ENHANCED VECTOR FIELD VISUALIZATION BY LOCAL CONTRAST ANALYSIS

A. Sanna B. Montrucchio C. Zunino P. Montuschi

Dipartimento di Automatica e Informatica
Politecnico di Torino, C.so Duca degli Abruzzi 24
I-10129 Torino
Italy
{sanna,montru,c.zunino,montuschi}@polito.it

ABSTRACT

Visualizing vector fields is one of the most important and attractive research areas in scientific visualization. Several techniques are known in the literature; some traditional approaches use 2D/3D arrows or particle traces, while other methodologies display vector fields by dense or sparse textures. This paper focuses on the clustering problem arising for dense texture algorithms where some low contrast areas can appear in images thus reducing the capability of investigating flow details. The proposed method assigns pixel colors according to a local contrast analysis phase. In this way streamlines denoting flow characteristics are always well distinguishable.

Keywords: vector field visualization, scientific visualization, texture based algorithms, LIC, local contrast.

1 INTRODUCTION

Texture based algorithms have been proved to be one of the most effective way to display vector fields. Almost all characteristics can be visualized showing direction, orientation, and magnitude of the flow; moreover, dense texture based methodologies use the pixel resolution in order to show as much as possible vector field details by providing an efficient tool for investigating and analyzing flow characteristics. Flow direction is denoted by integral curves having tangent vectors coincident to the vector field (streamlines) and information about orientation and magnitude can be coded either by varying gray tones along each streamline or using colors. The colors (or gray levels) of the pixels along a streamline should be highly correlated in order to show the field line denoted by the streamline itself and the correlation among the pixels placed along directions perpendicular to the field lines should be low.

One of the main drawbacks of several dense texture based algorithms known in the literature, is to assign a starting gray value in a random way and then varying this value along the streamline to code orientation and flow. This can lead to visual *artifacts* or *clustering*; if similar values are assigned to neighbor streamlines, low contrast areas (also called macro-structures) can appear in the image. In these zones the field details are difficult to be distinguished and the quality of the resulting texture can be also strongly affected.

In this paper a new technique based on the analysis of the local contrast is proposed. We start from a dense texture based algorithm called TOSL (Thick Oriented Streamline) [Montru01] able to show all flow characteristics but suffering of clustering problem. After having performed an analysis phase of pixels close to the one under computation. In this way, neighbor streamlines are depicted by different tones and all details can be well appreciated.

Section 2 presents the main texture based algorithms and, in particular, reviews the TOSL algorithm used as a starting point for the proposed methodology. Section 3 explains the basic idea behind this work, while the details of the algorithm are found in Section 4. Finally, examples and results are presented in Section 5.

