AUFLIC: An Accelerated Algorithm For Unsteady Flow Line Integral Convolution

ZhanPing Liu Robert James Moorhead II

zhanping@erc.msstate.edu rjm@erc.msstate.edu

NSF Engineering Research Center, Mississippi State University, MS, USA

Abstract

UFLIC (Unsteady Flow Line Integral Convolution) is an effective texture synthesis technique to visualize unsteady flow with enhanced temporal coherence, but it is time-consuming to generate. This paper presents an accelerated algorithm, called AUFLIC (Accelerated UFLIC), to speed up the UFLIC generation. Our algorithm saves, re-uses, and updates pathlines in the value scattering processes. A flexible seeding strategy is introduced so that a seed particle may be directly extracted from the previous scattering processes to make best use of the saved pathline so as to reduce computationally expensive pathline integration calculations. A dynamic activation-deactivation scheme is employed to maintain the fewest necessary pathlines. Avoiding excessive pathlines achieves acceleration and nearly-constant memory consumption. With very low memory cost, AUFLIC cuts UFLIC generation time nearly in half without any image quality degradation.

CR Categories and Subject Descriptors: 1.3.3 [Computer Graphics]: Picture / Image generation; 1.3.6 [Computer Graphics]: Methodology and Techniques; 1.4.3 [Image Processing]: Enhancement.

Keywords: flow field, unsteady flow visualization, pathline, texture synthesis, convolution, LIC, UFLIC.

1. Introduction

Flow visualization is one of the most challenging issues in scientific visualization and plays an important role in computational fluid dynamics simulation, as well as meteorological and oceanographic modeling. Arrows, streamlines, and stream surfaces are straightforward approaches for steady flow visualization. Commodity graphics hardware can be exploited to accelerate primitive rendering to achieve real time visualization. However, inappropriate seed placement may either produce cluttered images or display the field only at a local, discrete, and coarse level. To address this problem, Wijk proposed a texture synthesis technique, spot noise [1], to visualize flow data by distributing tiny spots within the field and transforming them along the underlying vector directions. Furthermore, Cabral and Leedom [2] presented Line Integral Convolution (LIC) to compute each output pixel value by convolving a white noise texture along the windowed streamline symmetrically advected in both directions from the pixel. LIC takes advantage of a synthesized image to provide a global and continuous view with finer details. Since then, there have been many optimizations and extensions to the original LIC method including fast and resolution independent LIC [3], parallel LIC [4], LIC on curvilinear grids [5], LIC on arbitrary 3D polygonal surface [6], magnitude LIC based on multi-frequency noise [7], oriented LIC [8], enhanced LIC with flow feature detection [9], LIC incorporated with dye advection [10] and volume LIC [11, 12].

The increasingly rapid progress in computing power and storage capacity enables numerical simulations of unsteady flow fields which usually involve hundreds of time steps, large-scale data sets, changing features and possibly moving grids. Unsteady flow visualization provides more insight into the evolving flow than can be revealed using instantaneous visualization techniques. Streaklines and pathlines [13] are basic time-dependent solutions to visualize unsteady flow by continuously tracking released particles. Unsteady flow volume [14], the adaptive tetrahedralization of streakline union, is an effective method which can be applied to multi-zoned curvilinear grids. Besides these traditional methods, there are some texture-based techniques for unsteady flow visualization. Forssell [5] extended LIC to unsteady flow fields by using pathlines as convolution paths. Vivek [15] introduced template textures to speed up LIC computations and applied the PLIC method to unsteady flow. Furthermore, Bruno [16] presented a hardware-accelerated texture advection algorithm.

The best known approach for unsteady flow visualization is UFLIC (Unsteady Flow Line Integral Convolution) proposed by Han-Wei Shen [17]. UFLIC uses a time-accurate value-scattering scheme and a successive feed forward strategy to exploit both spatial and temporal correlations. For each *scattering process* which usually covers several time steps, a new seed is released from each pixel and scatters its texture value to the succeeding pixels along the newly advected pathline. A huge amount of UFLIC computation time is poured into intensive pathline integration calculations, which slows down the scattering processes. Up to now there has been little research reported on UFLIC acceleration.

To speed up UFLIC, we present an optimized algorithm called AUFLIC (Accelerated UFLIC). AUFLIC saves, re-uses, and dynamically updates pathlines in the scattering processes to reduce pathline integration calculations to the minimum amount. AUFLIC is based on a flexible seeding strategy that a seed particle, instead of being always released, may be directly exacted from the previous scattering processes to make best use of the saved pathline. A dynamic activation-deactivation scheme which decides whether to save the current pathline or not is used to ensure there is only one seed particle from each pixel. The activation-deactivation scheme also helps maintain nearly-constant memory consumption.

This paper is organized as the follows. We first give an overview of the underlying UFLIC method. Next, we present our AUFLIC algorithm in detail. Then we give some results to demonstrate AUFLIC's acceleration. We finally conclude this paper with some ideas for future work.

2. Background

LIC is an image-space texture synthesis technique which is widely used to visualize steady flow fields. For each pixel of the output image, the correlated pixels are first located along the windowed streamline which is symmetrically advected in both directions from the pixel. The corresponding noise texture values are then convolved to obtain the output pixel value. Spatial coherence along the vector direction is well characterized and visualized using one dimensional low pass filtering. However, LIC can not be directly applied to unsteady flow visualization because pathline convolution fails to support spatial coherence, and on the other hand, the phase-shift technique does not maintain temporal coherence.

