Fast Oriented Line Integral Convolution for Vector Field Visualization via the Internet

Rainer Wegenkittl and Eduard Gröller

Institute of Computer Graphics, Vienna University of Technology *

Abstract

Oriented Line Integral Convolution (OLIC) illustrates flow fields by convolving a sparse texture with an anisotropic convolution kernel. The kernel is aligned to the underlying flow of the vector field. OLIC does not only show the direction of the flow but also its orientation. This paper presents Fast Rendering of Oriented Line Integral Convolution (FROLIC), which is approximately two orders of magnitude faster than OLIC. Costly convolution operations as done in OLIC are replaced in FROLIC by approximating a streamlet through a set of disks with varying intensity. The issue of overlapping streamlets is discussed. Two efficient animation techniques for animating FROLIC images are described. FROLIC has been implemented as a Java applet. This allows researchers from various disciplines (typically with inhomogenous hardware environments) to conveniently explore and investigate analytically defined 2D vector fields.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation - Viewing Algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques - Interaction Techniques.

1 Introduction

Texture based techniques for the visualization of flow fields have been investigated in detail in recent years. Examples are, e.g., [1], [9], [11] and [17].

Van Wijk [17] explores stochastic texture synthesis to visualize scalar and vector fields. A spot noise texture is constructed by adding randomly weighted and positioned spots. Spot noise is a versatile technique where characteristics of the spot are intuitively transfered to characteristics of the spot noise texture. Varying the shape and features of the spot locally enables a local control of the texture. A flow field can be visualized by taking elongated ellipses as spots. The larger axes of these spots are, for example, aligned with the locally varying flow direction. The result is an anisotropic texture which depicts the entire flow field and does not distract the viewer with larger geometric features. The anisotropy of the texture manifests itself by a high correlation along the direction of the flow and a low correlation perpendicular to the flow. The spot noise technique can be thought of as filtering an isotropic texture (e.g., white noise) with a locally varying filter kernel. The shape and properties of the filter kernel encode local features, e.g., direction and velocity magnitude, of the underlying flow field. This is typically achieved by aligning the convolution kernel to the tangential direction of the flow. Four improvements and extensions to the spot noise technique are discussed in [4]. Spots are bent to better approximate the underlying flow. Filtering is done to eliminate undesired low frequency components from the spot noise texture. Graphics hardware methods for the acceleration of the spot noise texture generation are discussed. Furthermore the synthesis of spot noise on grids with irregular cell sizes is presented. Spot noise has also been utilized to visualize turbulent flow patterns [3].

Cabral and Leedom [1] introduced the Line Integral Convolution (LIC) method. In their approach filtering of a white noise input texture takes place along (curved) streamline segments. LIC uses one-dimensional filter kernels which are determined by integrating the underlying vector field. The intensity $I(x_0)$ at an arbitrary position x_0 of the output image is calculated by

$$I(x_0) = \int_{s_0 - s_l}^{s_0 + s_l} k(s - s_0) T(\sigma(s)) ds,$$

where T is the input texture, $\sigma(s)$ is the parameterized streamline through x_0 ($x_0 = \sigma(s_0)$) and k() describes the convolution kernel. s_l specifies the length of the streamline segment used in the filter operation. The texture values along the streamline segment $\sigma(s)$, $(s_0 - s_l \leq s \leq s_0 + s_l)$, are weighted with the corresponding kernel values $k(s - s_0)$. They are accumulated to give the intensity $I(x_0)$ at position x_0. Various kernel functions $k()$ can be used in the filter operation. For single images a constant filter kernel gives a good impression of the flow direction. Taking periodic low-pass filter kernels and phase shifting these kernels in successive images allows to animate the flow field. The animation shows flowing ripples which also encode the orientation of the flow.

Forssell [6] extended the Line Integral Convolution method to curvilinear grid surfaces. Calculations are done on a regular cartesian grid in computational space, whereas the results are displayed on curvilinear grids in physical space. Animating (i.e., phase shifting) the convolution kernel with constant kernel length in computational space produces distorted and misleading flow visualizations in physical space. This is due to a usually nonlinear mapping between computational space and physical space. The problem is overcome by adapting the length of the convolution kernel locally. Speed encoding is achieved by changing the amount of the phase shift of the convolution filter according to the locally varying velocity of the vector field.

Calculating a LIC image is a rather time consuming task. Stalling and Hege [14] reduced calculation times considerably by exploiting coherence properties. Given a simple (constant) convolution kernel k(), the difference in intensity $I(x_0) - I(x_1)$ of two adjacent positions x_0, x_1 on the same streamline can be calculated by an easy incremental update operation. The computation order is not pixel-per-pixel as in [1] but streamline oriented. This gives an order of magnitude speed-up with the drawback of allowing only simple convolution kernels. Additionally Stalling and Hege's method allows to zoom into a specific area of the input texture. Thus the input texture and the resulting image do not have to be of the same resolution.

