MATLAB TOOLBOX FOR HIGH RESOLUTION VECTOR FIELD VISUALIZATION WITH APPLICATION IN IMPROVING THE UNDERSTANDING OF CRACK PROPAGATION MECHANISMS

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computational Science

---

by

Nima Bigdely Shamlo

Fall 2005

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Master Thesis of Nima Bigdely Shamlo:

Matlab Toolbox for High Resolution Vector Field Visualization with Application

in Improving the Understanding of Crack Propagation Mechanisms

ABSTRACT OF THE THESIS

Matlab Toolbox for High Resolution Vector Field Visualization
with Application in Improving the Understanding of Crack
Propagation Mechanisms
by
Nima Bigdely Shamlo
Master of Science in Computational Science
San Diego State University, 2005

Traditional vector field visualization methods are unable to demonstrate details in high resolution. In many cases these details are valuable to understand the underlying mechanism. Crack propagation in earthquake rupture dynamics is one of these examples.

Linear Integral Convolution (LIC) is one of the most popular texture methods for vector field visualization. The idea behind LIC is to demonstrate vector directions by blurring a texture along field lines. Several variations of this method (Regular LIC, Fast LIC, and Animated LIC) have been implemented in different computer languages (Delphi, Matlab and C). Evaluating the performance of these implementations, Fast LIC outperforms Regular LIC in large image and kernel lengths since its computation time is linearly dependent on the number of input pixels.

A Matlab toolbox is developed that helps geophysicists spot intricate details of a vector field. This toolbox, called VFV, uses Fast LIC method to produce texture and employs histogram determination methods to increase the contrast of output images. Matlab VFV toolbox provides a complete set of commands to generate LIC images and animations. The tools are arranged in a graphical user interface (GUI) for exploring vector fields with zooming capabilities. Demonstrated on several geophysics problems, these tools show the potential to provide new insights on complex mechanisms involved in crack propagation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I express my special thanks to Prof. Steven M. Day and Prof. Jose Castillo for their unparalleled support throughout my study in masters program.
Research resulting in this thesis has been supported by NSF EAR-0106924 (the SCEC Community Modeling Environment project) and NSF ATM.0325033.

CHAPTER 1

INTRODUCTION

Vector field visualization is a recent and fast growing subject. In this chapter basic concepts and methods in this area will be discussed.

A vector field is a map that assigns each point (x,y) a vector defined by a vector-valued function F(x,y). Figure 1.1 shows an example of a 2D vector field.

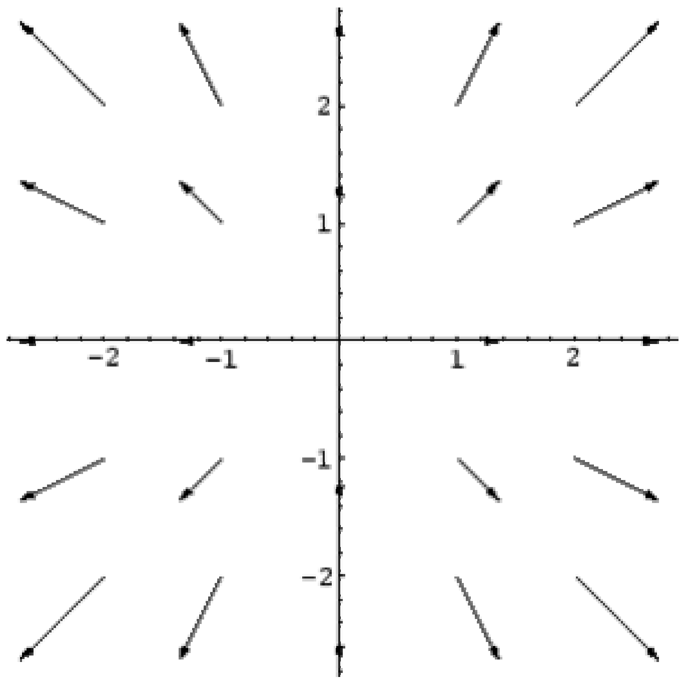


Figure 1.1 2D vector field.

Vectors fields appear in broad range of scientific applications. Magnetic and electric fields in electromagnetism (for an example of electric field see Figure 1.2 in which arrows are painted with lengths proportionate to field magnitude), heat flux vector field in continuum mechanics, vorticity in fluid dynamics or velocity and displacement fields in geophysics that contain valuable information about the underlying phenomena.