2 BACKGROUND

Texture based methods improve spatial resolution up to the pixel limit (dense textures). Van Wijk [Wijk91] proposed to convolve a random (white noise) texture along a straight segment whose orientation is parallel to the direction of the flow. This method (spot noise) was then extended by bending spot noise, filtering the image to cut low frequency components, and using graphic hardware methods, also on grids with irregular cell sizes (De Leeuw and Van Wijk [Delee95]). Cabral and Leedom [Cabra93] introduced the Line Integral Convolution (LIC) algorithm, which locally filters a white noise input texture along a streamline. Forssell [Forse94] extended the LIC to curvilinear grid surfaces, by doing calculations in computational space on a regular cartesian grid, and displaying results on curvilinear grids in physical space. As LIC is computationally expensive, Stalling and Hege [Stall95] improved the speed of LIC (fastLIC) by more than ten times, by observing that the LIC value computed for one pixel can be re-used, with small modifications, from its neighbor pixels; in this way, the computation is streamline oriented and not pixel oriented as in the conventional LIC. Zöckler et al. [Zockl97] showed a parallel implementation of fastLIC which is able to run in real-time on particular parallel architectures (i.e. on a Cray T3D). Bi-dimensional LIC images can be animated to show the orientation of the field besides the simple direction; this can be done by changing shape and location of the filter kernel k over time. To avoid the need for animation, Wegenkittl et al. [Wegen97a] introduced OLIC (Oriented Line Integral Convolution) and then Wegenkittl and Gröller [Wegen97b] FROLIC (Fast Rendering OLIC). OLIC simulates the use of drops of ink smeared to the underlying vector field. The algorithm can be made faster by positioning small and overlapping disks (FROLIC) in order to simulate the convolution. The length of the pixel traces shows vector orientation and local magnitude of the field. However, OLIC and FROLIC use sparse textures, and therefore, small details of the field may be lost in the visualization. A fast implementation of a LIC-like algorithm can be also found in [Risque98], where each field line is drawn by using a different gray value (the same value along the whole streamline). However, because of the constant gray value, this method cannot show orientation and magnitude of the field and it is not suitable for animation. Visualization of dense and oriented flow field is also performed by Jobard and Lefer [Jobar97]. The minimization of the number of the streamlines is performed by an evenly-spacing algorithm able to produce a good quality image; however the use of this algorithm slows down the speed in comparison to fastLIC. The vector field visualization can be achieved also using furlike textures (Khouas et al. [Khoua99]); the results are similar to FROLIC, by sparse textures. An extension to the LIC algorithm able to visualize unsteady flow data was proposed by Forssell and Cohen [Forse95]. Shen and Kao [Shen98] improved the algorithm overcoming the problems of coherence associated with the pathline convolution (with rather unsteady flows) and other drawbacks due to the difficulties of establishing temporal coherence on the pathline; this new algorithm (UFLIC) can manage LIC using the time-accurate value scattering and the successive feed-forward process. Although several algorithms can map on the texture the information of direction, orientation, and magnitude, adding scalar values such as temperature or pressure can be a problem; this issue has been addressed by Sanna et al. in [Sanna00a] and [Sanna01] by using the bump mapping technique to code scalar values by bumps and depressions.

2.1 TOSL ALGORITHM

In this section the TOSL algorithm [Montru01] is briefly reviewed since it is the starting point for the improvements introduced by the proposed methodology ECTOSL.

The TOSL algorithm has been proved to be one of the most effective algorithm representing vector fields by dense textures. It is able to show almost all flow characteristics: direction, orientation, and magnitude; moreover, examples have shown TOSL can be faster up to thirty times than LIC [Cabra93] and three time than fastLIC [Stall95]. The TOSL algorithm is split into two parts; during the first phase only a percentage (previously set by the user) of the image is filled, while the second part provides the filling of the output texture by computing streamlines for all the pixels not yet considered. A bidimensional Sobol sequence is used to identify a set of starting pixels for the streamlines to be computed in the first part, while in the second part the filling is provided considering the pixels in sequence (this is the only difference between the two phases). Chosen a starting pixel by using Sobol sequence, the streamline computation is performed in the same way as LIC; if the streamline in the point cannot be computed, for instance because the starting pixel is placed in a corner of the image and the vector field has opposite orientation with respect the field of the neighboring pixels, a random value is assigned to the pixel. In order to show the flow speed, for each pixel of the streamline, the local magnitude of the vector field is computed. The maximum value is used to normalize the increment of gray tones between a pixel and its successor along a streamline. A starting gray value is randomly set and next value along the streamline are computing adding the previous value to a normalized increment depending on the local magnitude. The choice of starting pixels is the most critical point of the algorithm; the use of a quasi-random Sobol sequence attempts to evenly spread streamlines over the texture to reduce the probability of *clustering*. Clustering occurs when neighbor streamlines have about the same gray value; this can lead to macro-structures in the texture where the vector field details cannot be appreciated; an example is shown in Fig. 1 where a set of macro-structures has been outlined by white rectangles. Clustering is basically due to the random choice of the gray value for the starting pixel and it produces very low contrast areas in the image; it is important to note this phenomena does not affect the whole contrast of the image, as it can be seen by the uniform distribution of gray tones in the histogram shown in

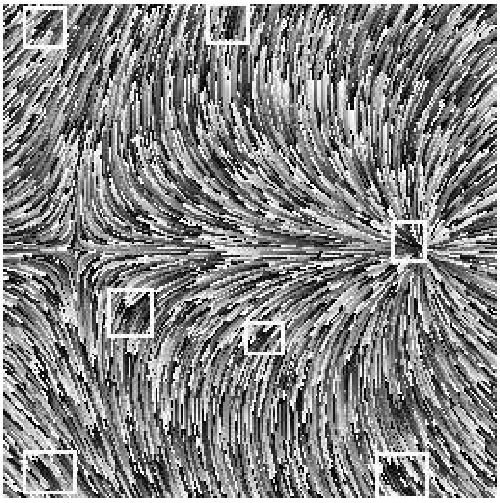

Figure 1: Example of a texture obtained by TOSL showing the clustering problem. White rectangles outline clustered areas.