Kiu and Banks [8] use Line Integral Convolution with multi-frequency noise texture. The locally varying magnitude of the vector field determines which frequency components are selected and integrated into a multi-frequency noise image. Areas with high velocities are represented by a noise function with lower spatial fre-

*Institute of Computer Graphics, Vienna University of Technology, Karlsplatz 13/186/2, A-1040 Vienna, Austria email:{wegenkittl, groeller}@cg.tuwien.ac.at

quency. Further texture based techniques are described, e.g., in [2], [10], and [13].

The variations of the Line Integral Convolution method presented so far do not encode the orientation of a flow within a still image. In section 2 Oriented Line Integral Convolution (OLIC) [16] is described, which overcomes this disadvantage. Section 3 discusses a new technique for Fast Rendering of OLIC images (FROLIC). Section 4 investigates the design of sparse textures for FROLIC. Animation aspects are described in section 5. The internet offers great potential of allowing various users in a heterogeneous setup to use the same visualization software. Section 6 deals with a Java implementation of the FROLIC algorithm. Finally in section 7 some conclusions are given and future work is outlined.

2 Oriented Line Integral Convolution (OLIC)

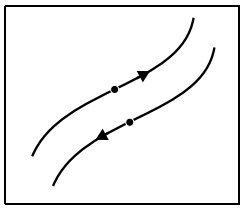


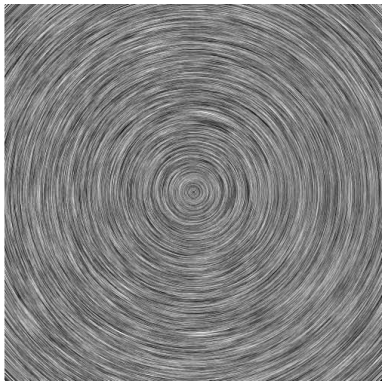
Figure 1: Two streamlines with equal direction but opposite orientation

LIC images encode flow direction and velocity magnitude, but they do not show the orientation of the flow in still images. Figure 1 shows two streamlines with equal direction but opposite orientation. LIC does not distinguish between these two cases. Flow orientation can be illustrated through animation. But there are cases where only still images are available or necessary, e.g., reproduction of vector fields in books or journals.

Furthermore LIC images are characterized by high spatial frequencies normal to the flow. This gives a good impression of the overall vector field, but is susceptible to aliasing problems in case an image has to be manipulated like, e.g., resized or printed. Oriented Line Integral convolution (OLIC) [16] was designed to show the orientation of a flow even in still images and it is not as much prone to aliasing effects as LIC. There are two major differences between LIC and OLIC. LIC images typically use dense noise textures wheras OLIC utilizes only sparse textures. A sparse texture can be thought of as a set of ink droplets which are thinly distributed on a sheet of paper. The vector field smears these ink droplets but the ink droplets are so far apart from each other that blurred traces of droplets usually do not overlap. The second difference between LIC and OLIC is that OLIC uses asymmetric convolution kernels (figure 2). A ramp-like kernel as in figure 2 produces traces of droplets with intensity variation along the streamline. As a sparse texture is taken traces do not overlap very much and the orientation of the flow is visible in still images.

In figure 3 the difference between LIC and OLIC is clearly visible. Figure 3(a) shows the LIC image of a circular flow but it is not recognizable if the flow is in clockwise or counterclockwise orientation. Figure 3(b) shows the OLIC image of a circular clockwise flow and figure 3(c) shows the OLIC image of a circular counterclockwise flow. The additional information in the OLIC image is gained at the expense of spatial resolution.

The initial positions of the droplets in the sparse texture must be selected carefully to avoid the formation of undesirable patterns in the OLIC image. In [16] the droplets are positioned on a regular grid. Additionally these positions are slightly jittered. If the distance between droplets is too large a lot of flow information is not depicted in the result image. On the other hand if the droplets are too close to each other, many traces will overlap. If the overlapping is too extensive the orientation is not clearly visible any more. Section 4 investigates the placement of droplets in the input texture.



