the field, which produces hand-drawing style images. Recently, Turk and Banks presented an imaged-guided streamline placement to compute hand-drawing style representations of a flow field [9]. This paper presents another effective algorithm for the placement of evenly-spaced streamlines. The main technical contribution of this work is to propose a single method to compute a wide variety of flow field images, ranging from texture-like to hand-drawing styles. Indeed the control of the density of the field is very easy since the user only needs to set the separating distance between adjacent streamlines, which is related to the overall density of the image.

⋆ This work was supported by the council of the Nord-Pas-de-Calais region.

1 Related work

Visualizing a vector field in a general manner requires high spatial resolution techniques to properly render fine-grain details. Such methods generally yield dense field representations. However there are situations where a sparse density image is needed, by instance to produce an illustration similar to those used to enhance the purpose of text fields in a book. Methods proposed to visualize a flow field falls into these two categories: dense field representations and hand-drawing style.

The first method for representing a flow field with high spatial resolution has been proposed by van Wijk [10]. The *spot noise* method creates a directional texture by superimposing many flow-oriented ellipses. Each ellipse is generated by projecting a spherical spot onto a surface and by advecting the spot with the direction and magnitude of the vector field at the projection point. This amounts to the flow field-controlled generation of a band-limited noise. Initially straight, the spots are now bent along short streamlines to follow the curvature of the vector field [2]. An important feature of this method is the local control on the generated image. More spots gives rise to more accuracy. With spot noise the generation time depends on the number of spots used to generate the texture. Consequently, by setting the number of spots a trade-off can be obtained between image quality and rendering time.

Another interesting method is the *line integral convolution* (LIC) proposed by Cabral and Leedom [1] [4] [8]. A LIC texture is generated by convoluting an input texture with a streamline-oriented one dimensional filter kernel. The images obtained with this technique are very effectives, showing more details than the previously enumerated one. But this is obtained at the expense of computation. This computation cost is mainly due to the number of streamlines that have to be computed, and let us notice that even with the *fastLIC* method [8], several streamlines cover each single pixel of the resulting texture, giving rise to frequent recomputations.

The image-guided streamline placement method proposed in [9] uses a stochastic mechanism to iterately refine the placement of the streamlines. First an initial set of randomly placed streamlines is created. Then this set of streamlines is updated using three valid operations: (1) changing the position and/or length of a streamline, (2) joining streamlines that nearly abut, and (3) creating a new streamline to fill a gap. At each step of the refinement process a small change, i.e. a combination of the three operations mentioned above, is randomly performed. An energy function consults a low-pass-filtered (blurred) version of the image to measure the variation of energy between the current and the updated images and the modification is only accepted if the variation of energy if negative. This method produces high quality images but the convergence is very slow and obtaining a good visual appearance often requires several minutes for each image to be computed. Moreover this method is not suitable for dense field images because of the combinatorial explosion of the possible modifications at each step.

The remainder of this paper is organized as follows. In section 2 we present a method for effective user-controlled evenly-spaced streamlines placement. Section 3 describes the use of the method to produce hand-drawing style images and we compare our approach with the image-guided streamline placement from Turk and Banks. In section 4 we show how texture-like images can be obtained and we discuss the advantages and drawbacks of this method compared to LIC. We conclude and offer directions for future research in section 5.

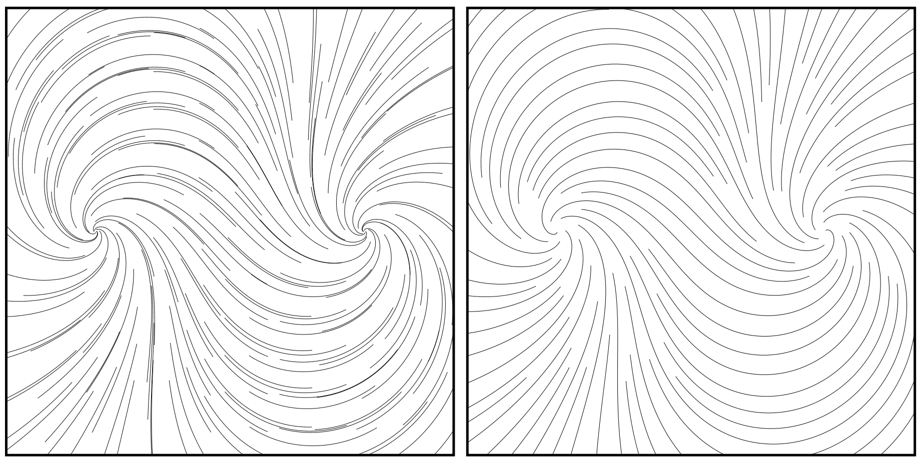

Fig. 1. (a) Long streamlines with seed points placed on a regular grid (left); (b) Same flow field computed using our streamline placement method (right)