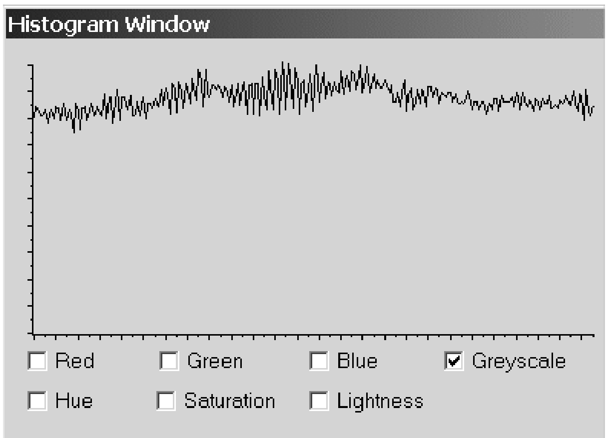

Figure 2: The global histogram of the picture shown in Fig. 1.

Fig. 2, but it concerns local properties of the texture.

3 BASIC IDEA

In order to allow users of identifying streamlines everywhere in a texture, low contrast areas should be strictly avoided. If the goal is to assign streamline colors in such a way every streamline could be always well distinguished from its neighbors, the color assignment should be performed according to a local contrast analysis phase and not in a random way. Contrast is defined by:

$$C = \frac{L_{max} - L_{min}}{L_{max} + L_{min}} \tag{1}$$

where L_{max} is the peak luminance and L_{min} is the minimum luminance [Ware00]. Given an area of n pixels with center the starting pixel of a streamline, the maximum local contrast is obtained when a gray tone maximizing the differences among it and all $n-1$ pixels of the area is chosen (i.e. it is maximized the numerator of (1)). As gray tones along a streamline are coded in the range `[0, 255]`, the value next `255` is `0`, therefore, we can assume to place the value on a circle (see Fig. 3). Let us assume to consider an area of two pixels: one value is in the range `[0, 127]` (p_1 in Fig. 3) and the other one is the starting pixel of the streamline to be computed. The maximum contrast is obtained by assigning to the starting pixel a value to the opposite of the circle, that is $p_1 + 128$. On the other hand, for values in the range `[128, 255]`, the maximum contrast is obtained subtracting `128` (or by adding modulo 128). For larger areas, average values for pixels in the range `[0, 127]` and `[128, 255]` have to be considered.

4 ECTOSL - ENHANCED CONTRAST TOSL

In this section we present the details of ECTOSL - Enhanced Contrast TOSL; ECTOSL computes streamline in the same way as TOSL and uses increments of gray tones to code local flow magnitude. As for TOSL, the streamline starting points are chosen by means of a bidimensional Sobol sequence, but it is completely different the technique for the

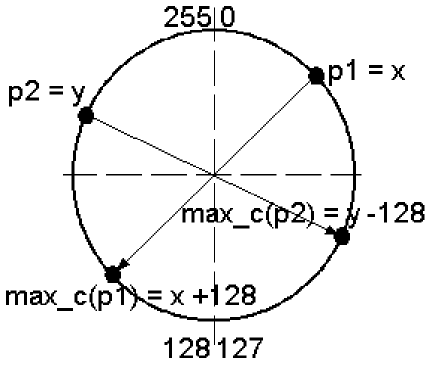

Figure 3: Given a pixel its opposite one the circle provides the maximum contrast.

choice of the starting gray value. TOSL assigns the starting tone in a random way and this can lead to artifacts as macro-structures. More complex is the technique used by ECTOSL where the starting value is assigned according to an analysis of the local contrast; a C-like pseudo code of the procedure for the choice is shown in Fig. 4.