(a)

(b)

(c)

Figure 3: LIC image (a), OLIC images for two flows with opposite orientation (b), (c)

OLIC allows to encode flow velocity by the length of the traces of individual droplets. The animation of OLICs can be achieved by simply phase shifting the convolution kernel for each frame of the animation sequence. The phase shift is adapted to the length of the trace of a droplet. Short traces have small phase shifts and long traces have large phase shifts. Initially each droplet is assigned a random phase shift (offset) to avoid synchronization artefacts. In [16] OLIC images are calculated pixel by pixel. In a precalculation step areas of the result image which are not affected by any of the droplets can be determined and skipped in the following convo-

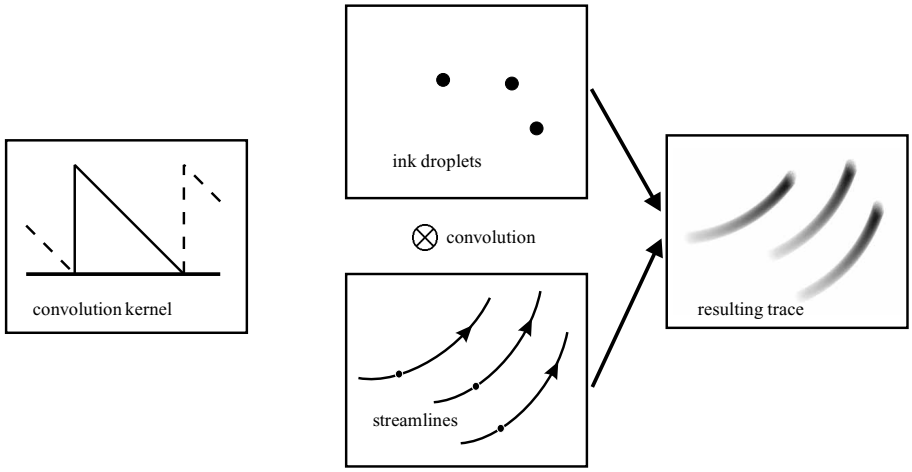


Figure 2: Sparse texture and ramp-like kernel-function for OLIC

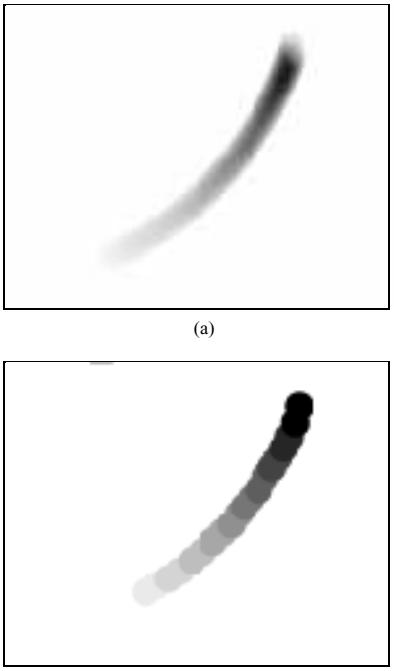
lution process. Despite this optimization the calculation of OLIC images is quite slow and comparable to the cost of calculating LIC images. In section 3 a fast approximative calculation of OLIC is presented.

3 Fast Rendering of OLIC (FROLIC)

One characteristic feature of OLIC is the usage of sparse textures. Usually a sparse texture is made up of constant intensity droplets although one can think of droplets with a different, e.g., gaussian distributed, intensity function. The distance between droplets is large enough so that traces of neighboring droplets typically do not overlap too much. This allows, in combination with the asymmetric convolution kernel, to illustrate the orientation of the flow. Convolution with a sparse texture is not as difficult as convolution with a dense texture. This will be now used for Fast Rendering of OLIC images (FROLIC).

FROLIC calculates an approximate solution to the exact convolution result of OLIC thereby achieving a considerable speed-up. With a sparse texture the convolution at a specific point of the result image involves at most one single droplet. Each droplet produces a trace with intensity increasing from tail to head. Due to the circular shape of a droplet the intensity varies slightly along the breadth of a trace as well (figure 4(a)). As a trace is rather small FROLIC approximates the shape of a trace by a set of small, possibly overlapping disks (figure 4(b)). The disks are positioned along a short portion of a streamline. Each disk has constant intensity, but the intensity varies between adjacent disks. If n disks are taken to approximate a trace then intensity increases in n discrete steps from tail to head of the trace. The intensity of adjacent disks is increasing to simulate the continuous ramp kernel of the OLIC method. Although being an approximative variant of a LIC-type algorithm, FROLIC is also somewhat in the spirit of iconic vector field representations. Such approaches are, e.g., spot noise [17], surface particles [18], and particle traces on 2D surfaces [11].

The FROLIC calculation is done as follows: for each droplet a short streamline portion is calculated by integrating the underlying flow field. The length of a streamlet indicates local flow velocity. The smallest and largest velocities in the vector field are assigned specific streamlet lengths, intermediate velocity values are linearly interpolated. A prefixed number of disks is positioned in regular intervals along a streamlet. The processing order is from tail to head of the trace, i.e., darker disks are drawn first and might be



(a)

(b)

Figure 4: Exact trace of a droplet with OLIC (a) and approximated trace of a droplet with FROLIC (b)

partially occluded by brighter disks which are drawn later on. In our approach the number of disks positioned along a streamlet is independent from the length of the streamlet. This allows an easy animation algorithm as described in section 5.