2 Algorithm overview

The goal of this work is to produce long and evenly spaced streamlines in a single pass. The basic principle of our algorithm is similar to a method presented by Max et al. in [6]. The goal of their work was to cover a 3D surface (not necessary tangential to the field) with a set of streamlines. Once a seed point has been selected in the field, they make a streamline growing beyond that point backward and forward. The growing process is stopped when the streamline reaches an edge of the surface, a singularity in the field (source or sink) or becomes too close to another streamline. The streamline is then divided into a set of small segments of contrasting color and projected onto the surface. Although Max's method was intended to visualize a flow on a 3D surface, it can be generalized to all kinds of steady 2D field.
We extend this work in the following manner. First we give a number of precisions concerning the implementation of the algorithm together with a couple of optimizations. Second we show how the algorithm can be controlled by the user to produce a wide range of flow fields images, ranging from hand-drawing

to LIC-like style.

An important feature of the algorithm is that it processes in a single pass (as compared to Turk's progressive refinement approach). To compute an image, a number of streamlines are calculated until a user-fixed density level has been obtained. Computing a new streamline is achieved in the following manner. A new seed point is chosen at a minimal distance apart from all existing streamlines. Then a new streamline is integrated beyond the seed point backward and forward until either it goes too close from another streamlines or it leaves the 2D domain over which the computation takes place. The algorithm ends when no more valid seed point can be found. Figure 1b shows an image obtained with our algorithm and figure 1a shows the same flow field visualized using a distribution of the seed points over a regular grid. The three following sections detail three important points of the algorithm: the control of the distance between adjacent streamlines, seed points selection and streamline integration.

2.1 Control of the separating distance

Density is a global feature of the field. However we need to express it as a local feature in order to have a local control on the texture generation. We express the density as the distance between two adjacent streamlines. Let d_{sep} be this distance. Hence the control of the density of the field is achieved by controlling that there is not any pair of streamlines apart from a distance lower than d_{sep}. This control occurs during the construction of each streamline. During the construction, a new sample point is valid only if it is at a separating distance greater than d_{sep}. If it is not the case, the streamline is stopped in this direction (during construction streamlines grow in both directions independently). In order to make the computation of the separating distance faster, rather than computing the exact distance from the new seed point location to the streamline, we compute the distance from the seed point to the sample points along the streamline. To make this approximation acceptable, the sample points on a streamline must be evenly spaced and the distance between them must be smaller than d_{sep}. Thus, a new sample point is valid if the distance between it and all the existing sample points is greater than the separating distance. Since this test has to be computed for all the sample points, it must be as fast as possible. To accelerate the computation of the distance, we use a cartesian grid superposed to the vector field domain, the width and height of a cell being exactly d_{sep}. Each cell contains a list of pointers to the sample points located within the cell. Thus, given a new seed point location, the cell containing the location is easily determined. Let us call this cell the local cell. The distance has to be computed only for the sample points located either within the local cell or within the eight cells surrounding the local cell. In practice, we have noticed that an average of 5-7 distance computations is necessary to determine if a new sample point is valid or not. The denser is the grid, the less comparisons are required.

remark: Practically, we consider two important distances, d_{sep} and d_{test}. d_{sep} is the separating distance given by the user. It represents the minimal distance between seed points and streamlines. d_{test} is a percentage of d_{sep}. It corresponds to the minimal distance under which the integration of the streamline will be stopped in the current direction. We found $d_{test} = 0.5 \times d_{sep}$ gives good visual result by producing long streamlines. For instance, images of Figure 2 have been calculated with two different values of d_{test}.