UFLIC [17] is an object-space texture synthesis technique which is used for visualizing unsteady flow fields. The two important principles of UFLIC, the time-accurate value-scattering scheme and the successive feed-forward strategy, are based on the obvious physical phenomenon that a flow in the real world advects only along the forward direction over time. Upstream particles deposit their contributions to downstream particles, but not vice versa. For each scattering process, a new seed is released from each pixel and keeps scattering its texture value to the succeeding pixels along the newly advected pathline for several time steps. At the same time, each pixel maintains several convolution ring-buckets associated with different time-stamps to receive scattered values from upstream seed particles. Each frame is obtained by computing the convolution for each pixel in the convolution bucket associated with the corresponding time-stamp. Spatial coherence is therefore maintained. To achieve temporal coherence, each resulting texture is post-processed using a noise-jittered high pass filter and then fed forward as the input texture to the next scattering process. Figure 1 illustrates the UFLIC pipeline.

In the preprocessing stage, vector data is usually clamped and scaled, but not normalized, to produce a desirable visual effect. During several time steps, i.e., the *life span* in computational time, a seed advects a pathline with the length proportional to the particle's velocity magnitude. Generally, tens of succeeding pixels have to be located along the pathline using numerical integration calculations that involve temporal-spatial vector interpolations. In fact, over 90% of UFLIC computation time is spent on intensive pathline integration calculations, which slows down the scattering processes. It is worth mentioning that a large amount of pathline information potentially useful for subsequent scattering processes is neglected. This situation is illustrated in Figure 2 where t, t+1, t+2, t+3, and t+4 denote the consecutive integer computational time steps and the scattering life span is 4 time steps. The active seed particle, S_t, advects a pathline passing through the four key points, i.e., S_{t+1}, S_{t+2}, S_{t+3}, and S_{t+4} at the integer computational time steps. The four key points are not further used by UFLIC as eligible seed particles during the subsequent scattering processes along the existing pathline. Instead, new seed particles are

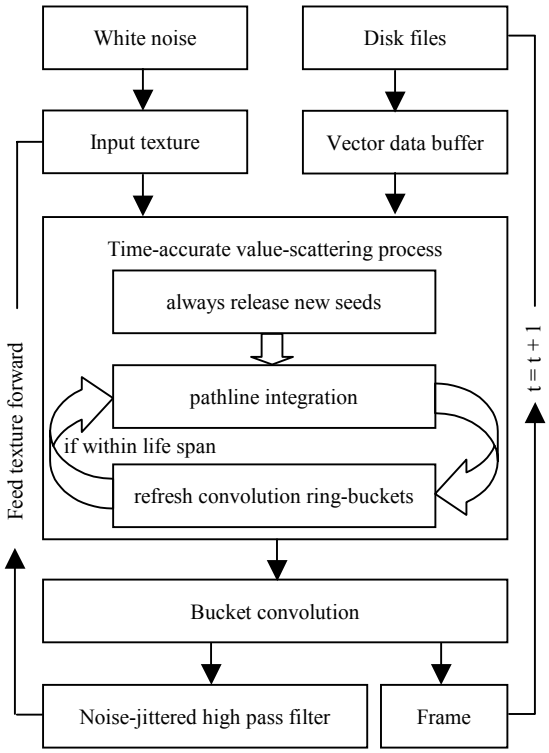


Figure 1: UFLIC pipeline

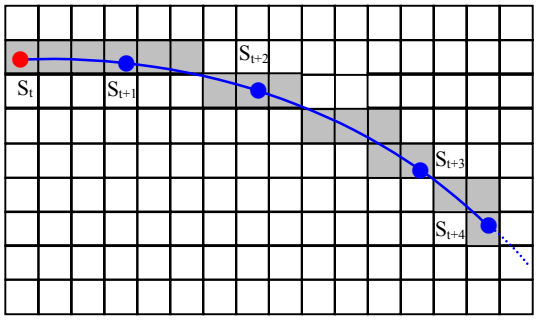


Figure 2: The key points, S_{t+1}, S_{t+2}, S_{t+3} and S_{t+4}, are not used by UFLIC as eligible seed particles during the subsequent scattering processes along the existing pathline advected by S_t, the active seed during the current scattering process.

released again and hence more time has to be consumed to advect the new pathlines.

In the following section, we present an optimized algorithm, AUFLIC, to speed up UFLIC. AUFLIC is based on the flexible seeding strategy by which it is not necessary to release a new seed from the pixel if a particle on the pathline saved in the previous scattering processes can be chosen as the seed. Pathlines are therefore saved, re-used, and updated during the scattering processes. Additionally, we employ a dynamic activation-deactivation scheme to ensure there is no more than one active seed from each pixel in a scattering process. Our algorithm reduces intensive pathline integration calculations to the minimum amount and hence accelerates UFLIC computation.

3. Optimized Algorithm

In this section, we present AUFLIC, an accelerated UFLIC algorithm. We first introduce the flexible seeding strategy followed by the dynamic activation-deactivation scheme. We then discuss how to save, re-use, and update pathlines during the scattering processes. Finally we give the algorithm description.

