# PLIC: Bridging the Gap Between Streamlines and LIC

Vivek Verma[1], David Kao[2], and Alex Pang[1]
[1] Computer Science Department, UCSC
[2] NASA Ames Research Center
vivek@cse.ucsc.edu, davidkao@nas.nasa.gov, pang@cse.ucsc.edu
www.cse.ucsc.edu/research/avis/plic.html

## Abstract

This paper lays the groundwork for comparing flow visualizations using streamlines and line integral convolution (LIC). Our approach is to identify and define relevant parameters in each of these flow visualization techniques. Mapping strategies are then designed to generate LIC-like images from streamlines and streamline-like images from LIC. The result is a technique which we call pseudo-LIC or PLIC. The main contribution being reported in this paper is a methodology for flexibly generating flow visualizations that span the spectrum of streamline-like to LIC-like. Among the advantages are: performance speedups over LIC, applicability to time varying data sets, and variable-speed animation.

**Key Words and Phrases:** unsteady flow, variable speed animation, jitter, texture mapping, comparative visualization.

## 1 INTRODUCTION

This work attempts to bridge the gap between streamlines and LIC by producing flow visualizations that vary continuously from being streamline-like to LIC-like in appearance. These two seemingly disparate techniques are often used to study the same phenomenon in fluid flow over a surface. Ultimately, we hope to get a better idea on how to compare the flow features in visualizations generated using streamlines and LIC.

Both streamlines and the LIC methods have advantages and disadvantages. Streamlines are generated by advecting a massless particle from an initial seed location and tracking its path using polylines. They are straightforward and relatively cheap to compute, are naturally extended to 3D, and provide a continuous, coherent image of the flow pattern. However, the flow patterns are also notoriously dependent on the seed placement, need to be computed in physical space, and not directly extendible to time varying data sets – streaklines, being the more appropriate method there. LIC images are generated by coloring each pixel according to a convolution of an input noise texture with a one dimensional filter kernel along a streamline passing through that pixel. They are also relatively straightforward to compute, can be done in computational space, and give a dense view of the flow without cluttering the display. On the other hand, they are more expensive than streamlines, difficult to extend to 3D, and may suffer from graininess or poor contrast.

One of the goals we hope to achieve is to take advantage of the strengths of both techniques. Our approach towards this end is to provide users with enough flexibility to select a flow visualization parameterization within the spectrum spanning streamlines and LIC. Specifically, we would like to identify the visualization parameters for streamlines and for LIC, understand how these affect the resulting visualization, and find a common or corresponding set of parameters to simulate one method with the other.

The next section provides some relevant background material that highlights some of the issues of streamlines and LIC. We then present one possible mapping of one flow visualization method with another. This is followed by a formalization of the PLIC method and discussion of issues such as seed placement, performance, extension to time varying data sets, variable-speed animation and limitations. We conclude with some additional ideas for future work.

## 2 RELATED WORK

Streamlines give a global view of the flow features. However, the presentation suffers from visual artifacts that result from seed placement. To address this problem, image-guided strategies were proposed to give a more uniformly distributed set of streamlines [17], and was subsequently improved to converge faster [10]. In addition, there have also been previous efforts in animating steady flows using color table animation [7] and cyclical set of textures [9].

In 1993, Cabral and Leedom [1] introduced a new method for visualizing 2D vector fields called line integral convolution or LIC. Since then, a number of papers have appeared that expanded the scope and usefulness as well as improved and refined the original work. The list includes animation of steady flows [19, 5] and unsteady flows [15, 14, 6], use of color to show flow properties [13], extension to curvilinear grids [5], improvements to LIC appearance [12, 11], improvement in speed [16], and extensions to 3D [8].

In general terms, the visual appearance of PLIC is very similar to those of LIC images, spot noise [18, 3], and textured streamlines [9]. We now describe our approach and how it differs from previous work.

## 3 MAPPING STRATEGIES

Our general strategy is to identify and understand how the different parameters for streamlines and LIC affect the resulting visualization. As a first attempt to understand the differences between the two techniques and have the ability to compare them, we explored ways to generate images that looked like streamlines using the basic LIC algorithm. The next step was to use the streamline method to generate images that look like LIC. We manipulated the parameters of both methods to get a better understanding of the techniques.

The parameter list for the two different methods are as expected. For streamlines, they are seeding density and/or strategy, integration method and associated parameters, whether backward integration is done or not, and thickness of streamlines. For LIC, they are properties of the input texture, type of filter used for convolution, how particle advection is done, integration method and associated parameters. We experiment with some of these parameters below.

### 3.1 LIC to Streamlines

One of the important steps in the LIC algorithm is to generate streamlines before convolving them with the input texture. We experimented with several input textures. We wanted to study the effect of using patterns other than random noise. Some of the patterns we experimented with and the corresponding LIC images are shown in Figure 1. Our experiments demonstrated that the reason why LIC works so well is because of the lack of correlation in the input texture. Any correlation in the input texture would get strengthened in ways dependent on the flow field, and show up in the output image as artifacts. Figure 1 shows some input patterns we experimented with and the corresponding LIC images of the bluntfin data set.