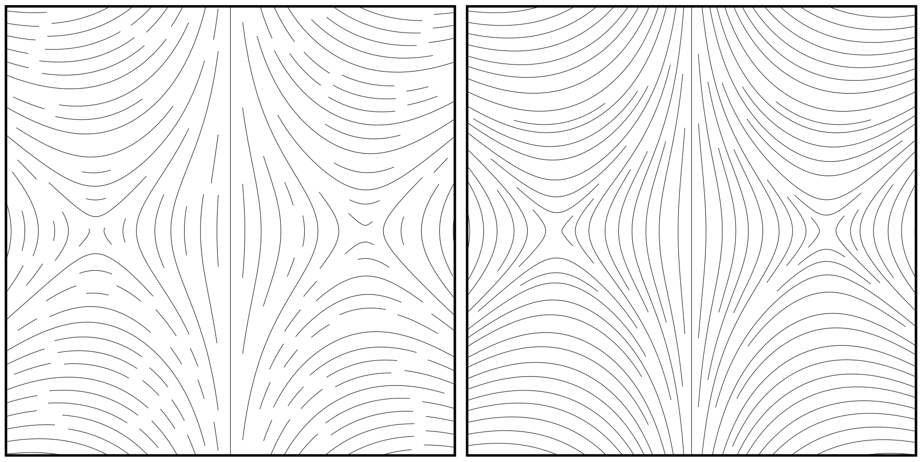

Fig. 2. increasing difference between d_{sep} and d_{test} lengthen streamlines; (left) $d_{1_{test}} = 0.9 \times d_{sep}$; (right) $d_{2_{test}} = 0.5 \times d_{sep}$

2.2 Seed points selection

When using streamlines for vector field visualization, a common problem is to select proper seed points for path tracking. Dovey proposed two approaches to resample non-uniformly spaced grids in order to achieve an uniform density of vector glyphs [3]. A vector field is represented with short segments oriented by the flow. In case of short streamlines or hedgehogs, the resulting image mainly depends on the distribution and density of the seed points over the domain. In case of long streamlines a constant density of seed points do not ensure a good distribution of the streamlines.
In order to obtain a good visual appearance of the flow field, an accurate seed point selection has to be perform. The principle of our algorithm is to derive all the seed points possible to find from an existing streamline before trying with another existing one. The proposed seed points are chosen at a distance $d = d_{sep}$ from the sample points of each streamline. Our algorithm uses a queue to store the newly created streamlines which are processed from the older one to the more recently created one. The algorithm is given below.

```
Compute an initial streamline and put it into the queue
Let this initial streamline be the current streamline
Finished := False
Repeat
    Repeat
        Select a candidate seedpoint at $d = d_{sep}$ apart from the current streamline
    Until the candidate is valid or there is no more available candidate
    If a valid candidate has been selected Then
        Compute a new streamline and put it into the queue
    Else
        If there is no more available streamline in the queue Then
            Finished := True
        Else
            Let the next streamline in the queue be the current streamline
        EndIf
    EndIf
Until Finished=True
```

Figure 3 shows all the streamlines the algorithm has been able to derive from the first streamline for a given vector field.