3.1. Flexible Seeding

The time-accurate value-scattering scheme is aimed at releasing seed particles at a time step and then checking where they will leave their footprints during the life span. The original UFLIC method always releases a new seed particle from each pixel at the beginning of each scattering process (at a time step) even if the seed particles released during the previous scattering processes have not yet died. Additionally, a pathline will be terminated as soon as the seed particle stops value scattering even if the pathline has not run outside the field or encountered any critical points. Suppose the vector field resolution is *nXRes* × *nYRes* and the scattering life span is *N*, there are nearly *N* × *nXRes* × *nYRes* seed particles simultaneously active at time step n (*n* >= *N*-1). However, *nXRes* × *nYRes* seed particles at any time step are just enough to evaluate the flow evolution provided that there is a seed particle from each pixel. It is not necessary to release a new seed if a seed can be directly extracted from the pathlines advected during the previous scattering processes. The extracted seed can take advantage of re-using the saved pathline to reduce integration calculations. In Figure 2, the seed particle S_t will stop the current scattering process when the life span expires. However, S_{t+1}, S_{t+2}, S_{t+3} and S_{t+4}, the key points through which the pathline passes at the integer computational times (time steps) may be further used as seed particles during the following scattering processes. Such key points are called *potential seeds*. They will be either activated or canceled in the subsequent scattering processes by the dynamic activation-deactivation scheme to be discussed in section 3.2. Potential seeds may not be located at pixel centers. The texture value of a potential seed can be obtained by bilinear interpolation or the nearest neighbor access. The flexible seeding strategy is based on as-long-as-possible pathline advection. A pathline will advect forward until one of the following cases occurs:

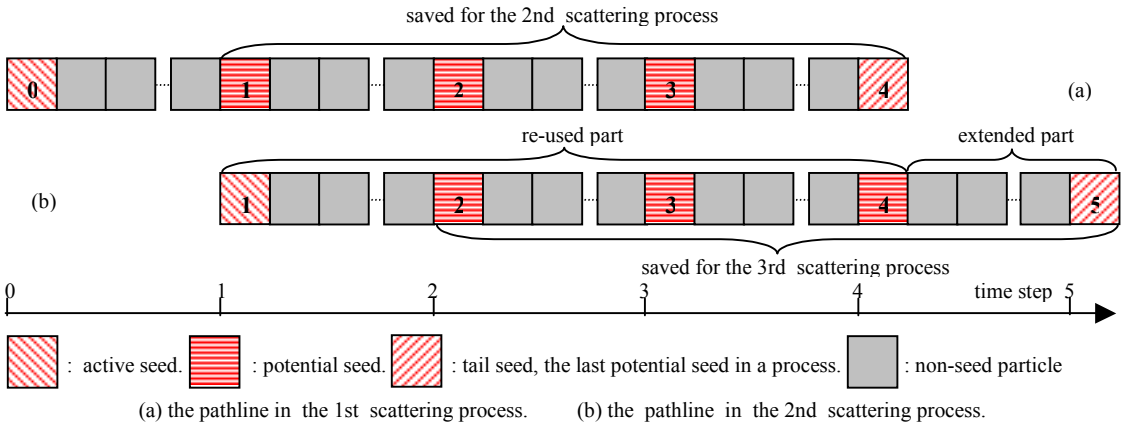


Figure 3: Save, re-use, and update pathlines (value scattering life span = 4 time steps).

- it reaches the field boundaries,
- it encounters a critical point, or
- it is killed by the dynamic activation-deactivation scheme.

3.2. Dynamic Activation-Deactivation

As pathlines are saved, re-used, and advected forward, an unavoidable situation is that there are many potential seeds passing through some pixels simultaneously while there are many pixels not visited by any potential seed. To visualize all flow features, new seed particles have to be released from the un-visited pixels. Gradually there may be more and more seeds simultaneously active in a scattering process, which will inevitably degrade the performance. Also, there will not be enough physical memory to save excessive pathlines.

To deal with this problem, we employ a dynamic activation-deactivation scheme by applying two *mutexes*, *CurMutex* and *NxtMutex*, to each pixel. A mutex has two possible values (0 or 1) for exclusion purposes. A cleared mutex indicates there has been yet no active seed from the pixel while a set mutex means there has already been an active seed from the pixel. A CurMutex implies the pixel status in the ongoing scattering process and determines whether to release a new seed from the pixel or not. A NxtMutex indicates the pixel status in the next scattering process and checks whether to activate a potential seed and hence save the current pathline for the next scattering process or just to cancel it. The CurMutexes are refreshed with the NxtMutexes before each scattering process begins while the NxtMutexes are dynamically updated during the scattering process. Excessive potential seeds can be effectively deactivated by NxtMutexes and the corresponding pathlines are then deleted. The dynamic activation-deactivation scheme helps maintain the fewest necessary pathlines in a scattering process. Excessive pathlines are prevented from consuming memory or affecting acceleration performance. Given the field resolution *nXRes* × *nYRes*, memory consumption can be predetermined because there is never more than *nXRes* × *nYRes* pathlines which need to be saved for any scattering process.

3.3. Save, Re-use, and Update Pathlines

The portion of a pathline advected during one time step is called a *pathlet*. A pathlet begins with an active seed or a potential seed followed by some non-seed particles (pixels). Once a potential seed is activated, the subsequent pathlets previously saved will be directly re-used during the current scattering process without on-the-fly pathline integration. All that needs to be done is to advect the existing pathline forward by only one pathlet. The saved and re-used pathline reduces computationally expensive integration calculations and therefore accelerates the scattering process.

Figure 3 illustrates how to save, re-use, and update pathlines. The CurMutex is cleared at the beginning of the first scattering process and a seed is therefore released from the pixel to advect four pathlets. The seed scatters the texture value (white noise) to the succeeding pixels along the pathline and refreshes their convolution ring-buckets. Once the first potential seed is encountered, the corresponding NxtMutex is checked. If the NxtMutex is cleared, the following three pathlets are saved for the second scattering process as the first scattering process continues. The NxtMutex is then set to refuse other potential seeds and the release of a new seed from the pixel center. All pixel indices and integration step sizes are saved in a pixel ring-buffer. Other information to be saved includes:

- all seeds' buffer-indices: to demarcate pathlets in a pixel ring-buffer by potential seeds and a tail seed.
- pathline status: whether the pathline runs outside the field, meets a critical point, or neither.

- tail seed coordinates: to advect the pathline one pathlet forward during the next scattering process.