The main advantage of FROLIC as compared to OLIC is that drawing disks is much faster than doing a costly convolution calculation. Instead of calculating the result image pixel per pixel as OLIC does, only the rather small set of droplets has to be processed. In an image with resolution of 600x600 about 1000 droplets are sufficient. Furthermore drawing simple geometric primitives like disks can be done with hardware support. During experiments we found that the approximation error introduced by the FROLIC method is well justified by the efficiency gain. The calculation time for the FROLIC images in figures 5(b) and 6(b) is about two seconds on

a Pentium 100Mhz PC. Investigations show that FROLIC (without hardware supported rendering) is approximately two orders of magnitude faster than OLIC. Figures 5 and 6 give a comparison between OLIC and FROLIC images. The images were calculated with a resolution of 600x600. The calculation of the FROLIC image in figure 5 was 295 times faster than the corresponding OLIC image. In figure 6 the speed-up was 170. The difference in the speed-up factors is mainly due to the longer streamlets in figure 5 which are more costly to calculate with OLIC than with FROLIC.

OLIC is independent from the specific texture, whereas the cost of FROLIC depends on the number of droplets. Increasing the kernel length increases the cost of both methods. In this case OLIC has to convolve the texture with a larger streamline portion and FROLIC has to draw more disks for each streamlet. The radius of the droplets has no effect on the OLIC calculation, while it slightly influences the FROLIC performance. This is due to the fact that larger disks have to be drawn in this case. A slight performance degradation happens with FROLIC when the resolution of the output image is increased. A larger image resolution, on the other hand, slows down the OLIC calculation considerably. Speed encoding influences the OLIC performance, as the length of streamlets (and therefore the convolution cost) may vary to a great degree between different flow fields. This is not the case with FROLIC as the number of disks drawn for each streamlet is independent from its length.

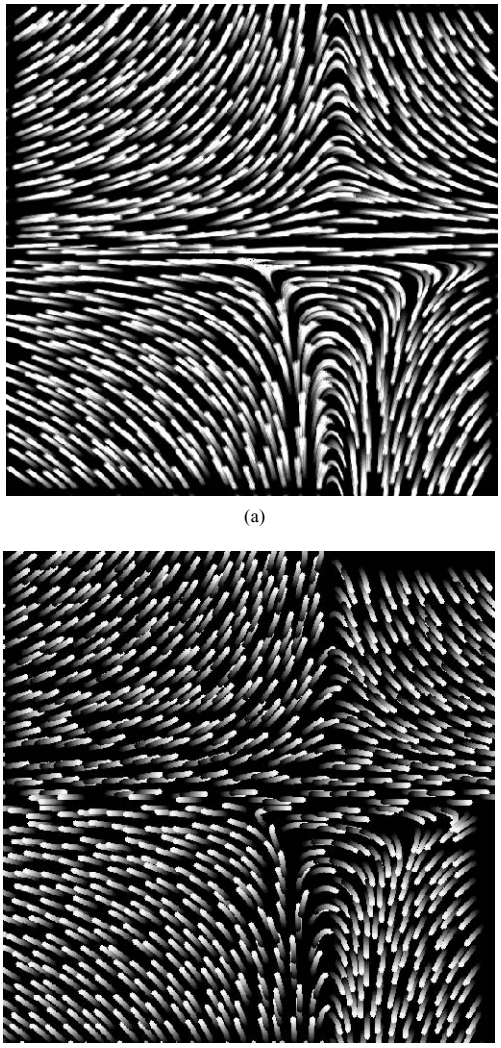
4 Droplet Texture Design

This section deals with the placement of droplets on a sparse input texture. There are two criteria which should be optimized but which are opposed to each other. One criterion calls for a dense filling of the output image with streamlets. This ensures that most of the vector field information is represented in the output image. The second criterion is that overlapping streamlets should be avoided as far as possible in order to clearly illustrate flow orientation.

Finding an optimal droplet distribution in the input texture so that a tight packing of streamlets results in the output image is quite intricate. The optimal texture depends on the underlying vector field as well as on the chosen minimal and maximal streamlet lengths. Furthermore changing the position of a droplet has nontrivial consequences concerning the induced streamlet. The streamlet may change its length or shape. This complicates filling algorithms which are based on distributing and moving droplets in the input texture. Another consideration is that the arrangement of the streamlets should not produce macro structures which are easily perveived by the human visual system and which disturb the interpretation of the flow data. Such macro structures might result for example if streamlets are exactly aligned along a specific streamline or the alignment of streamlets is such that they form wavefront patterns.

