Highly Parallel Vector Visualization Using Line Integral Convolution

Brian Cabral and Casey Leedom *

Abstract

Line Integral Convolution (LIC) is an effective imaging operator for visualizing large vector fields[1]. It works by blurring an input image along local vector field streamlines yielding an output image. LIC is highly parallelizable because it uses only local read-sharing of input data and no write-sharing of output data. Both coarse- and fine-grained implementations have been developed. The coarse-grained implementation uses a straightforward row-tiling of the vector field to parcel out work to multiple CPUs. The fine-grained implementation uses a series of image warps and sums to compute the LIC algorithm across the entire vector field at once. This is accomplished by novel use of high-performance graphics hardware texture mapping and accumulation buffers.

1 Introduction

Line Integral Convolution (LIC) is used to visualize large vector fields[1][1]. The basic notion presented in [1] is to render an output scalar pixel value for each vector in the vector field. The output pixel value encodes the local vector field by filtering (convolving) a set of input pixels which lie under a local streamline. The result is an output image that looks like the input image, but blurred in the directions of the vector field. When the input image is white noise, the resulting output image looks like fine hair running in the direction of the input vector field streamlines. A white noise input images is an especially effective input image for this application of LIC because it is isotropically uncorrelated. The convolution performed by LIC effectively correlates pixels which lie along streamlines and leaves pixels uncorrelated transverse to the streamlines.

The convolution is performed at each point, p, in the vector field, $V(p)$, as follows: a parametric curve, $P(p, s)$, is created in the vector field by locally following the vector field forward and backward for some distance, L. The parametric curve is laid over the corresponding input image pixels, producing a pixel valued function, $F(P(p, s))$. $F(P(p, s))$ is then convolved with a filter kernel, $k(s)$, to produce the Line Integral Convolution. Finally, the output pixel, $F'(p)$, is normalized by the area under the convolution kernel to

*Silicon Graphics Computer Systems, cabral@sgi.com and leedom@sgi.com, 2011 North Shoreline Blvd. M-405, Mountain View, CA 94043-1389

[1] LIC is a general purpose imaging operator which is also used to generate textures and to produce novel special effects[1].

produce an even average brightness in the output image:

$$F'(p) = \frac{\int_{-L}^{L} F(P(p,s))k(s)ds}{\int_{-L}^{L} k(s)ds} \tag{1}$$

Depending on the shape of the filter, the input image, the input vector field and several control parameters, a wide variety of results are obtained (see [1]).

LIC is inherently local in nature resulting in highly parallel implementations. Two different methods of parallelizing the LIC algorithm have been developed. The first method is a coarse-grained implementation which uses a straightforward partitioning of the vector field to parcel out work to multiple CPUs. The second method is a fine-grained implementation which uses a series of image warps and sums to compute the LIC algorithm across the entire vector field at once.

2 Coarse-grained parallel implementation

Equation (1) is rewritten in discrete form:

$$F'(p) = \sum_{i=0}^{l} F(P_i)h_i \;/\; \sum_{i=0}^{l} h_i \tag{2}$$

Where P_i is the i^{th} vector field position along the parametric curve $P(p, s)$, h_i the analytical area under the portion of the kernel which spans P_i, and $l+1$ is the number of vector field positions on the parametric curve. Equation 2 is written in algorithmic form as:

For each p vector field position
  $F'(p) = 0$
  For each P_i, defined over the local vector streamline at p
    $F'(p) = F'(p) + F(P_i)h_i$
  $F'(p) = F'(p) / \sum_{i=0}^{l} h_i$

The above algorithm leads to a very simple parallelization scheme which involves coarse-grain parallelizing the outer loop. Given LIC's local nature and lack of communication contention, the only parallelization issues necessary to consider are processor cache locality and per-set overhead.

The large amount of per pixel computation in LIC tends to hide cache latency. Several tiling decompositions were tested on a 100MHz R4400 SGI Indigo. The decompositions were designed to provide optimal cache locality for the algorithm. The test results showed no significant difference between these "optimal" decompositions and naive end-to-end traversal of the vector field. Even in the worst-case non-local ordering of vector cell computation, LIC experienced less than a 10% performance penalty over tiling decompositions. A row-tile decomposition was chosen because of its simplicity.