Finally, the saved pathline is added to a *pathline list* which maintains those pathlines to be re-used in the following scattering processes.

In the second scattering process, the saved pathline is accessed from the pathline list. The active seed, once the first potential seed in the first scattering process, scatters the noise-jittered high-pass-filtered texture value to the re-used pixels which are directly extracted from the pixel ring-buffer and to the pixels along the extended pathlet which is newly advected by the tail seed, once the last potential seed in the first process. The first potential seed, once the second potential seed in the first scattering process, is checked with the corresponding NxtMutex. If the NxtMutex is set, the pathline is deleted from the pathline list when the scattering process ends. Otherwise, the pathline remains in the pathline list and is updated by deleting the first pathlet and concatenating the newly extended pathlet. The NxtMutex is then updated with a set flag. The tail seed, pixel ring-buffer, and seeds' buffer-indices are updated accordingly.

After the whole pathline list is accessed and updated, new seeds are then released from the un-seeded pixels. As the subsequent scattering processes continue, pathlines are dynamically saved, re-used, and updated in the same routine as described above.

3.4. Algorithm Description

AUFLIC is designed to accelerate the time-accurate value-scattering process by reducing computationally expensive pathline integration calculations to the minimum amount. Pathlines are saved, re-used, and updated using the flexible seeding strategy and the dynamic activation-deactivation scheme. The algorithm is described as the follows.

```
typedef struct tagPATHLINE //pathline information
{struct{int index; float size;} pixels[SIZE];//pixel ring-buffer
 short nSeedIndex[N]; //seed buffer-index: to locate pathlets
 short nTailIndex; //tail buffer-index: to wrap-insert pixels
 short bDead;//vector==0? outside the field? or neither?(0,1)
 struct{float x, y;} mTailSeed; //for further advection
  tagPATHLINE *next; //pointer to the next pathline
} *PATHLINE; //added to the pathline list

void AUFLIC(int nFrameIndex) //produce a frame
{if(nFrameIndex==0){NxtMutexes[]=0; PathlineList=NULL;}
 CurMutexes[]=NxtMutexes[]; NxtMutexes[]=0;
 if (nFrameIndex != 0) /**re-use and update pathlines**/
 { for any pathline from PathlineList
   { extract ActiveSeed from pathline and use LastTexture;
     for(i = pathline->nSeedIndex[0]; i %SIZE < pathline->
     nTailIndex; i++)scatter ActiveSeed to pathline->pixels[i];
     for(i=0; i<N-1; i++) pathline->nSeedIndex[i]=pathline->
     nSeedIndex[i+1]; pathline->nSeedIndex[N-1]= -1;
     int *pNxtMutex= &(NxtMutexes[the first potential seed]);
     if( pathline->bDead == 0 ) //further advect the pathline
     { let pathline->mTailSeed advect pathline for a pathlet;
       while(Δt<1 time step & within the field & vector!=0)
       {integrate 1 step,find a pixel; scatter ActiveSeed to pixel;
         if(*pNxtMutex==0)//remain active in the next process
           { wrap_insert pixel; if (pixel is a potential seed)
             {pathline->mTailSeed=pixel;save tail buffer-index;}
           }
       } //END while( Δt<1……)
     } //END if( pathline->bDead == 0 )
    if( (*pNxtMutex==1) || ( pathline->nSeedIndex[0]== -1) )
    delete pathline from PathlineList; else *pNxtMutex = 1;
  } //END for any pathline from PathlineList
} //END if (nFrameIndex != 0)
 for(any pixel with CurMutex==0) /**release new seeds**/
{ if (nFrameIndex==0)use WhiteNoise; else use LastTexture;
  release a NewSeed from the pixel; pathline = NULL;
  while(within the value scattering life span)
  { if (vector==0) {if (pathline) pathline->bDead=1; break;}
    integrate and check potential seeds, then enter a pixel;
    if(out of the field){if(pathline)pathline->bDead=1; break;}
    scatter NewSeed texture value to pixel;
    if((pixel is 1st potential seed)&(pixel's NxtMutex== 0))
     { pixel's NxtMutex=1; pathline = new PATHLINE;
       pathline->nSeedIndex[0]=0;
       pathline->nSeedIndex[1…N-1]=-1;pathline->bDead=0;
       save pixel to pathline; insert pathline to PathlineList;
     } else if(pathline != NULL) { save pixel to pathline;
       if(pixel is a potential seed) save seed's buffer-index;
       if(pixel is a tail seed) save tail seed's coordinates;
     } //END if(…)
   } //END while(within the value scattering life span)
 } //END for(any pixel with CurMutex == 0)
  bucket convolution; output frame = resulting texture;
  LastTexture=NoiseJitteredHighPassFilter(resulting texture);
} //END AUFLIC
```