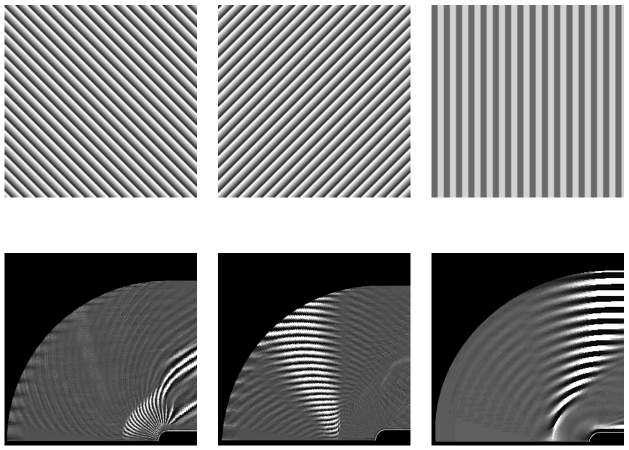

Figure 1: Input patterns and the corresponding LIC images of the bluntfin dataset. Note that aliasing artifacts and the bands of brighter regions move around as the input texture is changed.

We observed that the LIC algorithm smears the input texture in the direction of the flow. This behavior is much similar to the physical experiments conducted to study the fluid flow by injecting a colored dye in the fluid [13] (or particles: see album of fluid flow [4]). Such a process can be simulated by using the LIC algorithm in the following way:

```
(x,y) = seed point for streamline;

// set the seed point in the input texture

set InputImage      = 0;
set InputImage(x,y) = 255;

// generate "streamline"

repeat
  OutputImage = LIC(InputImage);
  If a pixel in OutputImage != 0, make it 255;
  InputImage  = OutputImage;
until (satisfied);
```

The result of applying 40 iterations to the repeat loop can be seen in Figure 2.

Although the above technique can be used to generate streamlines, it is highly inefficient. The purpose of the technique is not to generate streamlines using LIC but to aid in understanding the gap between streamlines and LIC. For example, the strategy outlined above basically produces thick streamlines. This can be used to demarcate regions of diverging flows that might be difficult to achieve using normal streamlines.

## 3.2 Streamlines to LIC

One of the goals was to provide a slider that can be adjusted to generate pictures that look like streamlines on one extreme and LIC on

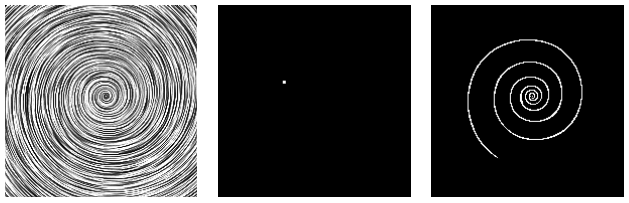

Figure 2: Left: Enhanced LIC image of the spiral data set. Middle: Input texture (only one pixel is turned on). Right: Result of repeatedly applying LIC to the input texture with only one pixel turned on.

the other extreme. We have found that there are multiple parameters that need to be manipulated. We explore some of these parameters below.

In order to understand the coherence of texture generated by LIC, we generated two images: one using streamlines and the other using LIC. These images were of the same dimensions. We used the streamlines image as a mask on the LIC image to generate the image in Figure 3(c).

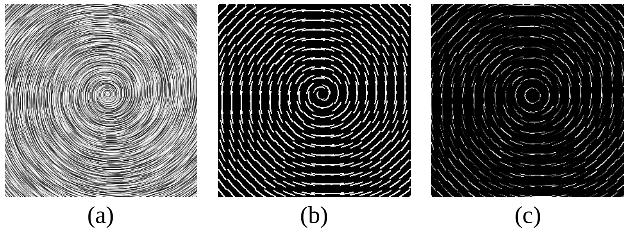

Figure 3: Left: Image generated using enhanced LIC. Middle: Streamlines. Right: Intersection of the two images.

From Figure 3(c) we can see that each streamline appears to be texture mapped. The texture appears to be the result of applying LIC to a one dimensional noise image and a unidirectional flow field. This experiment suggested that it is not necessary to do the expensive convolution in LIC. One can simply generate streamlines and texture map them so that the result looks like LIC. In fact, this approach is also taken by [9] where they apply cyclical textures on their streamlines. The main difference between our approach is the properties of the textures that we use and how we apply them.

We saw earlier that the white noise used in the input texture plays an important role in the sense that due to its random nature, there are no leftover patterns or artifacts in the LIC image. Likewise, we would expect artifacts to appear in the resulting image if we use the same one dimensional texture for all the streamlines. Hence we decided to use a different one dimensional texture for each streamline. A collection of one dimensional textures, that are different from each other, can be easily generated using LIC itself. Such a collection of one dimensional textures was generated by applying enhanced LIC [12] to a two dimensional flow field. The flow field and the resulting texture can be seen in Figure 4. Each column of the LIC image in Figure 4 is a one dimensional texture and is different from all the other columns. Each column can be used to texture map a different streamline. We refer to the enhanced LIC image of a constant flow field as the *template texture*. Note that we need to generate a template texture image only once, because the same template texture can be used for any flow field. Also note that unlike LIC, the template texture is not required to have the same dimensions as the flow field. To generate an image like Figure 3(c) we can generate streamlines and for each streamline, we can choose a column randomly from the template texture. This column is the one-dimensional texture used to texture map the given streamline.

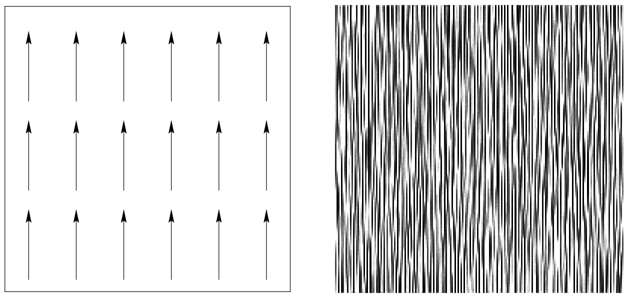

Figure 4: Left: A constant flow field. Right: The corresponding enhanced LIC image (*template texture*).