The user has to set a parameter `d` denoting the distance used by the procedure to search for pixels already computed close to the one under analysis `(x,y)`. The variable `n_less` is used to store the number of pixels having gray tones in the range `]0,127]` within a square area with center in `(x,y)` and side `2d`; in the same way, `n_greater` stores the number of values in the range `[128,255]`. The value `0` denotes pixels not yet computed. On the other hand, `c_less` and `c_greater` are used to store the average value of gray tones in the range `]0,127]` and `[128,255]`, respectively. The coordinates `(x_start,y_start)` and `(x_stop,y_stop)` represent the vertices of a square area where the routine will search for pixels already computed; if either `x_start` or `y_start` are negative they will be set to zero as well as `x_stop` and `y_stop` are limited to the texture resolution. The search area is analyzed by two nested cycles and then the average values are computed for gray tones both in the range `]0,127]` and `[128,255]` (if `n_less` and `n_greater` are not null). This phase is repeated for every pixel along the streamline under analysis. In order to enhance the local contrast (see Section 3), `128` is added to the average value in the range `]0,127]` while is subtracted for the range `[128,255]`. If no pixel already computed has been found in the searching area, a random gray tone is assigned to the starting pixel of the streamline, otherwise, an average value is set. Section 5 shows how

```
choose_color(int d,int x,int y)
{
 c_less = c_greater = 0;
 n_less = n_greater = 0;
 x_start = x - d; x_stop = x + d;
 y_start = y -d;  y_stop = y + d;

 for each pixel of the streamline
  for(i=x_start;i<x_stop;i++)
   for(j=y_start;jy_stop;j++) {
     if(pixel(i,j) in ]0,127])
     {
      n_less++;c_less+=pixel(i,j);
     }
     if(pixel(i,j) in [127,255])
     {
      n_greater++;
      c_greater += pixel(i,j);
     }                        }
 if(n_less != 0)
 c_less = c_less/n_less + 128;
 if(n_greater != 0)
 c_greater=c_greater/n_greater-128;
 if(not found pixels)
   pixel = random_value;
 else
   pixel = ave(c_less,c_greater);
}
```

Figure 4: Procedure for the choice of the starting gray tone.

this analysis of the local contrast avoids the possibility of low contrast zones where streamlines could be not distinguishable. It is worth to note that this phase of choice of the starting gray tone level could be also applied to the other texture based algorithms (for instance, [Risque98]) in order to improve the local contrast.

5 EXAMPLES AND RESULTS

In this section we show three examples used to compare the proposed algorithm ECTOSL to TOSL [Montru01]. The first example (attractor) is shown in Figures 5 and 6; Fig. 5 has been obtained by TOSL and Fig. 6 by ECTOSL. Although TOSL texture is well suited to depict orientation, direction and magnitude of the flow, it can be noticed several areas where streamline color is almost the same (macrostructures). From the point of view of the global contrast this is not perceptible, in fact, by analyzing the histogram of the image all gray levels are evenly distributed but low contrast zones affect the texture. On the other hand, these areas strongly reduce the local contrast and the streamlines cannot be well identified. The texture produced by ECTOSL maintains the same characteristics in term of ability to denote direction, orientation and magnitude of the flow, moreover, each streamline is everywhere distinguishable from its neighbors and areas of uniform gray values are avoided. The same result is also shown by the other two examples: critical point (see Figures 7 and 8) and two points (see Figures 9 and 10). For all examples ECTOSL gives a better and faithful visual representation of vector fields. All textures have a resolution of 256x256 pixels and the streamline length has been set to 25 pixels. ECTOSL experienced to be slightly slower than TOSL as the local contrast analysis phase is necessary to assign the pixel color (see Section 4); computational times strongly depends on texture resolution [Montru01] and are almost similar for all three examples. By using a 400 MHz Celeron, TOSL textures have been computed in about 0.37 s, while ECTOSL ones in 0.5 s. Textures computed by ECTOSL have been obtained setting a value of distance `d` to one pixel; we performed several tests by varying the value `d` up to 5 but we did not experienced noticeable improvements, moreover, higher `d` values lead to greater computational times due to the local contrast analysis phase. Interested readers can test TOSL, ECTOSL, as well as LIC [Cabra93] at: `www.hipeco.polito.it/wscg2002/`; examples presented in this paper are available, moreover, users can use their own test files in order to verify algorithm behavior. More than textures, it is possible to see classical 2D arrow vector field representations.