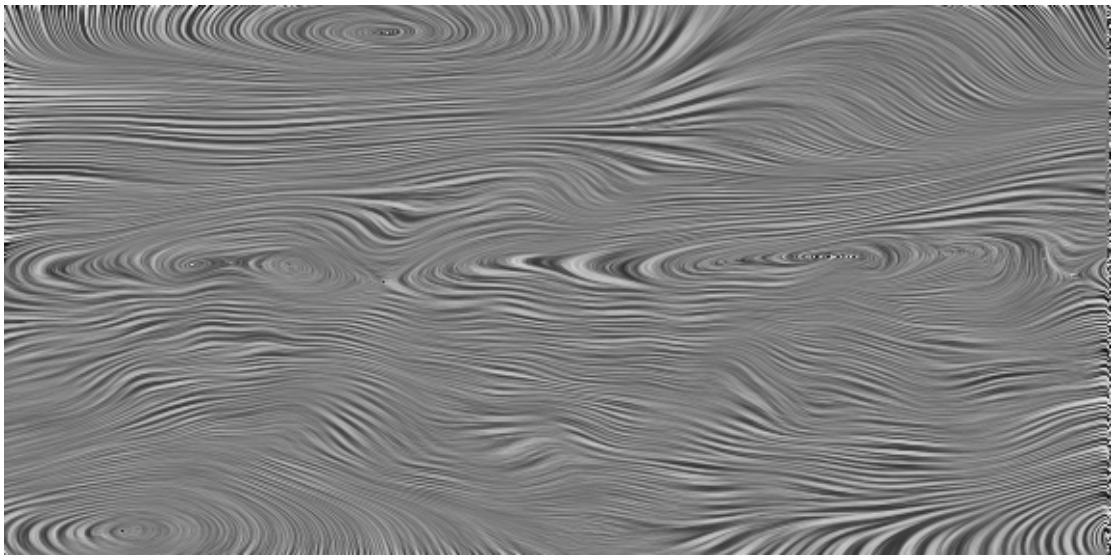
4. Results And Discussions

To demonstrate AUFLIC acceleration, we apply both UFLIC and AUFLIC to two 2D unsteady flow data sets. The weather data set has 41 time steps and the resolution is 576 × 291. The vortex data set has 101 time steps and the resolution is 397 × 397. The machine used for the experiments is a SGI Onyx2 with four 400MHZ MIPS R12000 processors and 4GB memory.

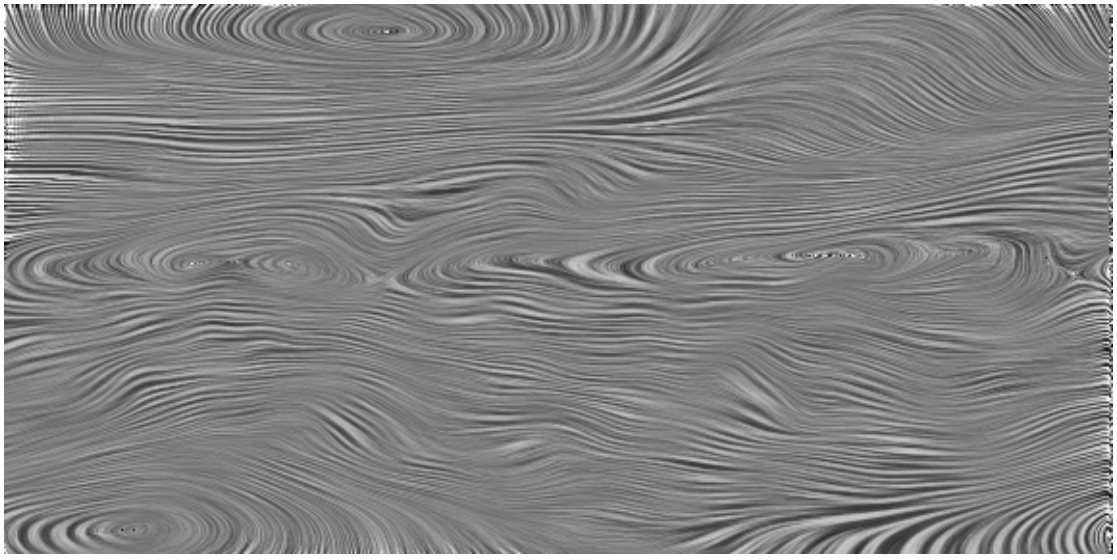
AUFLIC is designed to speed up UFLIC computation by accelerating the time-accurate value-scattering process in the UFLIC pipeline (Figure 1). AUFLIC and UFLIC use exactly the same code for data loading, bucket convolution, noise-jittered high pass filtering, color mapping, and image output, but not for value scattering. In the data loading process, we do not normalize vectors in order to produce a desirable visual effect. Instead, we scale and clamp the

original data to a limited magnitude range so that a seed particle scatters the texture value to a limited number of succeeding pixels along the pathline during the life span. A maximum number is therefore obtained for the AUFLIC pixel ring-buffer size. In all the experiments, we set the life span to 4 time steps for both UFLIC and AUFLIC. AUFLIC pixel ring-buffer size is set to 72. For the two unsteady flow data sets, 37 frames and 97 frames are produced respectively.

Figure 4 shows the images of the unsteady weather data set produced by UFLIC and AUFLIC respectively (see color section). Figure 5 shows the images of the unsteady vortex data set generated by UFLIC and AUFLIC respectively (see color section). All the images are color mapped based on vector magnitude with blue being lowest and red highest. The images produced by AUFLIC are nearly the same as those produced by UFLIC. No flow features in the UFLIC images



(a) UFLIC image.



(b) AUFLIC image.

Figure 4: AUFLIC compared with UFLIC in image quality. Unsteady weather data set (576 × 291).

are missing in the AUFLIC images and no additional artifacts are introduced by AUFLIC. The reason is that AUFLIC saves all that is needed in the scattering process: the contributed pixels along the pathline and the corresponding integration step sizes as the convolution weights. Furthermore, the fading effect factors can also be computed from the saved integration step sizes. AUFLIC uses the flexible seeding strategy, so seed particles extracted directly from saved pathlines may not be located at pixel centers and therefore they are not evenly distributed in a scattering process.

(a) UFLIC image.

(b) AUFLIC image.