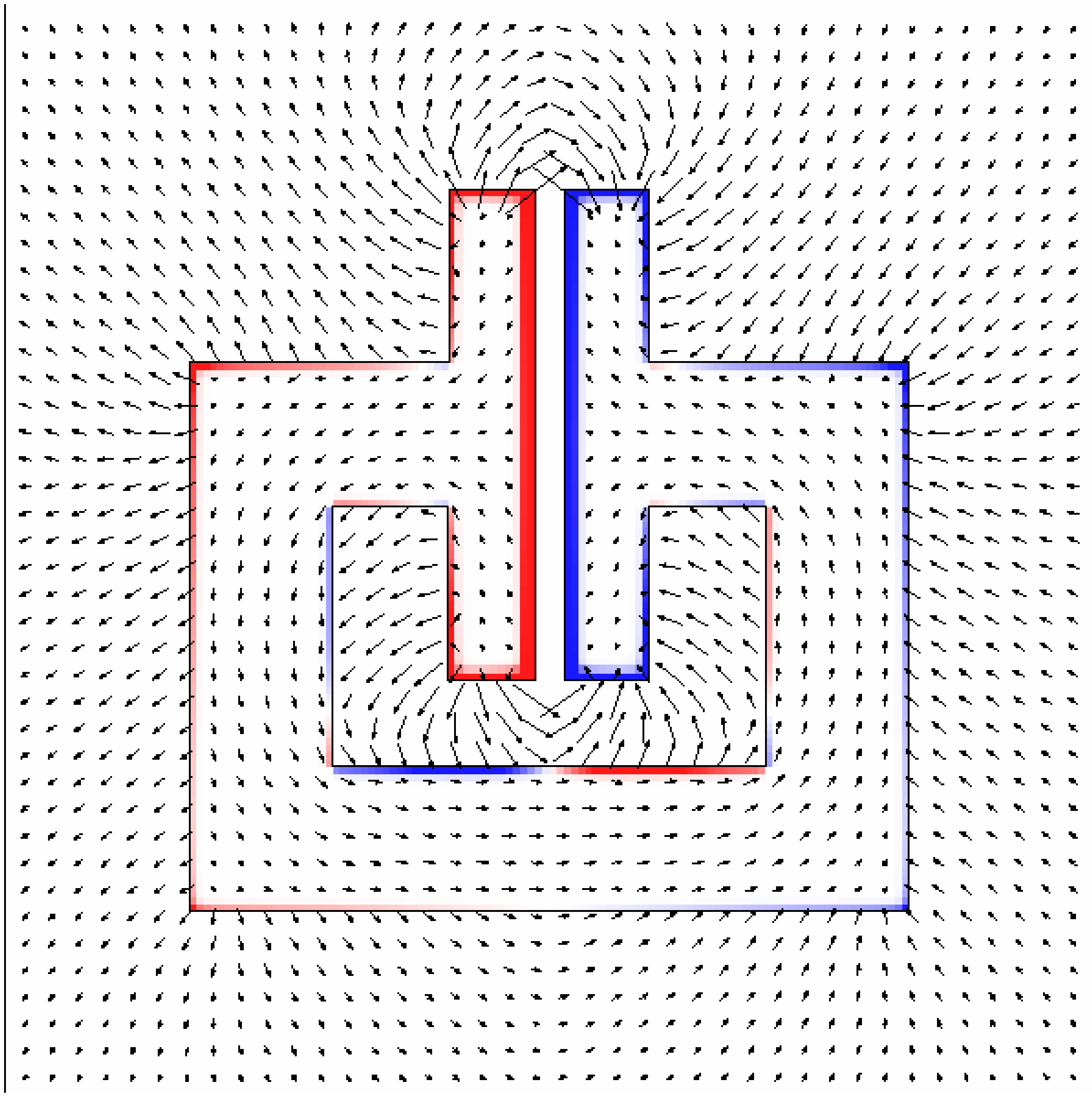


Figure 1.2 Example of vector fields in electrostatics: electric field in a resistor-capacitor system, source: galaxy.cofc.edu/cap11/cap11.html.

Presenting this rich information content in an intuitive manner is highly desirable for these and many other applications. Several visualization methods have been developed to address this need; these techniques can be categorized as:

1. Glyph or Icon representation (Klassen and Harrington, 1991): in this method small glyphs are placed at different parts of the field to convey overall structure. Figure 1.3 shows different types of glyphs suitable for demonstrating tangent and direction in two and three dimensions. This method works well for simple data, when putting sparse glyphs is adequate for observing field structure, but it has a poor resolution

since icons begin to ‘clutter’ at high densities. Figure 1.4 shows an example of this case.

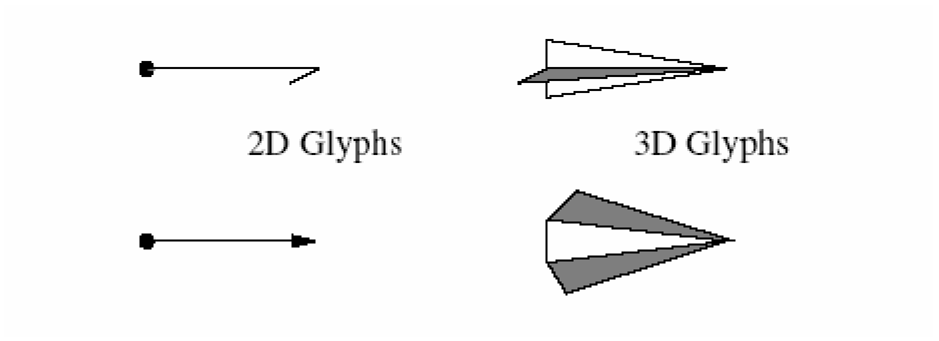


Figure 1.3 Different types of glyphs, Helgeland, 2002.