The result of texture mapping streamlines can be seen in Figure 5(b). It looks very similar to Figure 5(a) which was generated using the intersection of a LIC image and a streamline image.

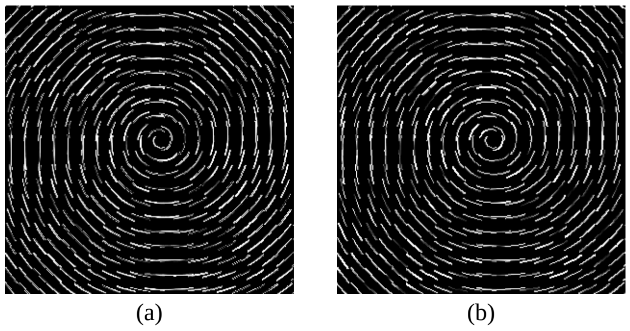

Figure 5: Left: Intersection of LIC and streamlines. Reproduced from Figure 3(c). Right: Texture mapped streamlines.

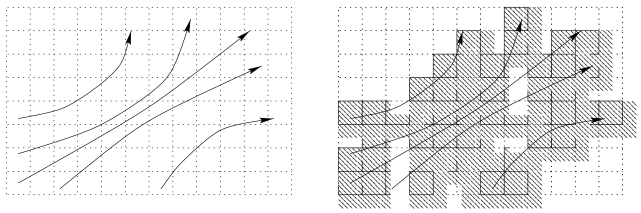

Figure 6: Left: Streamlines intersecting pixels in the image space. Right: The corresponding pixels have been turned on. Individual streamlines cannot be distinguished.

One of the disadvantages of streamlines is that more than one streamlines could intersect the same pixel in the image space, as can be seen in Figure 6. If we simply turn a pixel on if a streamline intersects it, and off if no streamlines intersect it, then we can get pictures in which individual streamlines cannot be distinguished from each other. To solve this problem, we maintain a count for each pixel that keeps track of the total number of streamlines that intersect it. During texture mapping of a streamline, for each pixel, we accumulate the contribution of the texture from all the streamlines that intersect that pixel. As a final step, we normalize the value of each pixel by dividing the accumulated texture values by the total number of streamlines that intersect that pixel. This process is similar to scattering in UFLIC [15]. The result can be seen in Figure 7, where we compare PLIC with enhanced LIC and the basic LIC methods.

The PLIC images in Figure 7 look remarkably similar to the basic LIC method. Each pixel in the images in Figure 7 corresponds to a cell in the computational space. These images were generated by seeding each cell in the computational space. We found that if we seed each cell then the pseudo-LIC (PLIC) method is only slightly faster than LIC. This is so because LIC spends most of its time in streamline computation. The convolution is done using precalculated function tables and the major cost in the convolution step is the normalization by the kernel area. The equivalent of this cost in PLIC is the normalization of each pixel by the total number of streamlines that intersect it.

PLIC can be made more efficient by having a sparser seed distribution to generate streamlines. The streamlines can be drawn thicker accordingly. Figure 8 shows that when the streamlines are drawn like ribbons then more than one column from the template texture can be used to texture map them. It is important that the columns chosen are adjacent to each other in the template texture image so that the PLIC image has good spatial coherence. The thickness of a streamline is specified in pixels. If a streamline is $n$ pixels wide and the texture chosen from the template image has column number $k$, then $n$ streamlines are drawn parallel to each other such that the starting points of the streamlines are adjacent to each other and in a straight line. The columns chosen to texture map these $n$ streamlines are $k - \frac{n}{2}$, $k - \frac{n}{2} + 1$, ..., $k$, $k + 1$, ..., $k + \frac{n}{2} - 1$. Hence, a subimage of the template texture is used to texture map the thick streamline (see Figure 9). Also note that the central streamline is traced once and is stored so that it can be replicated $n - 1$ times. $\frac{n}{2}$ copies of this streamline are placed adjacent to each other on one side and $\frac{n}{2} - 1$ copies are placed on the other side. It should also be noted that drawing thick streamlines using the above describe algorithm does not always produce streamlines that are parallel to each other. Nonetheless, we found the method to be satisfactory. An accurate computation and texture mapping would be considerably costlier.

The thickness $n$ of streamlines is usually chosen proportional to the sparseness of the seed placement. The further apart the seeds for streamlines are, the larger $n$ can be. We specify the distance between adjacent seeds using a parameter called *stride*. In simple terms the parameter stride determines how far apart the seeds are placed along each row and column of the grid. Varying these two parameters also allow us to create variably spaced flow images where some regions are LIC-like (where the flow merges together) and more streamline-like (where the flow either diverges or are parallel to each other). Following is the pseudo-code for PLIC:

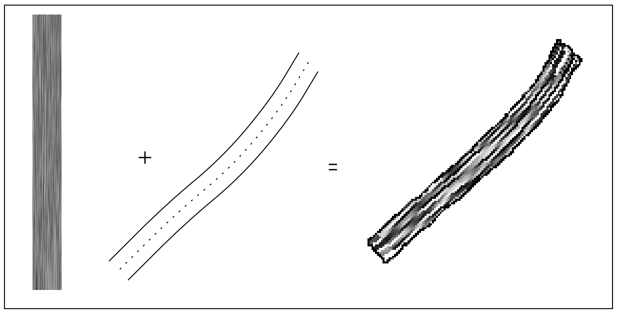

Figure 8: Texture mapping thick streamlines.

```
PLIC()
{
  stride_x = stride in the x direction;
  stride_y = stride in the y direction;
  n = width of streamlines;
  (max_x, max_y) = grid dimensions;
```