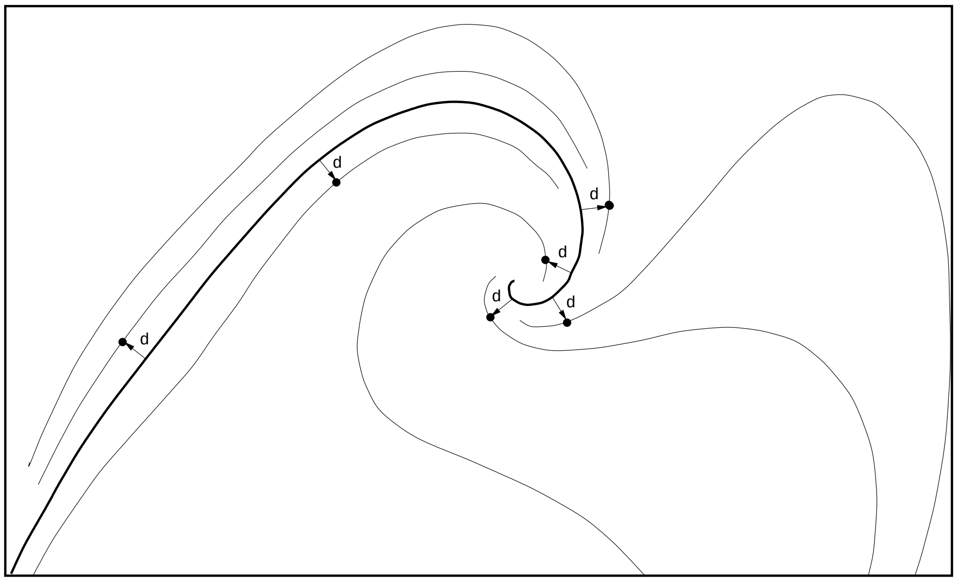

Fig. 3. streamlines are derived from the first (thick) one by choosing seed points (circles) at a distance d = d_{sep} from it

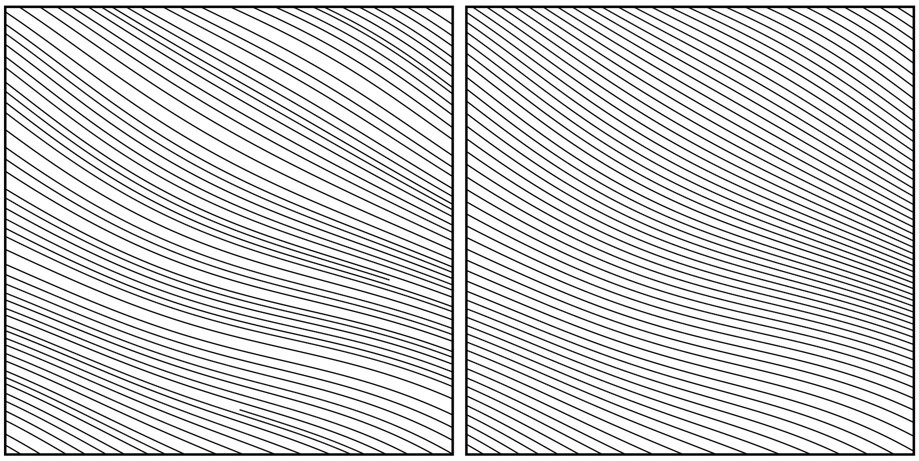

Fig. 4. Two seed points selection method with the same density of streamlines: (left) random selection, (right) our selection method

For sparse illustrations choosing to start streamlines close to existing ones gives better visual results than selecting seed points in a random fashion (see figure 4) but is more time-consuming. For dense texture generation, the quality of images produced with various seed point selection methods is quite similar.

2.3 Streamline integration

In order to measure a consistent separating distance between a point and a streamline, sample points along a streamline must be evenly spaced (see section 2.1). Many integrators are able to produce such evenly spaced sequences of sample points. They can be classified into three categories:

- fixed step size integrators such as Euler, Midpoint or Runge-Kutta methods with normalized vector fields,
- non constant or adaptive step size integrators with a post interpolation phase such as cubic Hermite-interpolation, which deal with large distances between neighboring sample points and curvature of the streamlines [8],
- continuous integration methods such as DOPRI5, which is a fifth order Runge-Kutta integrator with adaptive step size monitoring and fourth order error estimation and produces a dense output directly by using informations gathered at each step of the integration [5].

At present we use the Midpoint integrator but future investigations will concern the choice of a better integration method. In particular using an adaptive step size integrator will decrease the number of integrations required, reducing overall computation time.