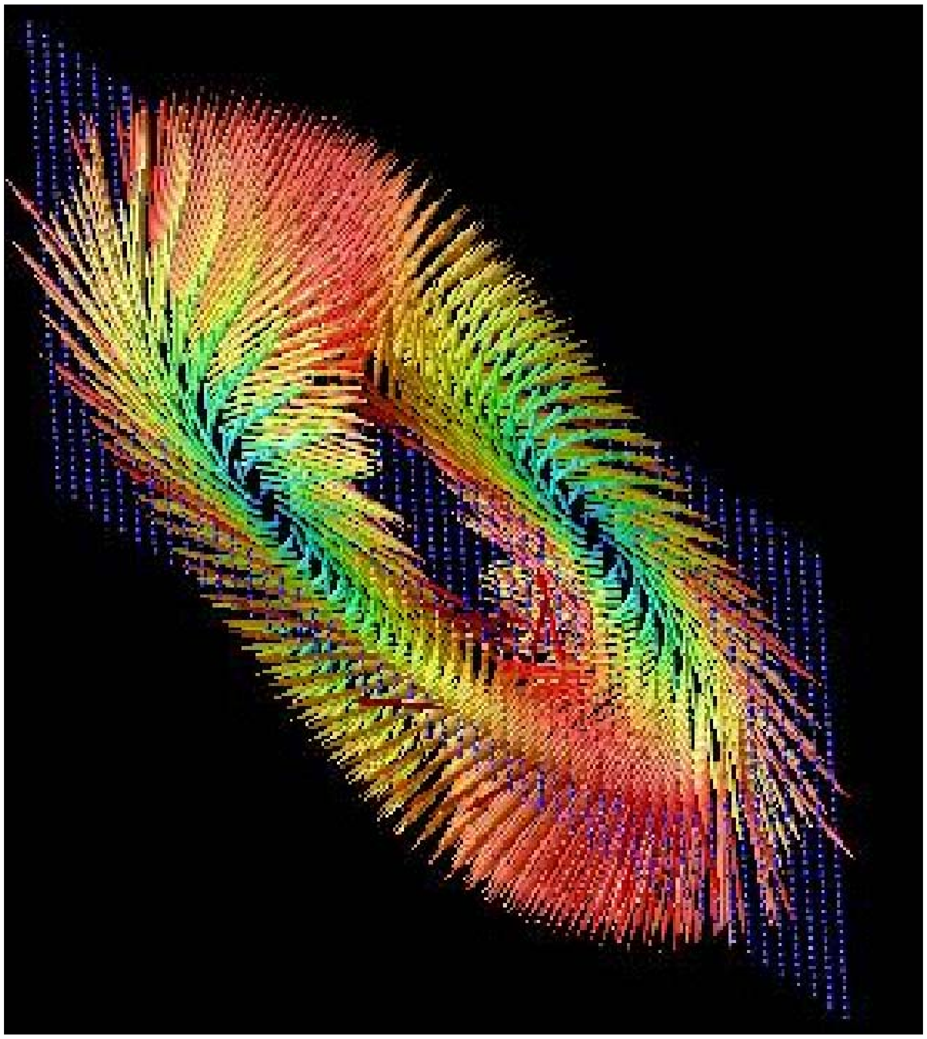


Figure 1.4 Example of glyph cluttering, source: http://www-beams.colorado.edu/dxhdf5/importhdf5field.html.

2. Stream Lines and Particle tracing (Kenwright and Mallinson, 1992): A stream line is the solution of $d\vec{x}/ds = \vec{E}/|\vec{E}|$, where x is the position vector, E the vector field being visualized and s is arc length along the stream-line curve. The Field can be visualized by a number of stream lines that are tangent to vectors at each point (Figure 1.5 shows glyph and stream line representations). On the other hand, path lines demonstrate the actual path of particles released from selected positions and move according to velocity and acceleration fields. Path lines are usually used in fluid dynamic visualizations. If the velocity field does not change in time, i.e. steady state flow, path lines will be the same as stream-lines. These techniques provide more insight on field structure but their usability is limited by several shortcomings. For example the final image is dependent on the choice of starting points. Also only limited number of stream lines can be plotted before image gets cluttered by them (Figure 1.6 shows 3D magnetic field stream-lines generated by two electric currents; notice how they overlap and clutter the picture). In spite of considerable advantages of these methods over Glyphs and their popularity, they offer limited resolution and fail to demonstrate fine structures.

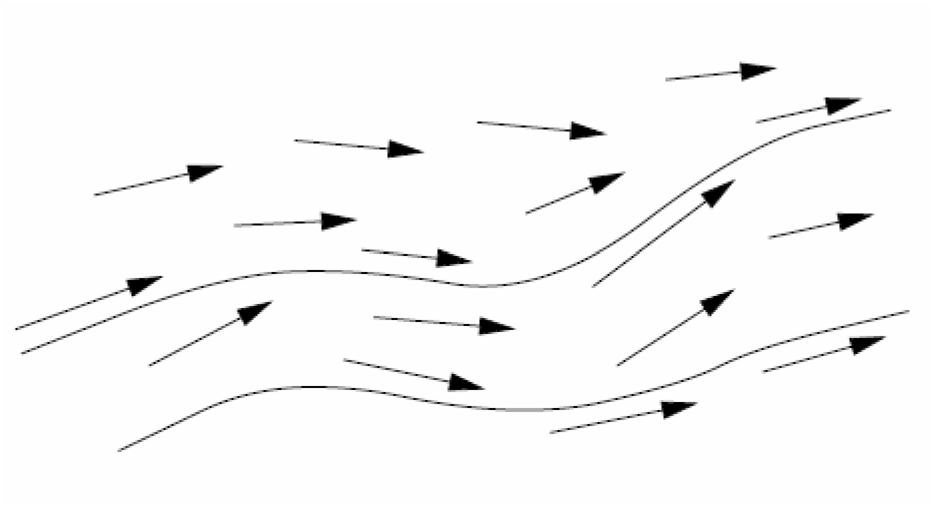


Figure 1.5 Stream-line representation, Helgeland, 2002.