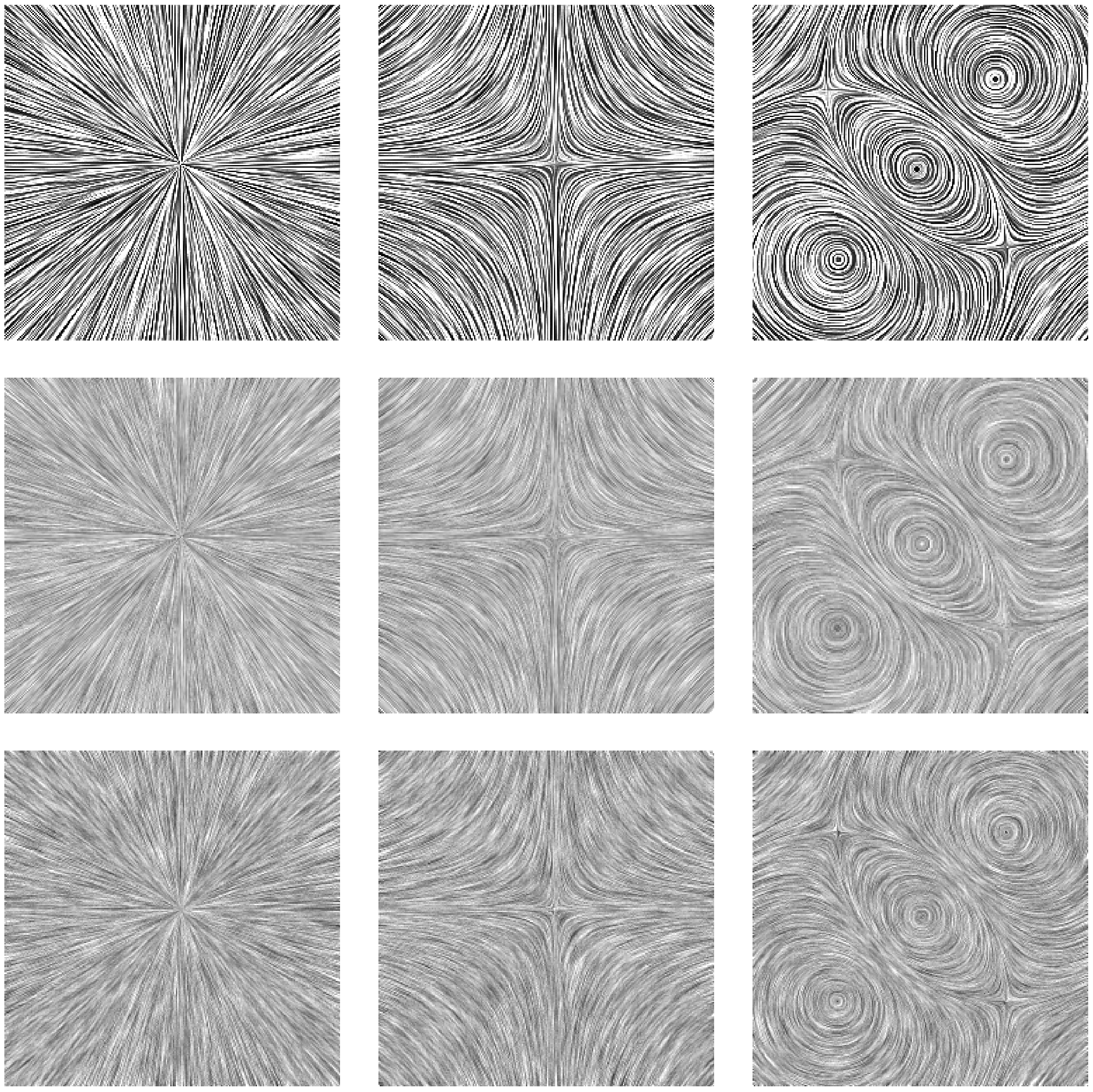

Figure 7: The image in the top row were generated using enhanced LIC, the images in the middle row were generated using the basic LIC, and those on the bottom were generated using PLIC. Left: sink. Middle: saddle. Right: dynvort.

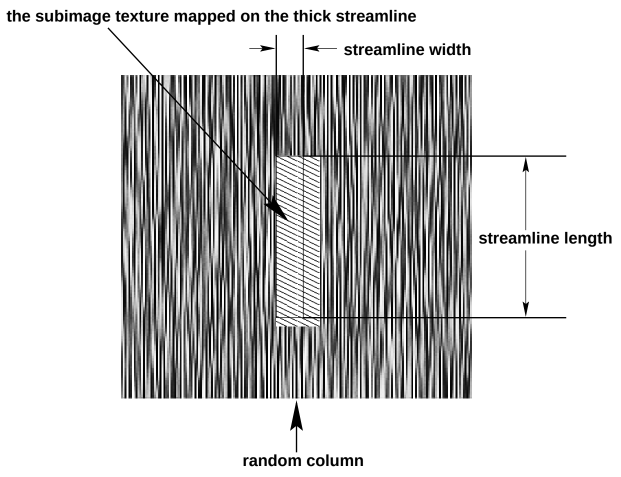

Figure 9: Subimage of the template texture is texture mapped to a thick streamline.

```
  for(i=0; i < max_x; i += stride_x)
      for(j=0; j < max_y; j += stride_y)
      {
          // jitter current seed point
          // position

          (x, y) = jitter_position(i, j);

          s = streamline(x, y);
          c = random column of template-
              texture;

          // texture map subimage of
          // template texture to the
          // thick streamline

          DrawThickStreamline(x, y, s, c, n);
      }

  Divide each pixel by the number of
  streamlines that intersect it;
}
```