Per-set overhead is largely a function of the operating system's cost for spawning and reaping threads. Since this is a function of the operating system and machine it is difficult to calculate analytically. Measurements of an SGI Challenge with 20 150MHz R4400 CPUs showed an average of 0.17 seconds to spawn and reap 20 threads with very little deviation. Each of these threads performed all the setup for determining the appropriate row-stripe to compute and then exited without executing the LIC algorithm. The effect of this overhead

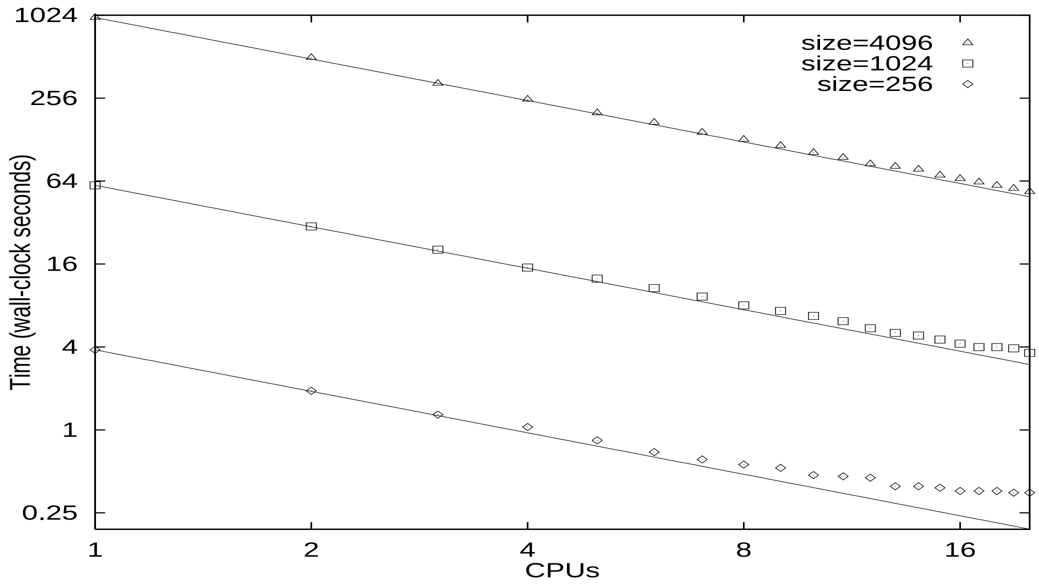

FIG. 1. *Log-log performance results for coarse-grained parallel implementation of LIC.*

on the smallest problem sets can be seen in the performance results shown in Figure 1. Figure 1 shows performance results for executing the coarse-grained parallel implementation of LIC on a set of vector fields ranging in size from 256x256 to 4096x4096. These tests were run on an SGI Challenge with 20 150MHz R4400 CPUs and 1Gb of 8-way interleaved memory. The lines under the timing samples show maximum theoretical linear speedup. As can be seen, this implementation of LIC achieves near-linear speedup for the large 4096x4096 vector field. The smallest 256x256 vector field shows the overhead of spawning and reaping 20 threads. A pure linear speedup of the single CPU time for this problem on 20 CPUs would result in a time of 3.81s/20 = 0.19s. Instead, the fastest measured execution time for 20 CPUs was 0.35s.

3 Fine grain parallelization

An alternate approach is to reverse the order of the loops so that the parallelization occurs across the LIC operation itself. This can be thought of as a series of input image warps and sums. On high-performance image rendering machines, such as the SGI RealityEngine^{TM}, both the image warping and summing are performed in a massively parallel fashion by the texture mapping and frame buffer hardware. The image warping is performed by texture mapping a triangle mesh with the input image. The significant parallelization issue then becomes how fine a warping mesh is needed to accurately perform LIC. If the mesh is made too fine, then the time it takes to advect the mesh is preponderant. Too coarse a mesh will inaccurately approximate the vector streamline component of the LIC algorithm.

This version of LIC reverses the order of the loops found in the coarse-grained algorithm. It renders all the pixels each outer loop iteration, advecting the mesh as it proceeds. The hardware accumulation buffer is used to sum all the pixels at each iteration[2, 3]. Upon termination of the outer loop the accumulation buffer is scaled by the number of iterations; effectively normalizing all the pixels in the frame buffer. Because all the pixels are equally scaled this technique is less general than what was presented in [1].

The fine-grained version of equation (2) is written algorithmically as:

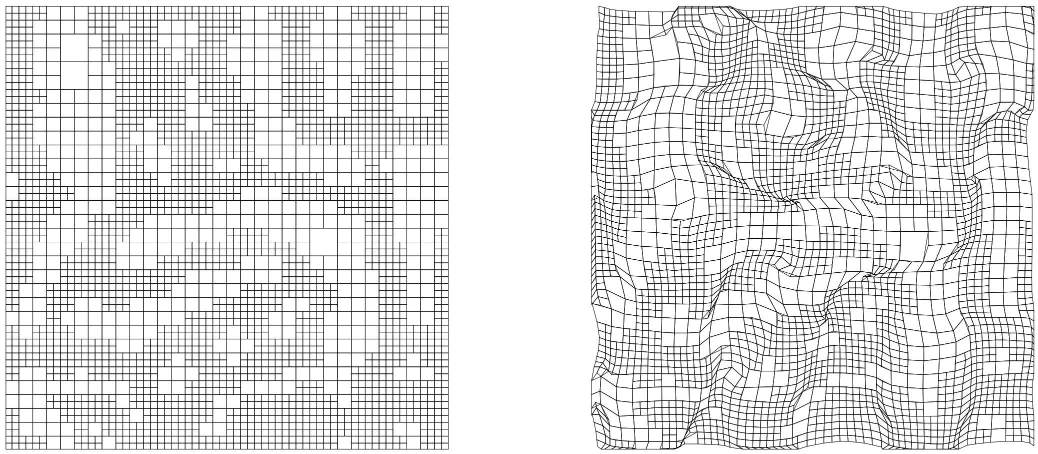

FIG. 2. *The mesh used created for the turbulent vector field imaged in figure 3. The geometric mesh is on the left. The texture coordinate mesh is shown on the right after ten advection steps.*

$F' = 0$
For each i from 0 to $2L\sqrt{2}$
  advect mesh for all vector streamlines
  For each p position if the vector field
    $F'(p) = F'(p) + F(P_i)h_i$
  $F'(p) = F'/2L\sqrt{2}$

This reformulation exploits the ability of the graphics hardware to render textured triangle meshes at extremely high speed[3]. The texture used during rendering is the LIC input image; the mesh represents sample points in the vector field. This implies that the vector field varies linearly over the faces of the mesh triangles. If this assumption is violated then the fine-grained LIC algorithm will produce results which are inconsistent with the general LIC algorithm. The degree of inconsistency is a function of mesh density and is described in more detail below.

3.1 Texture mapping approach

The algorithm maintains a texture and a geometric coordinate for each mesh vertex. The texture coordinate is used by the hardware rendering system to index into the input image. The geometry coordinates are used to specify the screen position of the mesh vertices. The texture coordinates are advected each step of the outer loop, effectively warping the input image onto the fixed geometric mesh. Figure 2 illustrates both the geometric and texture coordinate meshes. The texture coordinate mesh is shown after ten forward advection steps on the field shown in Figure 3.

The mesh advection is done exactly as is the streamline advection described in [1]. Since each vertex travels a different distance the outer loop must iterate enough times so that all the mesh vertices get advected a distance L. The implementation which generated the image in Figure 3 set the iteration count to $L\sqrt{2}$. This produces results which are very similar to the general LIC algorithm. The differences in the images occur in regions of high curvature in the vector field where the mesh is insufficiently fine to accurately represent the

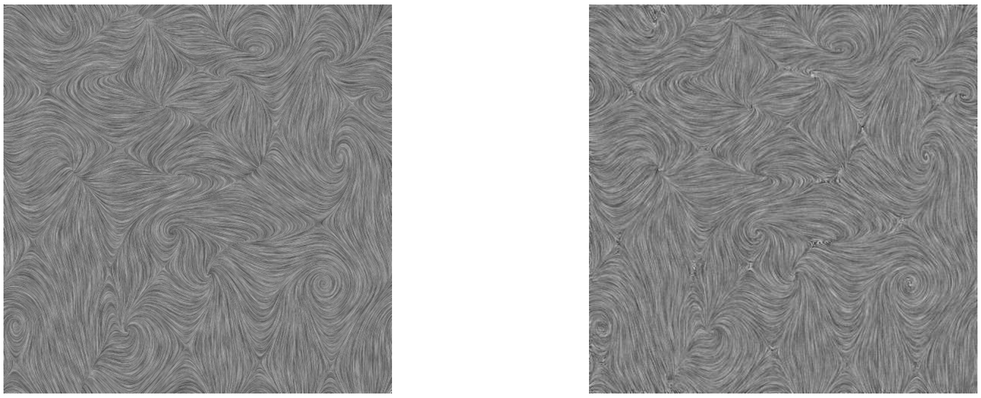

FIG. 3. *A band limited synthetic turbulence field rendered on the left using the coarse-grained algorithm and by the fine-grained version on the right.*

vector field.

3.2 Meshing

The fine-grained algorithm is structurally independent of the meshing scheme. It is designed around a generic mesh list which indexes into an arbitrary vertex list. The idea is to allow the testing of different meshing schemes without altering the basic form of the algorithm.

Since the focus is parallelization we opted for a simple recursive quad-subdivision meshing scheme. During the initialization phase the local vector field curvature is computed for each point in the input vector field. A mesh quad is subdivided if the integral of the curvature of the quad is greater than a given threshold and the quad is larger than a fixed size. The latter constraint is to prevent the meshing algorithm from overmeshing an area which contains a singularity. The subdivision algorithm builds a linear list of quads. Each of the four quad vertices points into a list of texture vertices. The vertex list is kept separate so that the advection algorithm only has to advect each vertex once. If the vertex values were maintained directly in the quads there would be many more vertices to advect, most of which would be redundant.