3. Texture based methods (Laramee and Hauser, 2004): The need to achieve higher resolution in vector field visualization has inspired researchers to developed new techniques; these methods exploit every pixel in the output image to incorporate vector information and uniformly span the entire domain of the field. (In contrast with last two methods that only select a small sample from the field). Spot Noise algorithm (Wijk, 1991), Reaction Diffusion techniques (Turk, 1991; Witkin, 1991), and different types of Linear Integral Convolution (LIC) methods (Cabral and Leedom, 1993) (see Figure 1.7 which shows a colored LIC image generated by magnetic vector field of two electric charges) are among these techniques. As these methods work fine in case of information-rich fields, they may not be the best candidate if variations in the field are small and it has a simple structure (where glyph and stream-line methods generate less complex results).

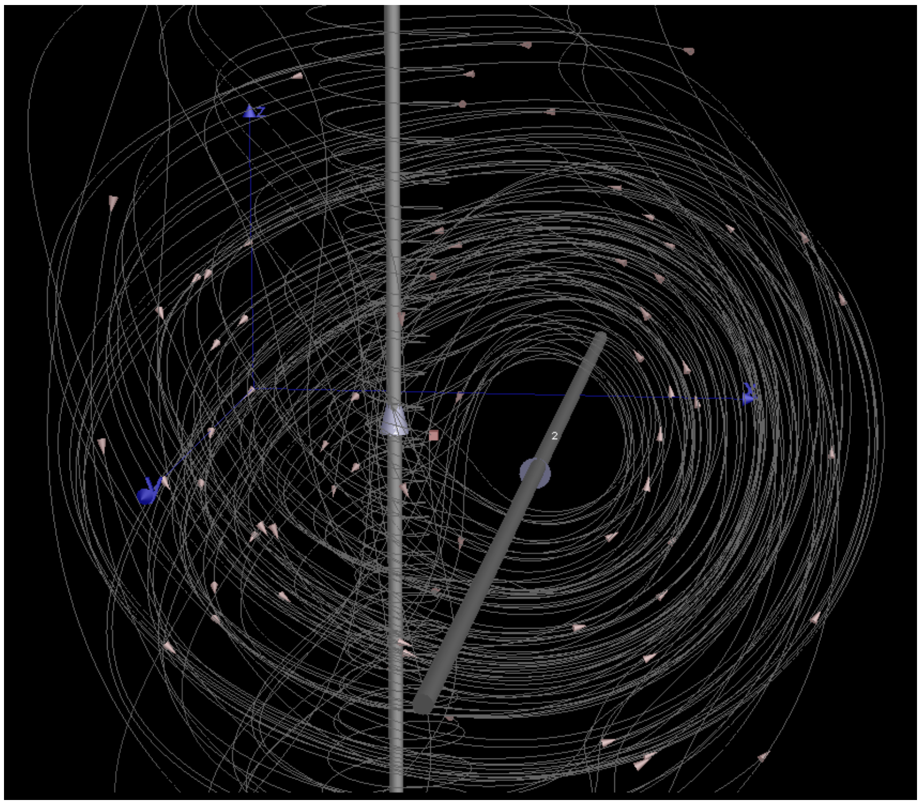


Figure 1.6 Example of stream-line cluttering: magnetic field lines generated from two electric currents.

In this thesis, a Matlab toolbox is presented that uses the LIC method for vector field visualization. In Chapter 2, availability of texture based vector field visualization methods for several common software packages is discussed. Then, in Chapter 3 different variations of LIC are explained. Details about implementing LIC in different computer languages are mentioned in Chapter 4. Applying Matlab toolbox to several crack propagation problems, its usability is evaluated in Chapter 5. Finally, Chapter 6 provides a summary and suggestions for future expansion of the toolbox.