## 4 PLIC FEATURES AND ISSUES

The previous section outlined the basic idea of PLIC – generating LIC-like images by texture mapping streamlines. We now focus on some of the parameters associated with this approach.

### 4.1 Effect of Template Texture on PLIC

The template texture in Figure 4 was generated using enhanced LIC with 10 integration steps per streamline. It should be noted that using longer integration steps makes the LIC images smoother. Figure 10 shows three template textures and the corresponding PLIC images. The templates were generated with integration lengths of 10, 20, and 30 respectively. The streamlines themselves were generated using 10 integration steps. It is clear from the images in Figure 10 that a template with longer integration length helps in producing a smoother PLIC image. It is also noteworthy that using a longer integration step template does not increase the computation time of the PLIC image. Hence one can simulate the effect of using large number of integration steps in LIC by simply choosing a template which was generated using appropriately large number of integration steps. For LIC the computation cost increases by a factor of four when the number of integrations are doubled [12]. For PLIC however, there is no increase in computation cost when a template with larger integration steps is used, hence PLIC has a clear advantage over LIC.

### 4.2 Seed Density and Thick Streamlines

Seed placement is very important for the visualizations generated using streamlines. Most flow data is represented on a grid and the grid points are a natural choice to place the seeds to generate streamlines. Unfortunately a simple strategy like this gives poor results because if the seeds are placed regularly on a grid then the underlying grid patterns are visible in the output image as artifacts (see Figure 3(b)). These artifacts serve not only to distract the user of the visualization, they can often lead to mis-interpretation of the flow field.

Since PLIC relies heavily on drawing texture mapped streamlines, we also encountered the problem of finding a good seed placement strategy. The PLIC images in Figure 10 were generated by placing seeds on the grid points such that every third grid point was seeded (stride of three). Since we also draw streamlines thicker (3 pixels wide in Figure 10), and integrate both forward and backwards, the streamlines overlap and leave no area uncovered. Although it is not guaranteed that the whole image will be covered, we have found that in practice this strategy gives satisfactory results. Also, we are using a stochastic texture to texture map the streamlines, hence the underlying grid does not appear as artifacts in the resulting images. We believe it is not necessary that the streamlines should cover the whole area of the image in order to generate a good visualization of the flow. Using PLIC we can generate images with sparse streamlines. Such an image can be seen in Figure 11. It should be noted that this image in no way conveys less information than the corresponding LIC image or the PLIC image where no area of the image was left uncovered. Seed placement strategy becomes important when we decide to draw sparse streamlines. We would like to use the important features in a flow to guide the placement of streamlines so that in areas of importance the image resembles a LIC image, and in area where the flow is not interesting the images looks like streamlines. Thus we would be able to combine the advantages of both LIC and streamlines in a single image. We are currently investigating a seed placement strategy based on the critical points of a flow field.

### 4.3 Reducing Aliasing

We have noticed that when the streamlines are seeded sparsely by sub-sampling the regular grid and drawn thin such that the streamlines do not fully cover the entire area of the image, the underlying grid patterns are visible in the output image. The aliasing effect can be reduced substantially by using a better seed placement strategy. Since the artifacts are a result of a regular seed placement pattern, if the seeds are distributed randomly, then the underlying seed placement pattern will not be visible in the resulting image. Cook's paper on stochastic sampling [2] describes the *Poisson disk distribution*. We have implementation a variation of Poisson disk distribution using jittering as described by Cook. In our implementation, the size of the "disk" is adjustable but is usually set to be equal to or less than the stride value. For example, if we are skipping every third grid point (i.e. generate 1 out of 16 possible streamlines), we typically set the jitter parameter such that the seed lies somewhere in the $7 \times 7$ window around the current grid point. While jittering helps to reduce the artifacts as described above for static images (see Figure 12), remnants of the artifacts are still visible in animations of time-varying data (see accompanying video). Another way to re-