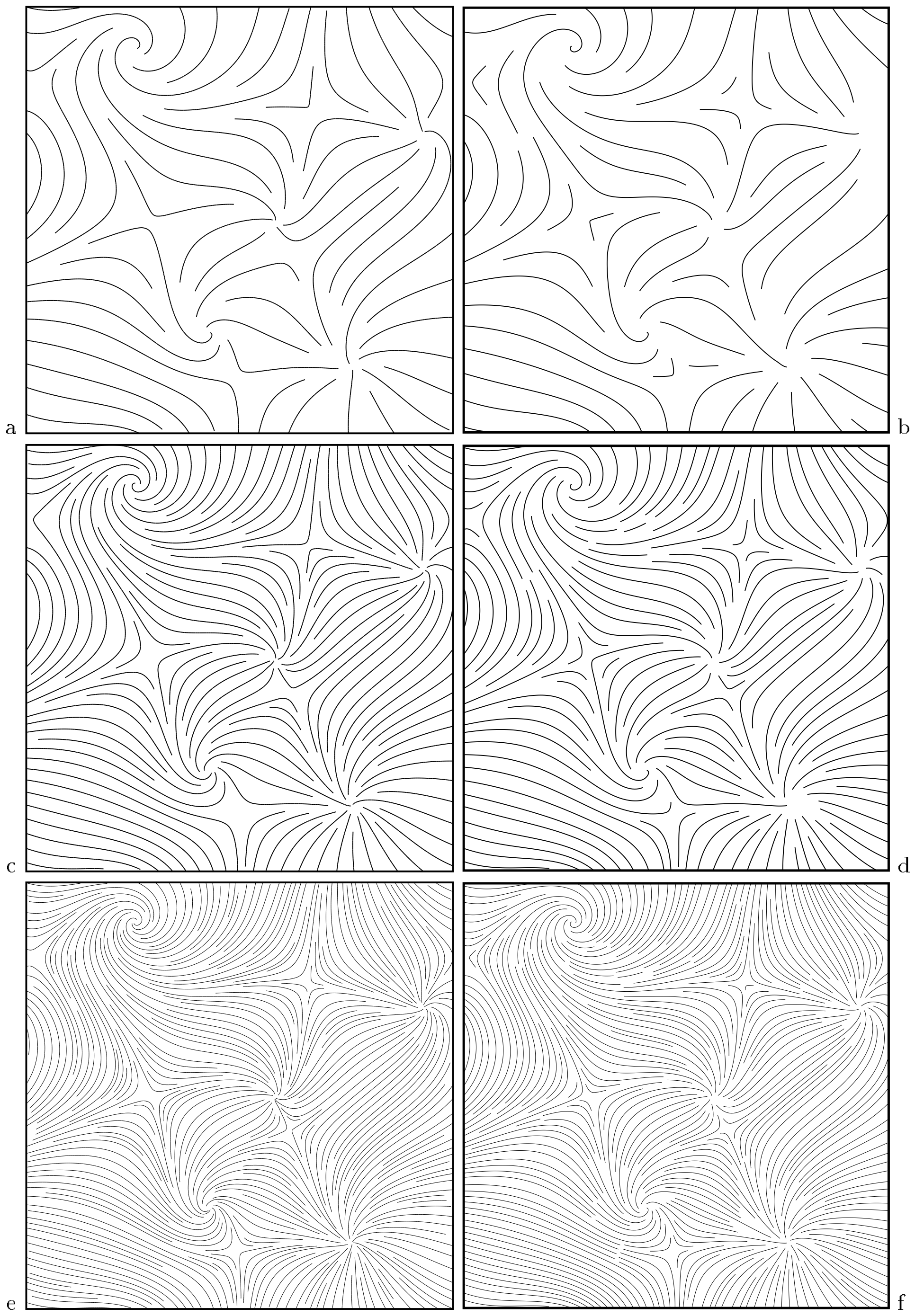

Fig. 5. Image comparisons for separating distances of 6%, 3% and 1.5% of image width; left column: Image-guided placement; right column: Our streamline placement.

3 Hand-drawing style

Sparse illustrations of flows fields are the more interesting application of our method. Turk et al. proposed a method to compute such a representation in [9]. The method computes an initial set of streamlines which is then iterately refined until the global energy of the image falls below a fixed threshold or the user stops the process. The images obtained with this method are of great quality but the convergence of the iterative process is very slow and becomes much slower when desired density increases. Moreover the energy function used to measure the quality of the result is not directly related to the visual appearance of the image, requiring the appreciation of the user to stop the process.
The advantage of our method as compared to Turk's one is that it processes in a single pass, computing the final image directly. Figure 5 shows the same flow field computed by Turk's method with various refinement degrees and by our method and table 1 gives computation times necessary to produce the different images. We see that our method produces images of the same quality as Turk's ones but is less time-consuming.

separating distance	**Image-guided placement**	**Our placement algorithm**
6% image width	Fig 5a: stopped at 2 mn	Fig 5b: 4 seconds
3% image width	Fig 5c: stopped at 4 mn	Fig 5d: 9 seconds
1.5% image width	Fig 5e: stopped at 10 mn	Fig 5f: 17 seconds

Table 1. Computation times on a MIPS R4600 Processor, 132Mhz with 32Mo. Image-guided placement images and computation times were obtained with the Greg Turk's original publicly available algorithm.