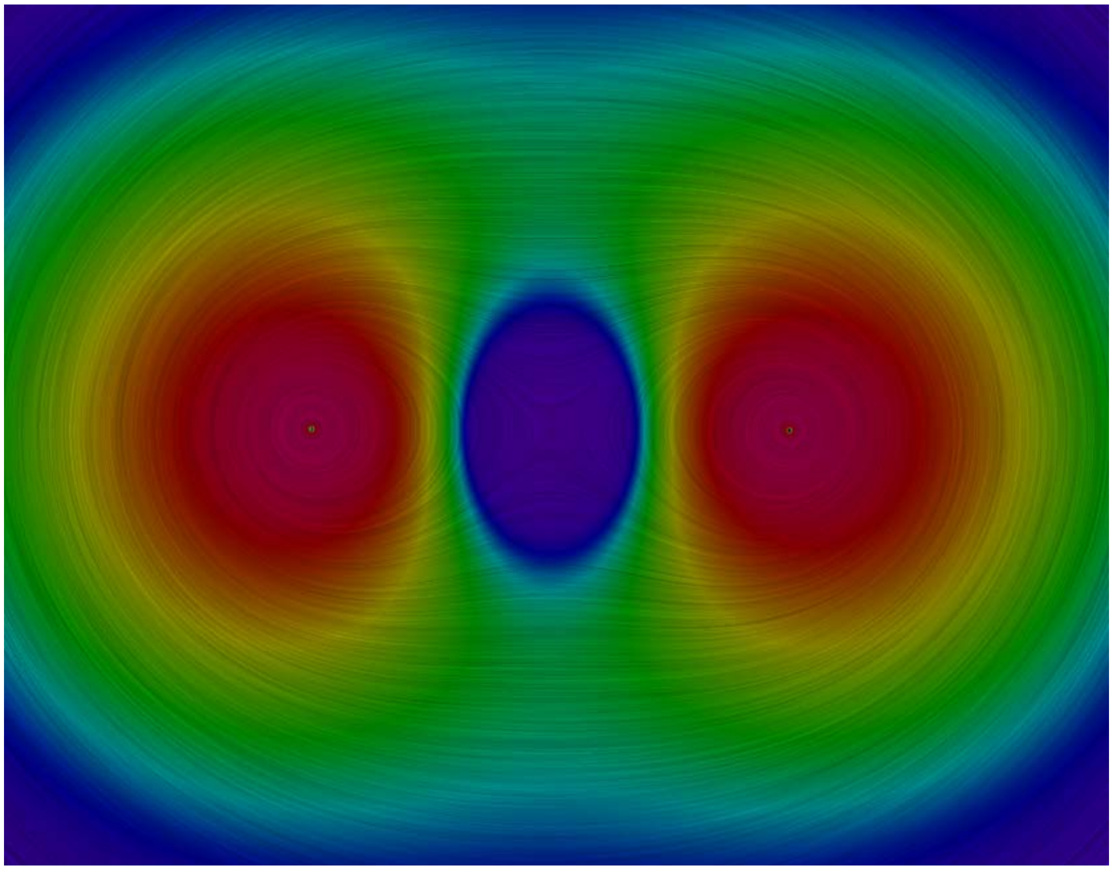


Figure 1.7 Using linear integral convolution on a magnetic vector field generated by two electric currents.

CHAPTER 2

MOTIVATION

The ability to effortlessly and precisely visualize scientific data plays an important rule in research conducted in variety of fields. For example computer simulations of earthquakes often produce vast volumes of data. Most of this data is composed of vector quantities which have to be presented in a visual manner in order to be thoroughly analyzed. Particle velocity, energy flux and particle displacement are among these quantities.

A typical simulation running on a supercomputer may use an 800 x 800 x 400 grid; this produces more than one gigabyte of data in each frame and vector fields account for most this data. Proper handling and visualization of these high resolution vector fields present a challenge for available software packages.

2.1 AVAILABLE PACKAGES

Table 2.1 shows availability of vector field visualization capabilities in some of common software packages:

Table 2.1 Vector Visualization Capabilities

Package	Glyphs	Stream Lines	Texture Based	Third Party Extensions
Excel 10	No	No	No	Unknown
OpenDX	Yes	Yes	No	Package for LIC method exists
Matlab 7.0	Yes	Yes	No	Package for LIC has been developed by author

Looking at Table 2.1, the lack of high resolution VFV (Vector Field Visualization) features is evident. This is an especially significant limitation in the case of Matlab, which is extensively used by researchers to visualize results of computer simulations of wave propagation, fluid flow, and other continuum physics problems in which high spatial resolution is necessary. A more detailed investigation in Matlab VFV capacities is presented the next section.

2.2 Matlab

Matlab is a popular tool among scientists. It is widely used for rapid prototyping and numerical analysis. Matlab version 7.0 offers several 2D and 3D VFV capabilities (see Figures 2.1 and 2.2 for some examples); but it lacks a method to study fine details of vector fields.

A Matlab Toolbox is developed by the author to utilize an optimized LIC method and image enhancing techniques (like histogram determination) tailored to the needs of scientists who work in geophysics and earthquake simulation field. The Matlab VFV toolbox relies on Matlab Image Processing Toolbox for contrast enhancement, color-map histogram equalization and other image processing operations. It uses external or 'MEX' functions to take advantage of C language speed in calculating LIC output.

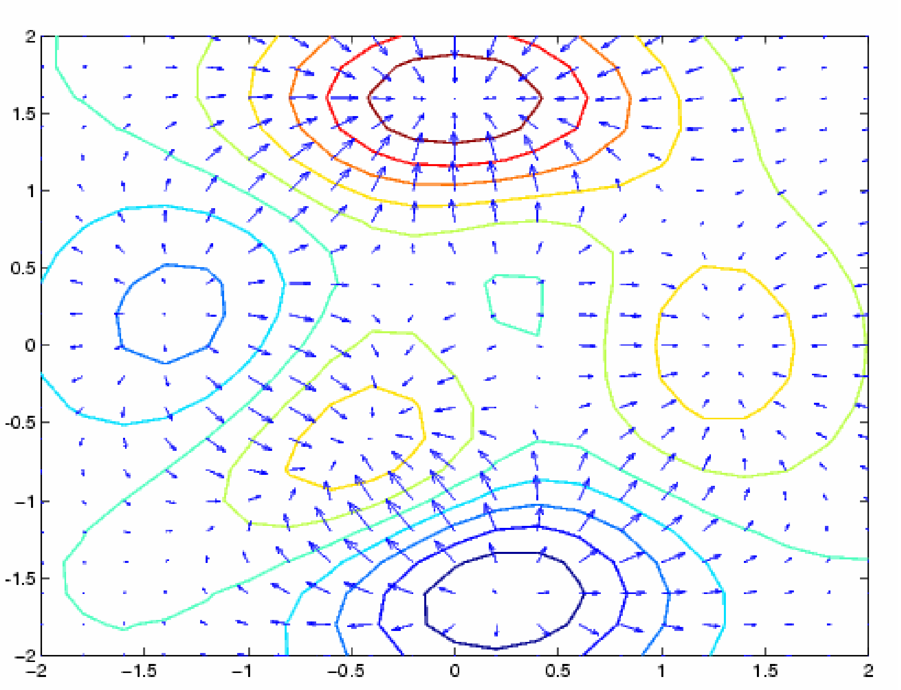


Figure 2.1 An example of matlab glyphs for 2D vector field visualization, from Matlab help.