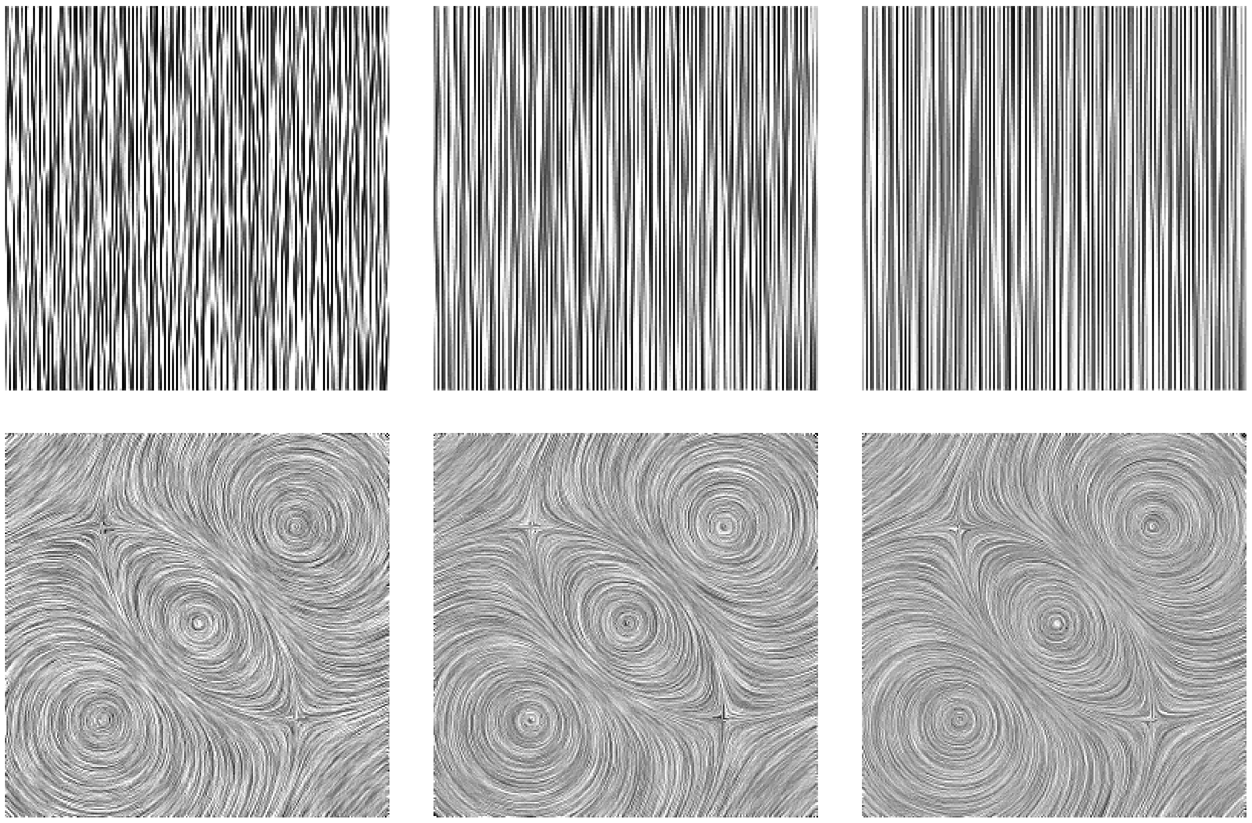

Figure 10: Effect of template texture on PLIC. Top row shows the template texture images and the bottom rows shows the corresponding PLIC images. PLIC images were generated using a stride of 3 and streamline thickness of 3. Left: Integration length of 10. Middle: Integration length of 20. Right: Integration length of 30.

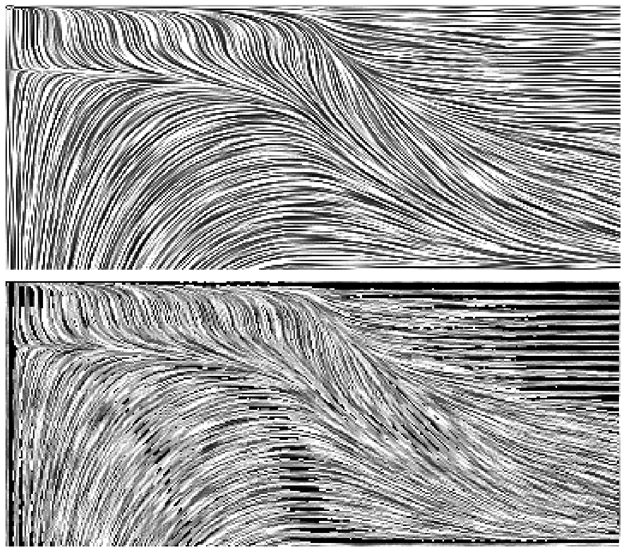

Figure 11: Bluntfin dataset in computational space. Top: Enhanced LIC; Bottom: Sparse PLIC – shows properties of both streamlines and LIC in the same image.

duce the artifacts from the underlying grid is to jitter the length of the streamlines. By jittering the length of the streamlines, we mean that for each streamline the length of the streamline has a Poisson disk distribution. Note that we integrate both in the forward and backward directions to generate our streamlines. Hence, when the length of the streamlines are jittered, the seed pattern is not visible in the output image. Our experiments indicate that the effect of jittering the seed locations is equivalent to the effect of jittering the length of the streamlines. It is sufficient to do only one type of jittering. Figure 12 compares the two types of jittering schemes described above. The streamlines in Figure 12 have a thickness of one and were seeded with a stride of five.