There has already been work on optimal streamline placement [7], [15]. Turk and Banks [15] use an energy function to guide the placement of streamlines. An image with an uneven distribution of streamlines (in certain areas streamlines are either too crowded or too sparse) has higher energy than an image with a more even streamline density. The optimization process decreases the energy function which approaches a local minimum. The resulting images look somewhat like elegant hand-designed streamline drawings but the optimization process itself is quite costly.

Our task deals only with the placement of short streamlets and is therefore inherently simpler than the streamline placement in [15]. Simple approaches for droplet placement are random distribution and placement of droplets on a regular or jittered grid. If the distance of adjacent grid points is in the same order as the maximal streamlet length then overlapping may occur but is usually not a severe problem. If overlapping of streamlets shall be avoided en-



(a)

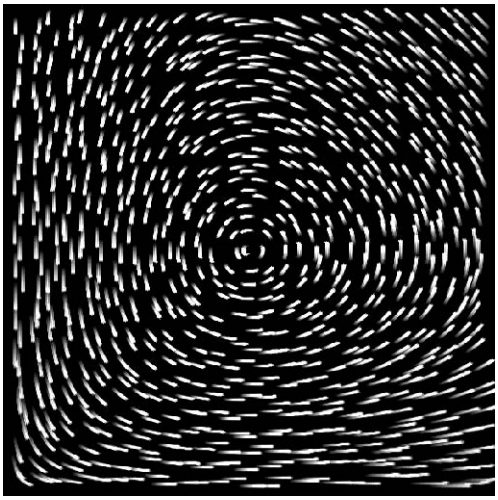
(b)

Figure 5: Econometric data with OLIC (a) and FROLIC (b)

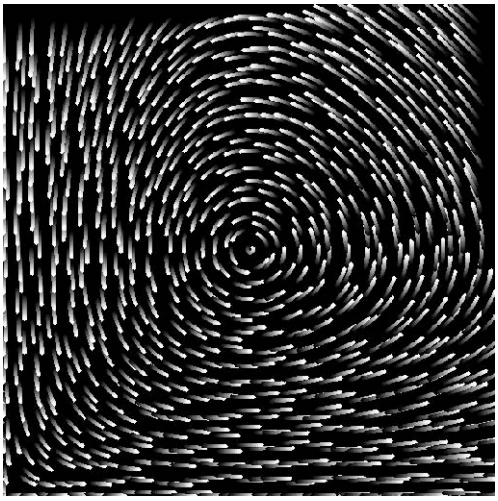
tirely, a distance image is used which has the same resolution as the final output image. For each pixel the distance image contains the distance to the closest streamlet drawn so far (figure 7(b),(c)). A new streamlet candidate is calculated and if it is too close to a previously drawn streamlet it is discarded. Otherwise the streamlet is drawn into the output image as a set of disks and the distance image is updated.

The update operation is based on a rasterized template of the disk (figure 7(a)). Pixels within the disk have the distance value zero, other pixels of the template contain the distance to the disk up to a certain threshold value d. For each disk of the streamlet the template is pasted into the distance image in a z-buffer fashion. If a pixel of the output image is now closer to the currently drawn disk its distance value is modified to the corresponding template value otherwise it remains unchanged. This approach ensures that streamlets are not closer to each other than the predefined threshold value d.

The threshold value d determines the density of streamlets in the output image. Experiments have shown that a small value of d produces a dense output image but undesirable macro structures might occur. If only pixel positions on, e.g., a jittered grid are tested for possible streamlet placement and overlapping streamlets are dis-

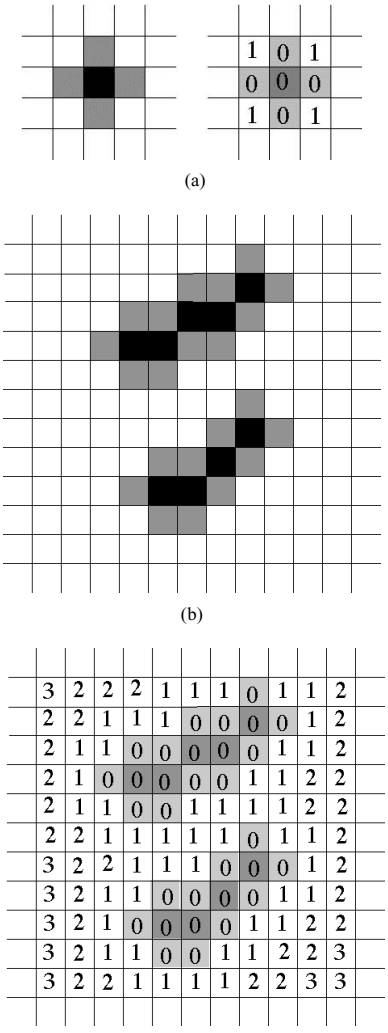


(a)



(b)

Figure 6: Circular flow with OLIC (a) and FROLIC (b)



(a)

(b)

(c)

Figure 7: Rasterized template of a disk (a), two streamlets (b), distance image (c)

carded there could be some thinly populated areas in the output image. In a second pass additional pixel positions are tested for streamlet placement. The search order in the second pass can be random, circular starting from the image center, or according to a Peano curve. The search order should not be too regular, e.g., in scanline order, so that the danger of macro structures is reduced.