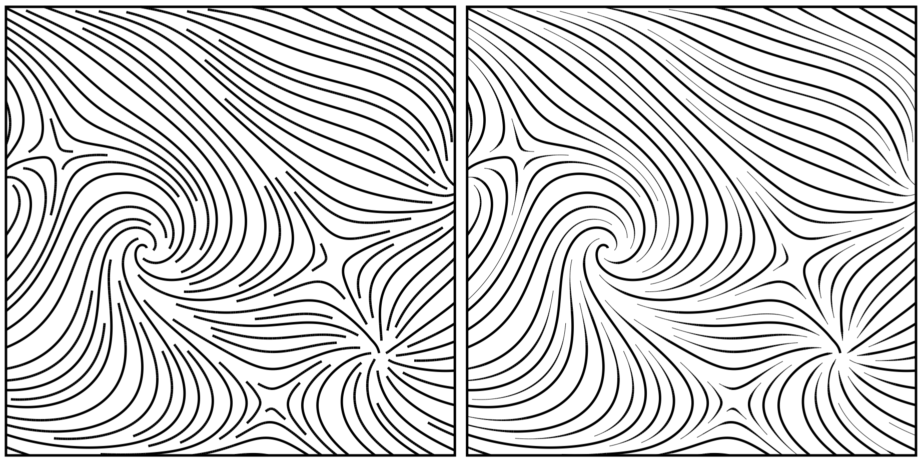

Fig. 6. Hand-drawing style images computed without and with the tapering effect

Tapering effect. As stated in section 2.1 the actual distance between streamlines is not constant. Since the density is related to the distance between streamlines, disparities of density may appear in the resulting image. To reduce this visual artifact, Turk suggested to *taper* the ends of the streamlines by decreasing the thickness of the lines as they go closer to another one. In case of Turk's algorithm, this is achieved in a post-processing step. Our implementation allows to directly include the tapering optimization during the growing process of each streamline. The width of the streamline is computed using the following formula:

$$thicknessCoef = \begin{cases} 1.0 & \forall d \geq d_{sep} \\ \dfrac{d - d_{test}}{d_{sep} - d_{test}} & \forall d < d_{sep} \end{cases} \; ; \; thicknessCoef \in [0;1]$$

where d is the distance to the closer streamline (see section 2.1 for the definition of the different distances).
Fig 6 shows the same image computed without and with the tapering effect.

Glyph mapping. Once the streamline placement has been computed, the streamlines can be viewed as *skeletons* on which directional glyphs can be mapped. Figure 7 shows an image obtained by mapping such kind of icons onto the computed streamlines. This enables to add a directional information in the field.