## 4.4 Unsteady Flow

So far we have described how to generate LIC like images for steady flow. We now describe how PLIC can also be used to animate steady and unsteady flows. Animation of steady and unsteady flows is typically achieved in LIC by using periodic filter kernels [5] and colormap cycling [9]. PLIC can also be used to generate animations for steady and unsteady flows. In section 3.2 we described the one dimensional textures used to texture map the streamlines. These one dimensional textures are columns of the template texture image. We choose the template texture image such that its height is larger than length of the streamlines. To animate the flow along a given streamline we cycle through its one dimensional texture to simulate the advection of texture along the streamline. We have included the results of our animation in the accompanying video. The video compares PLIC animation with instantaneous streamlines for unsteady flow using enhanced LIC [12], and UFLIC [14]. Figures 14 and 15 show one time-step from the unsteady datasets dynvort and delta respectively. These images compare PLIC with UFLIC and Enhanced-LIC, and are pseudo-colored by the velocity magnitude. It should be noted that although we have used instantaneous streamlines for our current implementation, PLIC can be easily extended for streaklines and pathlines for a more accurate representation of the flow. PLIC maintains good spatial and temporal coherence. Using periodic filter kernels for pathlines, it is difficult to establish temporal coherence unless the flow is relatively steady [14]. In Cohen and Forsell's method [6], the pathlines from the same seed vary over time. Hence, the same filter with shifted phases is applied over different convolution paths. Since, phase-shift method is effective in creating artificial motion only when the convolution is applied to the same path over time, temporal coherence becomes obscure for unsteady flows. Our method does not suffer from the problem of using periodic filter kernels applied to pathlines because the texture is fixed for each pathline and would follow it from frame to frame. Furthermore, PLIC does not require additional processing like UFLIC [15] to remove the potential problem of textures getting coarser over time.

When we jitter the seed locations we ensure that the seed positions do not change over the duration of the animation. Generating new seed locations for each frame is not desirable in order to maintain temporal coherence.

## 4.5 Variable Speed Animation

It is desirable that the animation should have faster motion where the flow has high velocity and slow where the flow has low velocity. Cabral and Leedom [1] achieve this effect by varying the frequency of the filter function. Forssell and Cohen [6] vary the rate of the filter function phase shift in proportion to the vector magnitude. We can achieve variable speed animation by determining the rate of cycling through the one-dimensional texture in proportion to the velocity magnitude at the seed location for each streamline. In addition, since the template texture is independent of the flow field dimensions, we can increase the dynamic range of the speeds by increasing the number of rows in the template texture. Finally, the relative speeds of different flow regions can either be exaggerated or played down by appropriately mapping velocity magnitude to the speed used for cycling through the texture.

## 4.6 Computation Time for PLIC

In this section we compare the running times of the basic LIC algorithm with PLIC. For LIC we used an integration length of 10. For PLIC, we chose an integration length of 15 with the seeds placed with a stride of 3 and the streamlines drawn to be 3 pixels wide. A side by side comparison (see Figure 10) reveals that using PLIC, with the parameters as described above, the images are better in contrast and the flow lines are more distinct. Experiments were done on an SGI Onyx-2 Infinite Reality Box with 3072 Mbytes of main memory using one 195 MHz IP27 processor. Table 1 shows that PLIC reduces the cost about 45% over the basic LIC method and 71% over enhanced LIC. We have also compared the computation time for PLIC with UFLIC and found that PLIC is about two orders of magnitude faster than UFLIC. It is noteworthy that these savings are greatly increased when sparser seeding is used to generate the streamlines for PLIC.

| | | running times | | |
|---|---|---|---|---|
| Dataset | Image Size | Enhanced-LIC | LIC | PLIC |
| sink | 544 × 544 | 12.67 | 6.33 | 3.63 |
| spiral | 544 × 544 | 12.73 | 6.36 | 3.67 |
| saddle | 544 × 544 | 12.67 | 6.33 | 3.64 |
| dynvort | 501 × 501 | 10.64 | 5.32 | 3.05 |
| bluntfin | 262 × 598 | 6.39 | 3.19 | 1.84 |

Table 1: Comparison of running times (in seconds) for LIC and PLIC.

## 4.7 More Differences

One advantage of LIC that is currently missing in PLIC is that one can have multiple passes of LIC to make the flow lines more well-

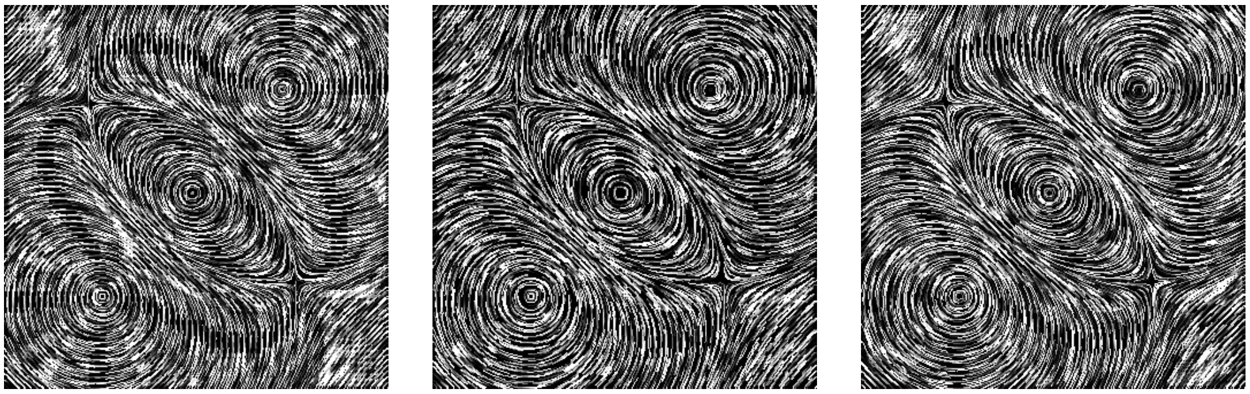

Figure 12: Left: seeds placed at regular grid positions; Aliasing is visible as light and dark bands and become more prominent when animated. Middle: seed locations for the streamlines are jittered; Right: length of the streamlines are jittered.

defined. Enhanced LIC uses two passes of LIC to remove noise, and then applies high pass filter to enhance edges, and finally does histogram equalization to increase image contrast. Our experiments with PLIC have revealed that although the PLIC images are better in contrast and have well defined flow lines than LIC, they still have less contrast than enhanced LIC. We are currently investigating ways to make the PLIC images to have better contrast and better defined flow lines. For example, instead of using an image processing approach like enhanced LIC, PLIC can improve its image contrast by adjusting the template texture, integration length, and stride (sub-sampling rate). In addition, one can always use PLIC instead of LIC as the first pass of enhanced LIC to reduce the computational cost of enhanced LIC.