6 CONCLUSION

This paper presents a significant improvement for dense texture based algorithms suffering of the clustering problem. A local contrast phase analysis allows to avoid areas where streamlines cannot be well identified; the assignment of the starting gray tone is performed by checking the values of the neighbor pixels in order to maximize the local contrast. Some examples show how the proposed methodology can overtake the clustering problem without significantly affecting computational times. Future work will be aimed to use local contrast, not only for improving the quality of the images, but also for coding further information in the textures, in fact, contrast levels could be also used to code scalar values.

REFERENCES

[Montru01] Montrucchio, B., Montuschi, P., Sanna, A. and Sparavigna, A.: *Visualizing vector fields: the*

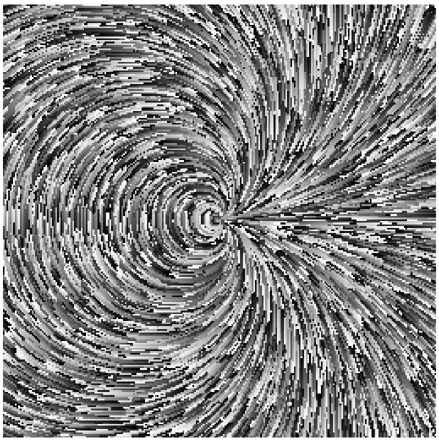

Figure 5: Example of a vector field that has only one attractor obtained by TOSL.

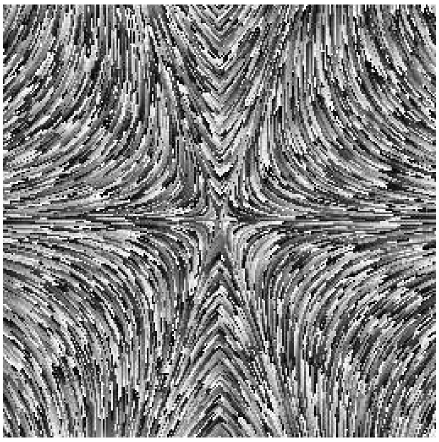

Figure 7: Example of a vector field that has only one critical point obtained by TOSL.

thick oriented stream-line algorithm (TOSL), Computers & Graphics, Vol. 25 No. 5, pp. 847-855, 2001

[Wijk91] Van Wijk, J.J.: *Spot noise-texture synthesis for data visualization, Proceedings of SIGGRAPH '91, Computer Graphics,* pp. 309-318, 1991

[Delee95] De Leeuw, W.C. and Van Wijk, J.J.: *Enhanced spot noise for vector field visualization, Proceedings of IEEE Visualization '95,* pp. 233-239, 1995

[Cabra93] Cabral, B. and Leedom, L.: *Imaging vector fields using line in-*

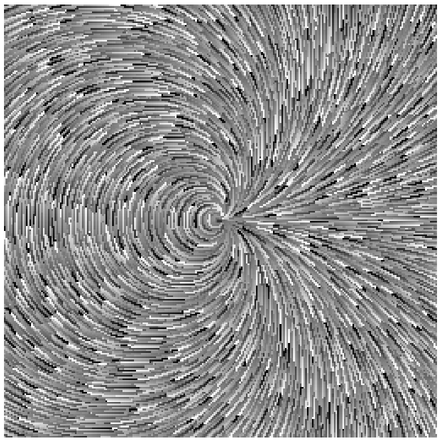

Figure 6: Example of a vector field that has only one attractor obtained by ECTOSL.

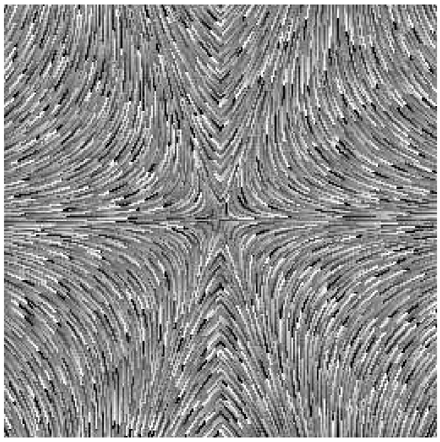

Figure 8: Example of a vector field that has only one critical point obtained by ECTOSL.