3.3 Textured rendering results

The performance of the fine-grained parallel implementation of the LIC algorithm is directly tied to the number of mesh vertices which are advected. As the mesh vertex count increases the algorithm is increasingly limited by CPU-time to advect the mesh. Figure 4 graphs wall-clock time versus mesh size, for a 1024x1024 vector field[2]. The flat part of the center curve shows that CPU time to advect the mesh quickly becomes preponderant. The time it takes to actually texture map or render the mesh is negligible. The performance of the fine-grained algorithm compares favorably to the coarse-grained implementation on 15+ processors.

The primary initialization overhead consists of the constant amount of time it takes to compute the curvature integration table. It is worth noting that the creation of the subdivided mesh absorbs little of the overall run time.

[2] All tests were run on a 4 Raster Manager RealityEngine with a 150 MHz R4400 cpu.

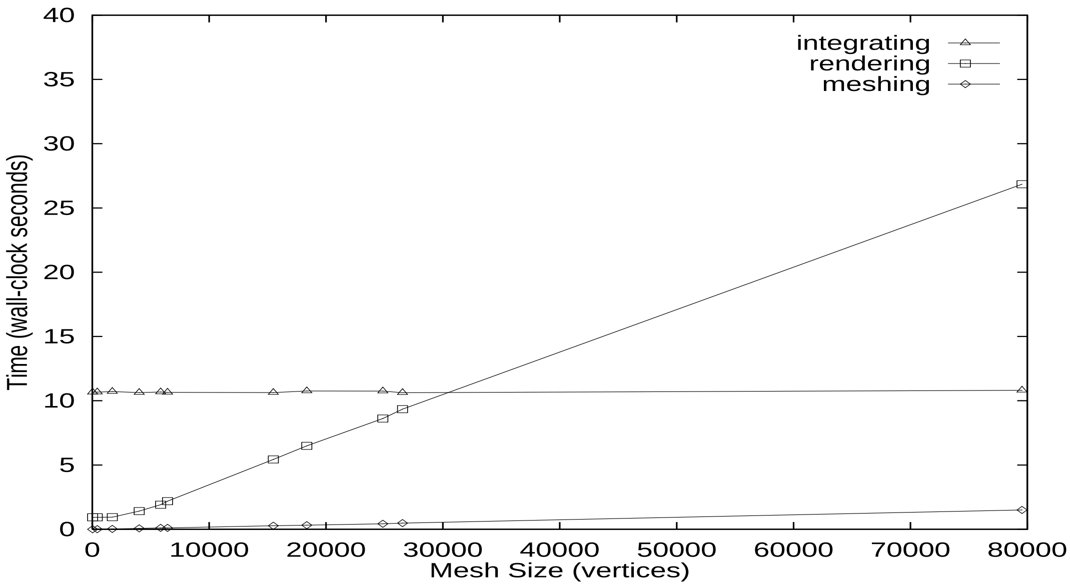

FIG. 4. *Performance of the fine-grained parallel LIC algorithm as a function of mesh vertices.*

4 Conclusions

The local nature of the LIC algorithm results in a near linear speedup for the coarse-grain parallel implementation of LIC. This is important since the vector fields LIC is used to visualize are often large and dense.

High-performance graphics imaging systems like the SGI RealityEngine provide programmable, massively parallel pixel and geometric imaging primitives. These facilities can be used in a general purpose manner to implement algorithms like LIC[4]. Since such hardware is often purchased for scientific visualization it means that the hardware can further be leveraged to visualize vector fields rapidly, possibly in real-time.

5 Acknowledgements

The coarse grain parallelization and initial algorithm development was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract W-7405-ENG-48. The fine grain parallelization work was done by both authors while at Silicon Graphics Computer Systems.

References

[1] Cabral, B. and Leedom, C. Imaging Vector Fields Using Line Integral Convolution. *Computer Graphics.* (August 1993), 263-270.

[2] Haeberli, Paul and Akeley, Kurt. “The Accumulation Buffer: Hardware Support for High-Quality Rendering.” Proceedings of SIGGRAPH ’90 (Dallas, Texas, August 6-10, 1990). In Computer Graphics 24, 4 (August 1990), 309-318.

[3] Akeley, Kurt. RealityEngine Graphics. Rendering. *Proceedings of SIGGRAPH ’93* (Anaheim, California, August 1-6, 1993). In Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, New York, 1993, pp. 109-116.

[4] Cabral, B., Foran, J., and Cam, N. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. *Proceedings of Visualization ’94 Symposium on Volume Visualization.* (Washington D.C., October 18-20, 1994).