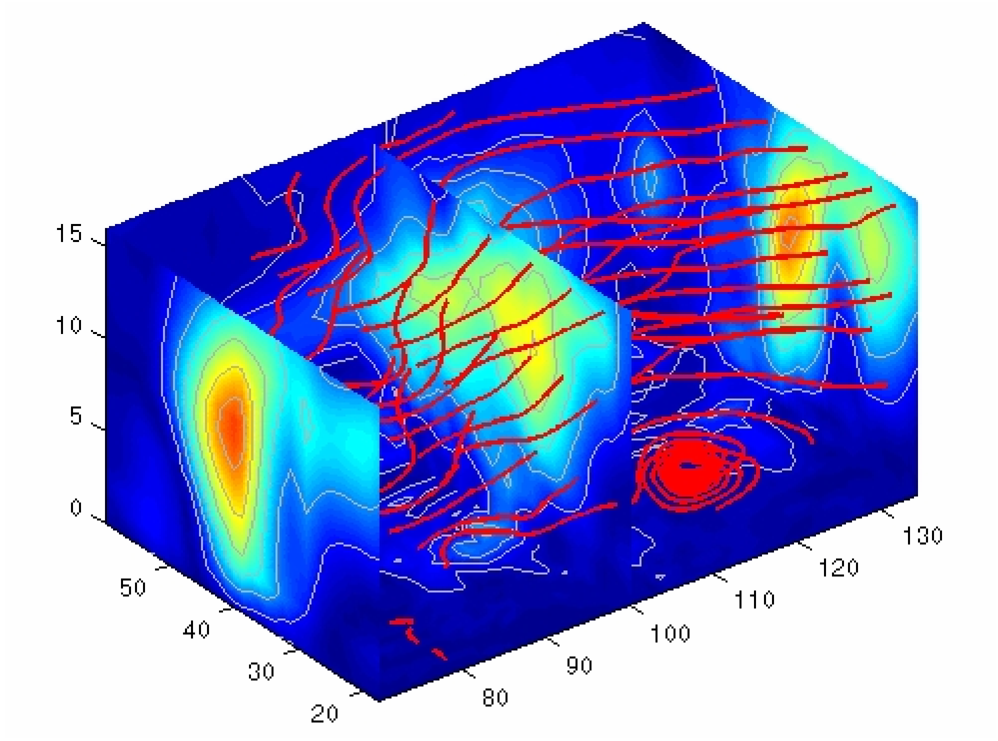


Figure 2.2 An example of matlab streamline VFV, from Matlab help.

CHAPTER 3

TEXTURE METHODS

Numerous texture based techniques have been developed; there is no general method that works best in all situations. Each technique has its own application and criteria of usability.

Sometime it is necessary to visualize vector fields that change in time, like velocity field in weather simulation or turbulent flow of a liquid; this is called unsteady-flow visualization. If the field does not change in time, steady-flow methods can be used. By this definition, some texture methods are categorized as steady-flow methods (DDA, Regular LIC, and Fast LIC) and some as unsteady-flow (UFLIC, AUFLIC). In this chapter we limit our discussion to steady-flow methods.

3.1 DDA CONVOLUTION

DDA stands for Digital Differential Analyzer. Traditional DDA (Bresenham, 1965) was developed to accelerate drawing line segments on the computer screen. DDA convolution is a generalization of traditional DDA line drawing techniques and Van Wijk (Wijk, 1991) and Perlin's (Perlin, 1985) spatial convolution algorithm. At each point of the input image a line with length 2L (DDA line) and tangent to the field at that point is calculated. This line extends in the positive and negative directions along the local vector field direction (with length L in each direction). A texture (usually white noise) is then mapped to the input image pixel by pixel; input texture values on the tangent line are multiplied by a filter kernel, summed and normalized to form a single value. This value is then placed in the output pixel image as the intensity value for the vector position (Figure 3.1 demonstrates the process). The total effect is to blur the input texture as a function of vector field (See Figure 3.2 for an example of final image).

The most common filter kernel is box kernel (calculating average along kernel length). Although any function can be used for this purpose, the DDA algorithm is very sensitive to symmetry in the kernel and the integral path. For example, if the algorithm weights the forward direction more than the backward direction, the circular field in Figure 3.2 appears to spiral inward implying a vortical behavior that is not present in the vector field.