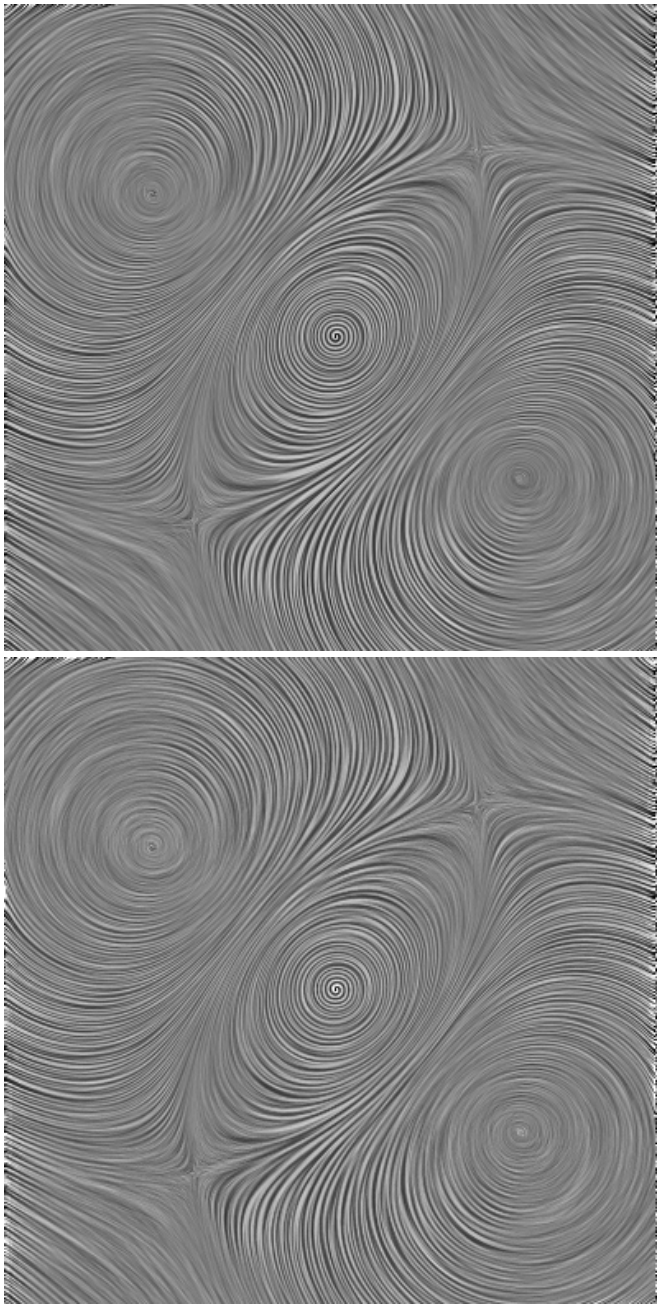


Figure 5: AUFLIC compared with UFLIC in image quality. Unsteady vortex data set (397 × 397).

Method	Load data	Scattering	Convolution	Filtering	Color & Output	Total	AUFLIC/UFLIC
UFLIC	5.00	635.35	3.77	3.37	1.06	648.55	52.94 %
AUFLIC	4.77	330.35	3.86	3.43	0.93	343.34	

Resolution: 576 × 291. Frames: 37. Life span: 4 time steps. AUFLIC pixel ring-buffer size: 72.

Table 1: AUFLIC compared with UFLIC in breakdown of the computation time (in seconds) for the weather data set.

Method	Load data	Scattering	Convolution	Filtering	Color & Output	Total	AUFLIC/UFLIC
UFLIC	13.19	1551.17	9.32	8.42	2.39	1584.49	56.96 %
AUFLIC	12.58	869.89	9.05	8.64	2.35	902.51	

Resolution: 397 × 397. Frames: 97. Life span: 4 time steps. AUFLIC pixel ring-buffer size: 72.

Table 2: AUFLIC compared with UFLIC in breakdown of the computation time (in seconds) for the vortex data set.

Method	AUFLIC	UFLIC
released pathlines	1762823	6201792
release percentage	28.42 %	100 %
un-reused pathlines	568155	6201792
re-used pathlines	1194668	0
re-use percentage	67.77 %	0 %
total re-use times	4438969	0
average re-use times	3.72	0
deactivated pathlines	561402	0
total pixels	146728381	309044731
average pathline length	83.23pixels	49.83pixels

(a) Weather data set.

Method	AUFLIC	UFLIC
released pathlines	3500366	15288073
release percentage	22.90 %	100 %
un-reused pathlines	1263664	15288073
re-used pathlines	2236702	0
re-use percentage	63.90 %	0 %
total re-use times	11787707	0
average re-use times	5.27	0
deactivated pathlines	1511223	0
total pixels	357898668	739725898
average pathline length	102.25pixels	48.39pixels

(b) Vortex data set.

*: deactivated pathlines: those re-used and later deleted from the pathline list by the dynamic activation-deactivation scheme.

Table 3: Pathline statistics for UFLIC and AUFLIC.

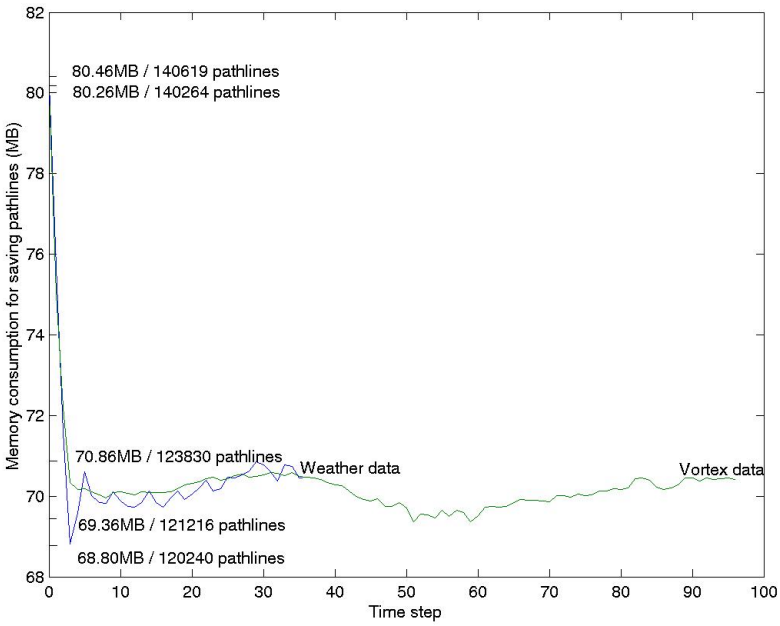


Figure 6: The nearly-constant and low memory consumption of AUFLIC for saving pathlines.