Another difference is that LIC can accept a color image as its input texture and apply LIC on it. On the other hand, the texture template in PLIC is used purely as a bank of one dimensional textures.

While both PLIC and the motion map approach presented by Jobard and Lefer [9] texture map streamlines, the intent and range of features vary significantly. Our intent is to use streamlines to mimic or simulate the appearance of a LIC image so that we can better understand the parameterization and can do a systematic comparison of these two methods. Note that the streamline-to-LIC mapping that is presented in this paper is but one possible mapping. The intent of the motion map approach is to produce the appearance of motion in steady state flow data much like the phase-shift [5] of the filter kernel to achieve a similar objective. The texture used by PLIC is significantly different than the functional cyclic texture used by the motion map approach. PLIC uses a pre-calculated texture template that is essentially treated as a bank of one dimensional textures randomly selected and mapped to streamlines. Finally, the PLIC approach allows us to visualize time varying data sets using instantaneous streamlines, but which are very similar to UFLIC [15] in appearance. In addition, PLIC can generate variable speed animations with options for adjusting the relative speeds as well as capturing a wide dynamic range of speeds.

## 5 CONCLUSIONS

We have presented a flow visualization method that can be smoothly adjusted to generate LIC-like images on one extreme and streamline-like images on the other. In between, it is also possible to generate images that have properties of both LIC and streamlines. This method also has several advantages over LIC in terms of computation time, image quality, and ease of handling time varying data sets. We see this work as a stepping stone towards being able to systematically compare streamlines and LIC. There are also a number of avenues that we are exploring beyond this initial work: extension of PLIC to 3D; investigate other LIC-to-streamline and streamline-to-LIC mappings; flow guided seed placement strategy; use more accurate pathlines instead of instantaneous streamlines to visualize unsteady flow data; and a comprehensive comparison of flow visualization techniques.

## ACKNOWLEDGEMENTS

We would like to thank the Santa Cruz Laboratory for Visualization and Graphics (SLVG) for the wonderful research environment. This project is supported by NASA grant ncc2-5281, LLNL Agreement No. B347879 under DOE Contract No. W-7405-ENG-48, DARPA grant N66001-97-8900, NSF grant IRI-9423881, and ONR grant N00014-96-1-0949.

## References

[1] B. Cabral and L. Leedom. Imaging vector fields using line integral convolution. *Computer Graphics Siggraph Proceedings*, pages 263–270, 1993.

[2] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1), January 1986.

[3] Wim de Leeuw and Robert van Liere. Comparing lic and spot noise. In *Proceedings of Visualization 98*, pages 359–365. ACM, 1998.

[4] Milton Van Dyke. *An Album of Fluid Motion.* The Parabolic Press, Stanford, California, 1982.

[5] L. K. Forssell. Visualizing flow over curvilinear grid surfaces using line integral convolution. In *Proceedings of Visualization 94*, pages 240–247, CP27. IEEE Computer Society Press, October 1994.

[6] L. K. Forssell and S. 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, June 1995.

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

[8] V. Interrante and C. Grosch. Visualizing 3d flow. *IEEE Computer Graphics and Applications*, 18(4):49–53, July-August 1998.

[9] B. Jobard and W. Lefer. The motion map: efficient computation of steady flow animations. In *Proceedings of Visualization 97*, pages 323–328. IEEE, October 1997.

[10] Bruno Jobard and Wilfrid Lefer. Creating evenly-spaced streamlines of arbitrary density. In W. Lefer and M. Grave, editors, *Visualization in Scientific Computing '97*, pages 43–55. Springer, 1997.

[11] Ming-Hoe Kiu and David C. Banks. Multi-frequency noise for lic. In *Proceedings of Visualization 96*, pages 121–126. ACM, October 1996.

[12] Arthur Okada and David Kao. Enhanced line integral convolution with flow feature detection. In *SPIE Vol. 3017 Visual Data Exploration and Analysis IV*, pages 206–217, February 1997.

[13] Han-Wei Shen, Chris Johnson, and Kwan-Liu Ma. Visualizing vector fields using line integral convolution and dye advection. In *Proceedings of the 1996 Symposium on Volume Visualization*, pages 63–70, 102. ACM, October 1996.

[14] Han-Wei Shen and David L. Kao. UFLIC: a line integral convolution algorithm for visualizing unsteady flows. In *Proceedings of Visualization 97*, pages 317–322, 556. IEEE, October 1997.

[15] Han-Wei Shen and David L. Kao. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):98–108, April-June 1998.

[16] D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. *Computer Graphics Siggraph Proceedings*, pages 249–256, 1995.

[17] Greg Turk and David Banks. Image-guided streamline placement. In *Proceedings SIGGRAPH*, pages 453–460, New Orleans, LA, August 1996. ACM SIGGRAPH.

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

[19] R. Wegenkittl, E. Groller, and W. Purgathofer. Animating flow fields: rendering of oriented line integral convolution. In *Computer Animation '97*, pages 15–21. IEEE Computer Society Press, June 1997.