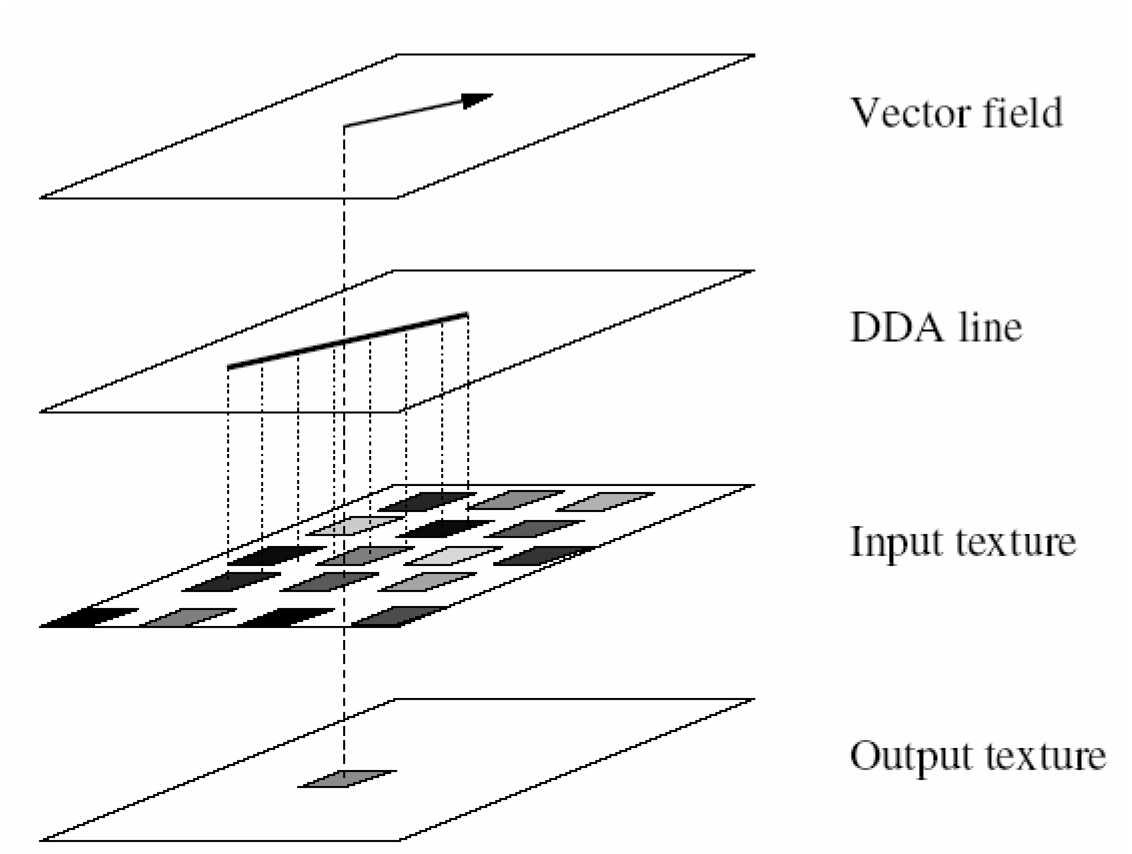


Figure 3.1 Generating an output pixel using a DDA line: (from top to the bottom) First a line segment in the direction of vector field is calculated at each point, then pixels from input texture that lie on the line are found and their values are convoluted with kernel filter. Finally, the result of the convolution is put into the output image. From Cabral and Leedom, 1993, copied by permission of the Association for Computing Machinery

The DDA approach is basically a first order approximation of the field, in the sense that it assumes field path-lines can be replaced by straight lines. Problems arise when complex structures with dimensions comparable to DDA lines (small eddies or vortices) exist in the vector field resulting in large curvatures that could not be neglected. DDA may produce invalid images in this case, since its first order approximation to the streamline trajectories is unable to demonstrate curvatures with small radius. Paying close attention to

Figure 3.2, which shows DDA output for circular and turbulent vector field, you can see ambiguity in high curvature areas.

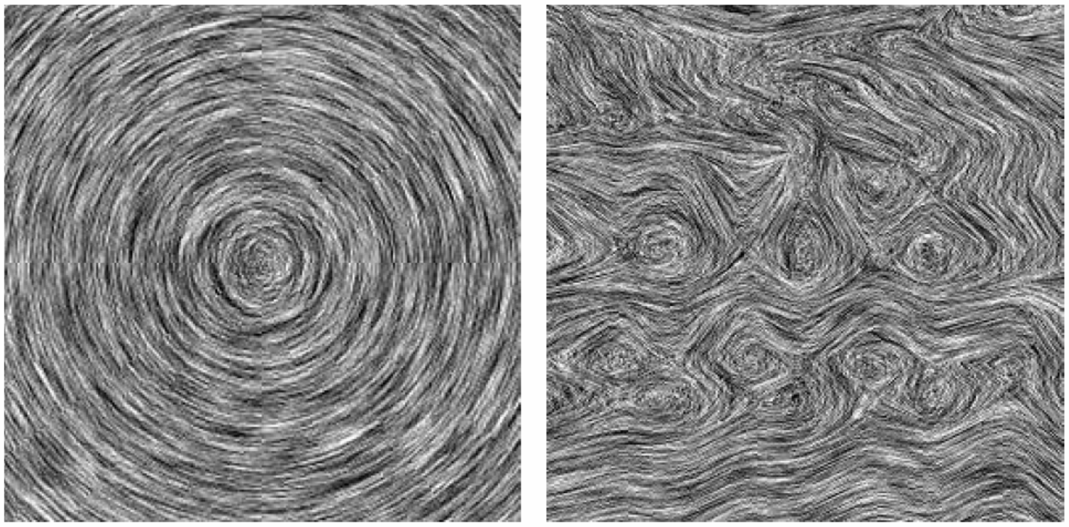


Figure 3.2 Circular and turbulent fluid dynamics vector fields imaged using DDA convolution over white noise, from Cabral and Leedom, 1993.