Fortunately, we have not noticed any side effects even without using texture interpolation. We have also considered saving only pixel indices without integration step sizes. For this case, value scattering is simplified by using a unit weighting scheme. Greater acceleration can be achieved and less memory is needed while the images produced are still comparable with UFLIC images. Note that all the experiments here are based on the standard AUFLIC.

Table 1 and Table 2 show the breakdowns of the computation time used by UFLIC and AUFLIC for visualizing the two data sets. Theoretically, AUFLIC should consume the same amount of time as UFLIC in data loading, bucket convolution, noise-jittered high pass filtering, color mapping and image output, because they use the same code for these processes. However, AUFLIC uses much less time in the scattering process than UFLIC. As for the whole pipeline, AUFLIC cuts UFLIC generation time nearly in half without any image quality degradation.

Table 3 shows various statistics for both UFLIC and AUFLIC. Compared with UFLIC, AUFLIC releases significantly less pathlines because many pathlines are re-used for several times. The pathlines are therefore advected longer than those in UFLIC. The dynamic activation-deactivation scheme effectively prevents excessive pathlines from consuming memory or degrading the performance. Figure 6 illustrates the additional memory AUFLIC requires for saving pathlines. Since we use a fixed-size pixel ring-buffer to save a pathline, the memory cost is proportional to the number of the saved pathlines which is never more than the resolution ($nXRes \times nYRes$) of the field because of the dynamic activation-deactivation scheme. Thus the memory requirements are reasonable. Generally, after three or four time steps (ie., after three or four scattering processes), memory consumption is small and varies little. The small memory footprint makes it possible to apply AUFLIC to large-scale unsteady flow visualization.

5. Conclusions And Future Work

We have presented AUFLIC, an Accelerated Unsteady Flow Line Integral Convolution algorithm, to speed up UFLIC computation for unsteady flow visualization. Our optimized algorithm uses a flexible seeding strategy and a dynamic activation-deactivation scheme to save, re-use, and update pathlines. A large amount of computationally-expensive integration calculations are avoided and the time-accurate value-scattering process is therefore accelerated. AUFLIC can produce the same quality images using half the time of UFLIC at very low memory cost.

As for future work, we would like to further enhance AUFLIC and apply it to three-dimensional unsteady flow data sets. How to efficiently encode pathline information to reduce memory cost is a challenging problem for three-dimensional AUFLIC.

Acknowledgments

This work was supported by the High Performance Visualization Center Initiative (HPVCI) funded by the DoD High Performance Computing Modernization Program (HPCMP). We would like to thank Dr. Han-Wei Shen from the Ohio State University for his help with UFLIC questions, implementation details, and valuable suggestions. Additionally, we would like to thank Dr. Rani Samtany and Dr. Han-Wei Shen for the unsteady vortex flow data set.

References

1. Jark J. Van Wijk. Spot Noise - Texture Synthesis for Data Visualization. *Computer Graphics*, Vol.25, No.4, pp.309-318, July 1991.

2. Brian Cabral and Leith (Casey) Leedom. Imaging Vector Fields Using Line Integral Convolution. *ACM SigGraph 93 Proceedings*, August 2-6, Anaheim, California, pp.263-270, 1993.

3. Detlev Stalling and Hans-Christian Hege. Fast and Resolution Independent Line Integral Convolution. *ACM SigGraph 95 Proceedings*, August 6-11, Los Angeles, California, pp.249-256, 1995.

4. Detlev Stalling, M. Zockler, and Hans-Christian Hege. Parallel Line Integral Convolution. *Proceedings of The First Eurographics Workshop on Parallel Graphics and Visualisation*, 26-27 September, Bristol, U.K., pp.111-128, 1996.

5. Lisa 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*. Vol.1, No.2, pp.133-141, June 1995.

6. Christian Teitzel, Robert Grosso, and Thomas Ertl. Line Integral Convolution on Triangulated Surfaces. *Proceedings of Fifth International Conference in Central Europe on Computer Graphics and Visualization (WSCG97)*, Feb, 10-14, Plzen-Bory, Czech Republic, pp.572-581, 1997.

7. Ming-Hoe Kiu and David C. Banks. Multi-Frequency Noise for LIC. *IEEE Visualization 96 Proceedings*, Oct 27–Nov 1, San Francisco, California, pp.121-126, 1996.

8. R. Wegenkittl, E. Groller, and W. Purgathofer. Animating Flow Fields: Rendering of Oriented Line Integral Convolution, *Computer Animation 97*, pp.15–21, 1997.

9. A. Okada and D. L. Kao. Enhanced Line Integral Convolution with Flow Feature Detection. *Proceedings of IS & T / SPIE Electronics Imaging 97*, Feb 8-14, San Jose, California, Vol. 3017, pp.206-217, 1997.

10. Han-Wei Shen, C. Johnson, and Kwan-Liu Ma. Visualizing Vector Fields using Line Integral Convolution and Dye Advection. *Proceedings of 1996 Symposium on Volume Visualization*, pp.63-70, 1996.