Doing the fill operation in one step without initially placing streamlets globally, e.g., on a grid, is prone to generating macro structures. By choosing the threshold value d to be at least three to four times the perimeter of the streamlets generally avoids the macro structures. Figure 8 gives a comparison between a FROLIC with and without overlapping streamlets respectively. The threshold d in figure 8 is small as compared to the streamlet perimeter. Therefore some macro structures are recognizeable.

5 Animation of FROLIC

Animation of OLIC images is realized by phase shifting the convolution kernel in consecutive frames of the animation sequence (see section 2). This approach can also be adapted to FROLIC. With FROLIC a streamlet consists of a set of disks with varying intensity. These intensities are cycled to convey to the viewer the impression of motion. The spatial position of streamlets is not changed. The starting points of streamlets can be thought of as nozzles which periodically introduce dye into the flow. This is an Eulerian approach [12] as opposed to using moving particles, which would be a Lagrangian approach. Moving particles have the disadvantage that depending on the flow field an uneven particle distribution may evolve quite rapidly. In the following we will discuss two algorithms how to realize animation of FROLIC images. The first algorithm was implemented as a Java applet [5] which will be described in more detail in section 6. The second algorithm realizes FROLIC animation by color-table animation.

The first algorithm is based on the fact that a linearly increasing intensity function has to be cycled. Each streamlet is again assigned a random initial phase shift to avoid synchronization effects. For each streamlet the current position c indicates the disk with highest intensity. Given frame i of the animation sequence the following frame i+1 is constructed by reducing the intensity of all pixels of frame i by a fixed amount. This amount is equal to the intensity difference between adjacent disks. Frame i+1 is built from the intensity-reduced frame i by drawing a single disk for each streamlet. These disks have highest intensity and are positioned at location (c+1) mod n. n is the number of disks used to represent a streamlet. Both operations (i.e., intensity reduction and disk draw-

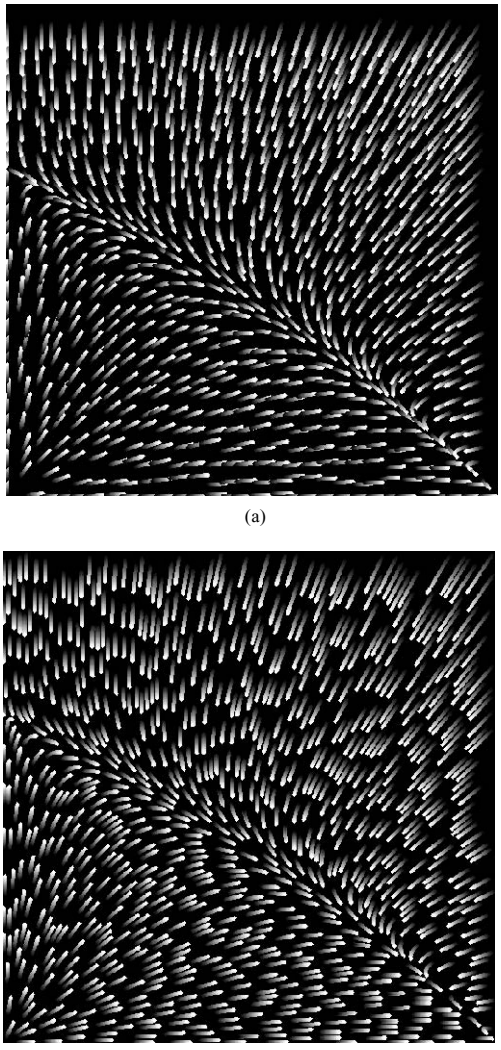


Figure 8: FROLIC image (a) with, (b) without overlapping streamlets

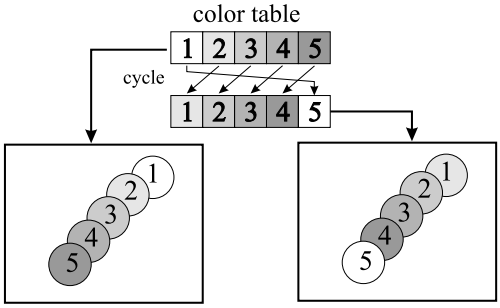


Figure 9: Color-table animation for FROLIC, two consecutive frames

ing) together cycle the intensity ramp one disk along the flow.