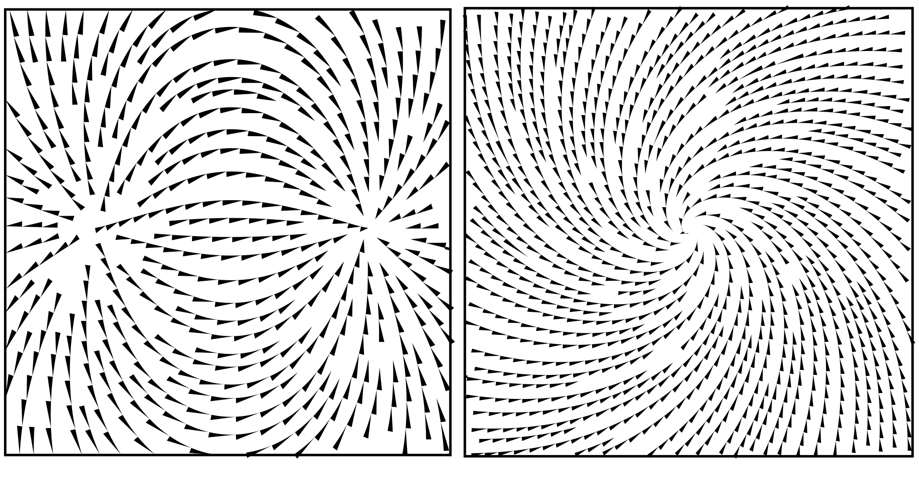

Fig. 7. examples of illustrations with glyph mapping

4 Texture generation

By decreasing the separating distance d_{sep}, the coverage of the streamlines becomes dense over the field. To depict the tangential component of the flow in a dense representation we have to differentiate close streamlines to be able to follow them over the field. This is achieved by mapping a periodic intensity function onto the streamline. Let us consider $f(x)$ a function which associates an intensity to every sample point on a streamline where x is the rank of the sample point within the streamline. f will give the shape of the intensity wave on the streamline. For instance, one may associate the two functions f_1 and f_2 given below:

$$f_1(x) = \frac{1}{2}(1 + sin(\frac{2\pi x}{N})) \quad and \quad f_2(x) = \frac{x \ mod \ N}{N-1}$$

where N is the length of the period as a number of sample points.

f_1 will give a smooth continuously increasing and decreasing intensity while the modulo function f_2 will produce discontinuous segments of increasing intensity in order to remove ambiguities about the orientation of the flow. Figures 8 and 9 show an image obtained using this pair of functions.

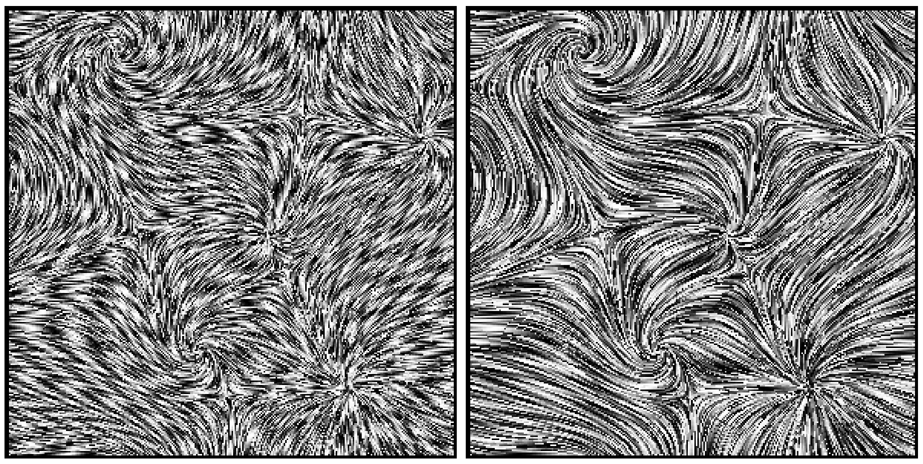

Fig. 8. textures generated with the f_1 function for intensity effects; (left) short and (right) long period of f_1

The images obtained with our method look like LIC images. In fact our algorithm is somehow a dual version of LIC. With LIC, for each pixel p of the output image, one integrates a particle path centered on p forward and backward and then averages intensities of the input texture pixels to get the intensity of the pixel being calculated. In our method, we find an optimal dense coverage of the field which minimizes the number of integrated streamlines and then associate an intensity to each pixel of all the streamlines.