tegral convolution, Proceedings of SIGGRAPH '93, Computer Graphics, pp. 263-272, 1993

[Forse94] Forssell, L.K.: *Visualizing flow over curvilinear grid surfaces using line integral convolution, Proceedings of IEEE Visualization '94,* pp. 240-247, 1994

[Stall95] Stalling, D. and Hege, H.C.: *Fast and resolution independent line integral convolution, Proceedings of SIGGRAPH '95, Computer Graphics,* pp. 249-256, 1995

[Zockl97] Zöckler, M., Stalling, D. and Hege, H.C.: *Parallel line inte-*

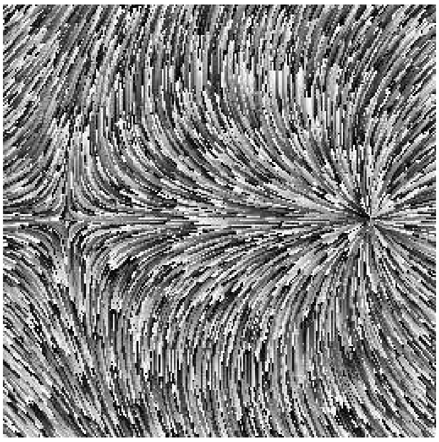

Figure 9: Example of a vector field that has two singularities obtained by TOSL.

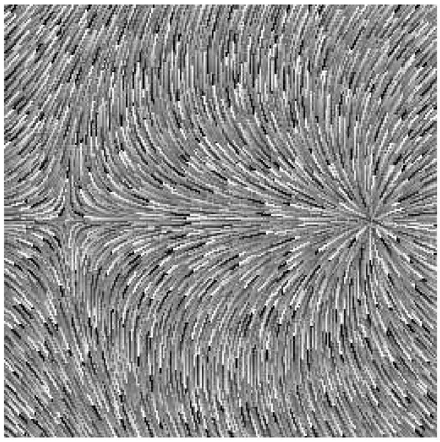

Figure 10: Example of a vector field that has two singularities obtained by ECTOSL.

gral convolution, Parallel Computing, Vol. 23 No. 7, pp. 975-989, 1997

[Wegen97a] Wegenkittl, R., Gröller, E. and Purgathofer, W.: *Animating flowfields: rendering of oriented line integral convolution, Computer Animation '97,* 15-21, 1997

[Wegen97b] Wegenkittl, R. and Gröller, E.: *Fast oriented line integral convolution for vector field visualization via the internet, Proceedings of IEEE Visualization '97,* pp. 309-316, 1997

[Risque98] Risquet, C.P.: *Visualizing 2D flows: integrate and draw, Proceedings of 9th Eurographics Workshop on Visualization in Scientific Computing,* 1998.

[Jobar97] Jobard, B. and Lefer, W.: *Creating evenly-spaced streamlines of arbitrary density, Proceedings of the eight Eurographics workshop on visualization in scientific computing,* pp. 57-66, 1997

[Khoua99] Khouas, L., Odet, C. and Friboulet, D.: *2D Vector field visualization using furlike texture, Data Visualization '99,* Vienna, pp. 35-44, 1999

[Forse95] Forssell, L.K. and Cohen, S.D.: *Using line integral convolution for flow visualization: curvilinear grids, variable-speed animation, and unsteady flows, IEEE TVCG,* Vol. 1 No. 2, pp. 133-141, 1995

[Shen98] Shen, H.W. and Kao, D.L.: *A new line integral convolution algorithm for visualizing time-varying flow fields, IEEE TVCG,* Vol. 4 No. 2, pp. 98-108, 1998

[Sanna00a] Sanna, A. and Montrucchio, B.: *Adding a scalar value to 2D vector field visualization: the BLIC (Bumped LIC), Eurographics'2000 Short Presentations Proceedings,* pp. 119-124, 2000

[Sanna01] Sanna, A., Montrucchio, B. and Montuschi, P.: *B^2LIC: an algorithm for mapping two scalar values on texture-based representations of vector fields, WSCG'2001,* pp. I138-I145, 2001

[Ware00] Ware, C.: *Information Visualization perception for design, Academic Press,* 2000