The human visual system is very sensitive to appearing or disappearing bright spots. This can be a problem for cycling an intensity ramp along a fixed streamlet, as a bright disk disappears at the end of a streamline and reappears at the start of the streamline. The impression of a pulsating effect can be, however, avoided by using a filter (e.g., Gaussian) which attenuates the intensity at the beginning and the end of a streamlet. As a streamlet is typically made up of a small number of disks we use in our implementation a simple filter which initially increases linearly, is constant in the middle portion, and finally decreases linearly. Whenever a new disk is drawn its intensity is modified with the above filter according to the disk position.

Color-table animation is the second approach for efficiently animating FROLIC images. With a color table the intensity value of a pixel is specified indirectly. Each pixel is assigned a short color-table index which points to a specific entry in the color table. Available intensities or colors are stored in the color table itself. Color-table animation changes the entries of the color table instead of changing the corresponding image. As the color table is only small in size, e.g., 256 entries, this can be done very fast. The possibilities of color-table animation are rather limited but sufficient for animating FROLIC images. Figure 9 illustrates the principle. A streamlet is initially built by drawing a set of disks. Adjacent disks are represented by consecutive color-table indices. The color table holds the intensity ramp, i.e., successive color-table entries contain increasing intensity values. After this initialization step the assignment of indices to pixels is not changed anymore. Animation is achieved by cycling the intensity values in the color table itself (see figure 9). The random initial phase shift (offset) is realized by starting each streamlet with a randomly selected color-table index.

There is a problem if the intensity of a streamlet should be additionally attenuated at the beginning and end of the streamline. In this case the intensity of a disk does not only depend on the offset within the intensity ramp but is also dependent on the spatial position. This means that a color-table index can not simultaneously represent a disk in the middle of one streamlet and a disk at the beginning or end of another streamlet. This situation is handled by subdividing the color table into non overlapping equal-sized regions. Each region represents all the streamlets with the same initial phase shift. If, for example, there are 256 color-table entries and each streamlet is represented by 32 disks then 8 (256/32) different initial phase shifts can be realized.

We have currently included the first algorithm into our prototype implementation (see section 6) as successive intensity reductions of entire images are easily realized in Java. As we can not manipulate color-table entries within our Java applet, we are currently implementing the color-table animation approach under another software environment.

6 FROLIC via the Internet

This section describes a prototype implementation of FROLIC and some other vector field visualization techniques within a Java applet [5]. The applet can be accessed at `http://www.cg.tuwien.ac.at/research/vis/dynsys/frolic/`. Over the last years we have been collaborating with researchers working on analytically defined dynamical systems. Depending on the professional background, e.g., mathematicians, economists, the scientists typically have a quite inhomogenous hardware equipment. Most of these scientists do not have high-end graphics hardware at their disposal. Therefore it can be difficult to provide them with visualization tools that are specifically tailored to their needs.

Internet computing provides a mechanism to offer people hardware-independent visualization capabilities. The same techniques can be used in different setups without the need to modify the implementation. Software maintainance and update is no problem as the latest version is available over the internet. The Java applet provides some methods to experiment with and investigate analytically defined 2D vector fields. This gives researchers the

possibility to easily visualize their analytical vector data without having to use any complex visualization tool.

A 2D vector field is defined by a system of two differential equations. A fast rendering of the flow behavior of such a system is crucial to efficiently explore different parameter settings and variations of a given vector field. As LIC and OLIC produce high quality images at a high computational cost, FROLIC was developed to provide an approximative but fast representation of the system behavior. The user submits among other parameters: the vector field equations, the area of interest and the resolution of the output image. During the calculation of a FROLIC image a numerical simulation of the vector field is done by applying, e.g, Euler or Runge-Kutta methods. FROLIC images can be calculated with or without the avoidance of overlapping streamlets. Animations of FROLIC images are also possible.

Other techniques included in the applet are the following: The speed of the vector field is encoded in an intensity image. High intensity areas correspond to high velocity regions of the vector field. A hedge-hog representation of the vector field illustrates the flow with arrow glyphs positioned on a regular grid. Isoclines, i.e., curves where the flow is either horizontal or vertical, may also be calculated. LIC and OLIC images are determined by doing exact (and costly) convolution operations. Particles may be introduced into the flow as well. As particles change their location usually an uneven particle distribution results very soon.

7 Conclusion

This paper describes various extensions to Oriented Line Integral Convolution (OLIC). As opposed to Line Integral Convolution (LIC) OLIC images depict the orientation of a flow field even within a still image. Calculating an OLIC image involves expensive convolution operations. FROLIC is presented in this paper which accelerates the calculation of OLIC images by about two orders of magnitude. This is achieved by approximating a streamlet by a set of disks with varying intensity.