11. Victoria Interrante and Chester Grosch. Strategies for Effectively Visualizing 3D Flow with Volume LIC. *IEEE Visualization 97 Proceedings*. Oct 19 – 24, Phoenix, Arizona, pp.421-424, 1997.

12. C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl. Interactive Exploration of Volume Line Integral Convolution Based on 3D-Texture Mapping. *IEEE Visualization 99 Proceedings*, Oct 24-29, San Francisco, California, pp.233-240, 1999.

13. David A. Lane. Visualization of Time-Dependent Flow Fields. *IEEE Visualization 93 Proceedings*, Oct 25-29, San Jose, California, pp.32-38, 1993.

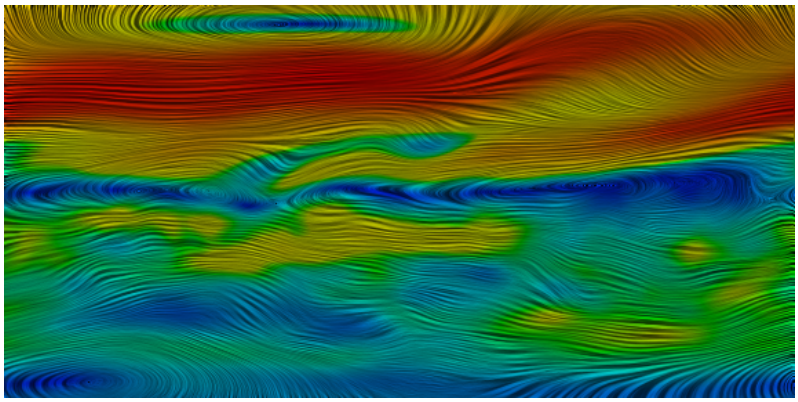
14. Barry G. Becker, David A. Lane, and Nelson L. Max. Unsteady Flow Volumes. *IEEE Visualization 95 Proceedings*, Oct 29-Nov 3, Atlanta, Georgia, pp.329-335, 1995.

15. Vivek Verma, David Kao, and Alex Pang. PLIC: Bridging the Gap Between Streamlines and LIC. *IEEE Visualization 99 Proceedings*, Oct 24-29, San Francisco, California, pp.341-348, 1999.

16. Bruno Jobard, Gordon Erlebacher, M. Yousuff Hussaini. Hardware-Assisted Texture Advection for Unsteady Flow Visualization. *IEEE Visualization 2000 Proceedings*, Oct 8 – 13, Salt Lake City, Utah, pp.155-162, 2000.

17. 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*, Vol.4 No.2, pp.98-108, April-June, 1998.

(a) UFLIC image.



(b) AUFLIC image.

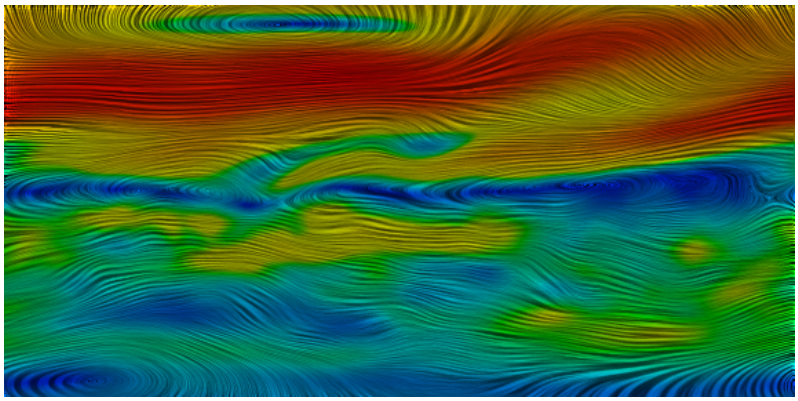
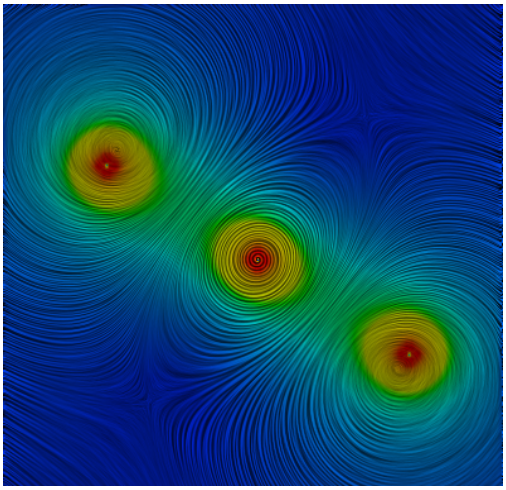
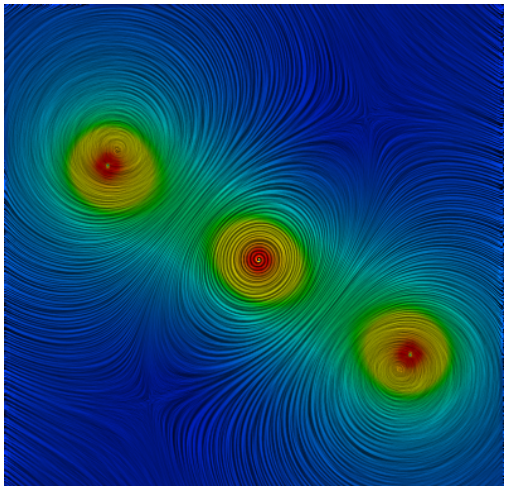


Figure 4: AUFLIC compared with UFLIC in image quality. Unsteady weather data set (576 × 291).



(a) UFLIC image



(b) AUFLIC image

Figure 5: AUFLIC compared with UFLIC in image quality. Unsteady vortex data set (397 × 397).