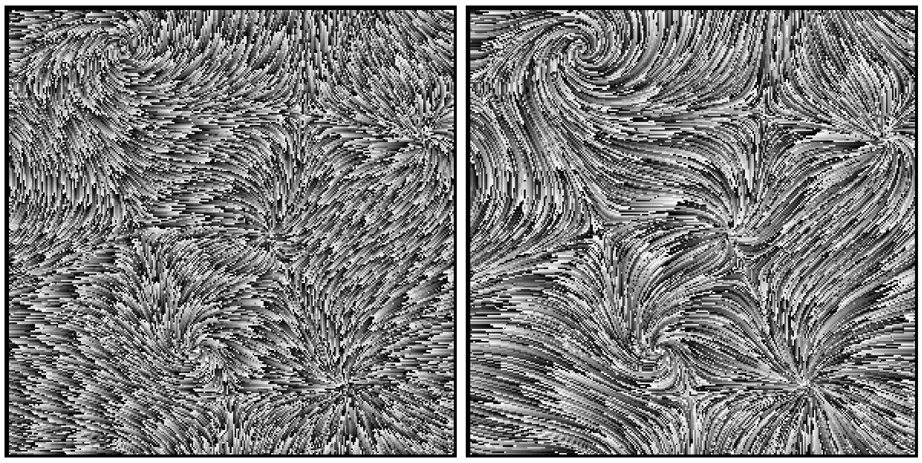

Fig. 9. textures generated with the f_2 function for intensity effects; (left) short and (right) long period of f_2

Now let us point on the advantages of our method over LIC. The first advantage is that our method does not require an input texture to process. With LIC a problem arises when one want to change the length of the apparent streamlines. In case of LIC, it is necessary to change the length of the convolution filter, which gives rise to a computation overhead. It is stated in [1] that doubling the length increases the computation time by a factor of four. Okada and Lane have proposed another solution, which consists in executing the LIC algorithm twice, the resulting image of the first execution being used as the input of the second execution [7]. This method results in lines which are more easily distinguishable but it concentrates the pixels values in a narrow range of intensity, which decreases the global contrast. This effect can be removed by applying post-filters to the final image, but this is done at the expense of a computation overhead (approximately by a factor of two).
With our approach we are able to change the length of the apparent streamlines on demand by simply changing the length of the period of the function as described above, without recomputation of any streamline.

In the traditional LIC algorithm, a streamline integration and a line convolution are computed for each individual pixels, so most of the time is spent in convolution and integration operations. Decreasing the number of streamlines computed would greatly benefit the LIC algorithm. Stalling and Hege proposed the *FastLIC* algorithm which reduces the overall number of streamline computations by sharing the line integral convolution information of each streamline with all the pixels it goes through [8].

With our approach, we compute the minimum number of streamlines necessary to cover entirely the 2D area over which the field is studied.
For instance, the computation of textures images of size 512×512 (such as figures 8 and 9) takes about 25 seconds on a R5000SC-64Mo based system. These results have been obtained using a random seed points selection method and a separating distance of 0.3%. As far as dense texture images are concerned, a drawback of our method is the aliasing effect due to the drawing of adjacent line segments of different colors. To remove this artifact we can smooth the final image by simply applying a blur filter.

5 Conclusion

We have presented an effective method to place long evenly-spaced streamlines with an accurate control on the density of the final image. By changing the separating distance between streamlines we are able to produce from sparse to dense representations of flow fields. We show that our method produces images of a quality at least as good as other methods but that it is computationally less expensive and offers a better control on the rendering process. Future investigations will concern a more efficient integrator, generalization to unsteady flows and real time animation.

References

1. Brian Cabral and L. Leedom. Imaging vector fields using line integral convolution. *Computer Graphics*, 27:263–272, jul 1993.
2. W. C. de Leeuv and Jarke J. van Wijk. Enhanced spot noise for vector field visualization. In *Proc. of Visualization '95*, pages 233–239. IEEE Press, Los Alamitos, CA, oct 1995.
3. Don Dovey. Vector plots for irregular grids. In *Proc. of Visualization '95*, pages 233–239. IEEE Press, Los Alamitos, CA, oct 1995.
4. 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.
5. E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I - Nonstiff Problems*. Springer-Verlag, 1993.
6. Nelson Max, Roger Crawfils, and Charles Grant. Visualizing 3D velocity fields near contour surfaces. In *Proc. of Visualization '94*, pages 248–255. IEEE Press, Los Alamitos, CA, oct 1994.
7. A. Okada and D. Lane. Enhanced line integral convolution with flow feature detection. Technical Report NAS-96-007, NAS, jun 1996.
8. D. Stalling and H-C. Hege. Fast and resolution independent line integral convolution. *Computer Graphics*, 29:249–256, jul 1995.
9. Greg Turk and David Banks. Image-guided streamline placement. *Computer Graphics*, 30:453–460, jul 1996.
10. Jarke J. van Wijk. Spot noise: Texture synthesis for data visualization. *Computer Graphics*, 25(4):309–318, jul 1991.