Two algorithms are discussed to efficiently animate FROLIC images. FROLIC is implemented as a Java applet to allow researchers with different hardware resources to easily explore 2D analytical dynamical systems through internet computing.

Future work will include variations of streamlet approximation. For example streamlets in strongly converging or diverging areas will be represented by disks with varying radii. Instead of approximating a streamlet by a set of (usually overlapping) disks one can also think of using predefined footprints. For each streamlet the best suited representation in a footprint table is selected.

Visualization over the internet is a promising new area of research. We plan to extend the implemented Java applet and adapt its functionality according to user reactions.

Acknowledgement

The authors would like to thank Andreas König and Werner Purgathofer for fruitful discussions and Andreas König for proofreading a draft version of this paper

References

[1] B. Cabral, C. Leedom, *"Imaging Vector Fields Using Line Integral Convolution"*, SIGGRAPH 93 Conference Proceedings, pages 263–270, ACM SIGGRAPH, 1993.

[2] R. A. Crawfis, N. Max, *"Texture Splats for 3D Scalar and Vector Field Visualization"*, IEEE Visualization '93 Proceedings, pages 261–265, 1993.

[3] C. W. deLeeuw, F. H. Post, R. W. Vaatstra, *"Visualization of Turbulent Flow by Spot Noise"*, Virtual Environments and Scientific Visualization '96, pages 287–295, Springer, 1996.

[4] C. W. deLeeuw, J. J. van Wijk, *"Enhanced Spot Noise for Vector Field Visualization"*, IEEE Visualization '95 Proceedings, pages 233–239, 1995.

[5] D. Flanagan, *"Java in a Nutshell"*, O'Reilly & Associates, Inc., 1996.

[6] L. K. Forssell, *"Visualizing Flow over Curvilinear Grid Surfaces Using Line Integral Convolution"*, IEEE Visualization '94 Proceedings, pages 240–247, 1994.

[7] B. Jobard, W. Lefer, *"Creating Evenly-Spaced Streamlines of Arbitrary Density"*, Proceedings of the Eight Eurographics Workshop on Visualization in Scientific Computing, France, pages 57–66, 1997.

[8] M.-H. Kiu, D. C. Banks, *"Multi-Frequency Noise for LIC"*, IEEE Visualization '96 Proceedings, pages 121–126, 1996.

[9] H. Löffelmann, A. König, E. Gröller, *"Fast Visualization of 2D Dynamical Systems by the Use of Virtual Ink Droplets"*, Proceedings of Spring Conference on Computer Graphics, Slovakia, pages 111-118, 1997.

[10] X. Mao, M. Kikukawa, N. Fujita, A. Imamiya, *"Line Integral Convolution for Arbitrary 3D Surfaces through Solid Texturing"*, Proceedings of the Eight Eurographics Workshop on Visualization in Scientific Computing, France, pages 67–76, 1997.

[11] N. Max, R. Crawfis, Ch. Grant, *"Visualizing 3D Velocity Fields Near Contour Surfaces"*, IEEE Visualization '94 Proceedings, pages 248–255, 1994.

[12] F. H. Post, T. Walsum, *"Fluid Flow Visualization"*, In H. Hagen, et.al. (eds) Focus on Scientific Visualization, pages 1–40, Springer, 1993.

[13] H.-W. Shen, Ch. R. Johnson, K.-L. Ma, *"Visualizing Vector Fields Using Line Integral Convolution and Dye Advection"*, 1996 Symposium on Volume Visualization, pages 249–256, ACM SIGGRAPH, 1995.

[14] D. Stalling, H.-C. Hege, *"Fast and Resolution Independent Line Integral Convolution "*, SIGGRAPH 95 Conference Proceedings, pages 249–256, ACM SIGGRAPH, 1995.

[15] G. Turk, D. Banks, *"Image-Guided Streamline Placement"*, SIGGRAPH 96 Conference Proceedings, pages 453–459, ACM SIGGRAPH, 1996.

[16] R. Wegenkittl, E. Gröller, W. Purgathofer, *"Animating Flowfields: Rendering of Oriented Line Integral Convolution"*, Computer Animation '97 Proceedings, pages 15–21, IEEE Computer Society, June 1997.

[17] J. J. van Wijk, *"Spot Noise Texture Synthesis for Data Visualization"*, Computer Graphics (SIGGRAPH 91 Conference Proceedings), volume 25(4), pages 309–318, July 1991.

[18] J. J. van Wijk, *"Flow Visualization with Surface Particles"*, IEEE Computer Graphics & Applications, pages 18–24, July 1993.