3.2 Linear Integral Convolution

As we discussed, the first order approximation of DDA may hide important details in the vector field. Linear Integral Convolution, or LIC, is a logical improvement of DDA. It computes accurate stream-lines (instead of DDA lines) and uses them in the convolution. A comparison between right image in Figures 3.2 and 3.3 shows a noticeable increase in visualization quality which is resulted from replacing DDA lines with LIC stream-lines. Vortices are more noticeable in Figure 3.3 with well defined curves around them, but in Figure 3.2 they are hidden because of the linear approximation involved in DDA.

Like DDA, LIC calculates a local stream-line at each point extending in both forward and backward directions from the point, as illustrated in Figure 3.4.

General steps for making an LIC image are the following:

1. Generating the input texture: Since LIC has a blurring effect and reduces high frequencies in output image, using a high frequency input texture is desirable. In most cases a white noise texture is used.

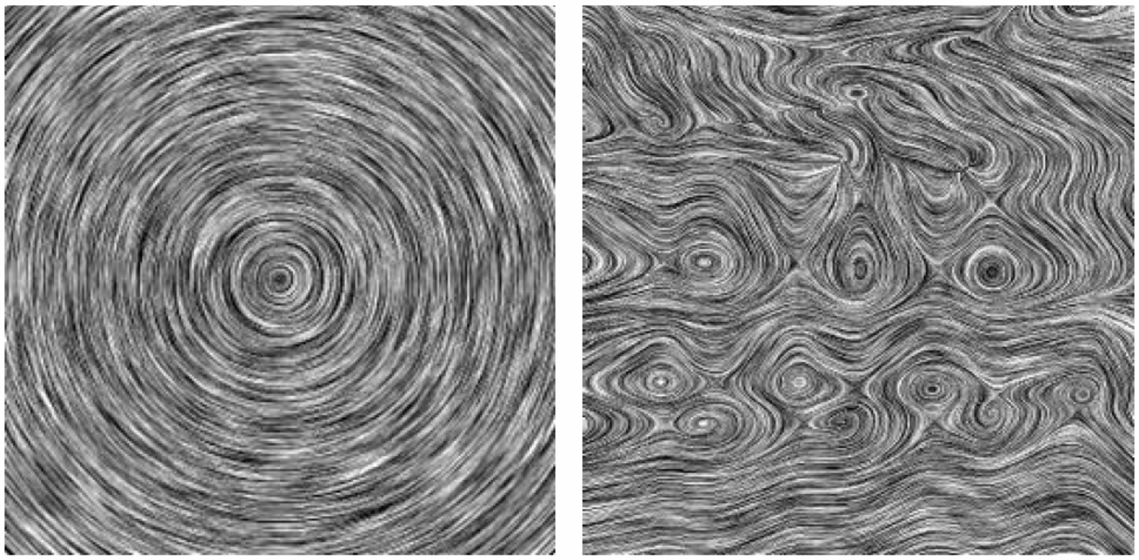


Figure 3.3 Circular and turbulent fluid dynamics vector fields imaged using LIC convolution over white noise (same data as Figure 3.2, from Cabral and Leedom, 1993.

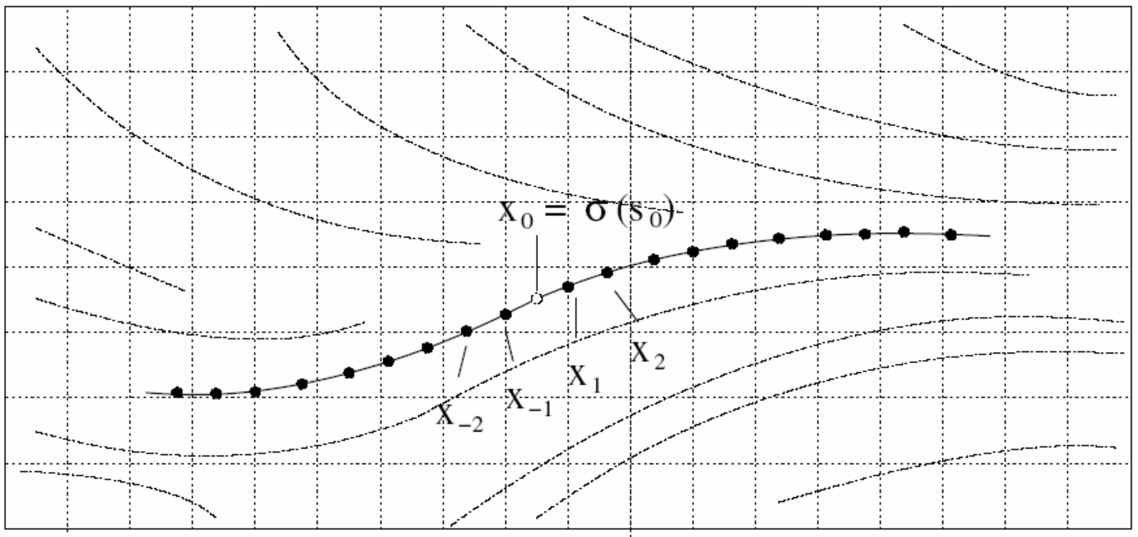


Figure 3.4 Two-dimensional vector field showing the local stream-line starting in point x0 and going in both directions, Helgeland, 2002.

2. Calculating stream-lines of the field in both directions for each point: in most cases an Euler integrator is chosen to increase the speed. A more accurate method like Runge Kutta can also be used but it may slow down the algorithm. To avoid unwanted effects near singularities, the vector field is usually normalized (all vectors normalized to unit length).
3. Calculate output image intensity values with this formula:

$$I(x_0) = \int_{s_0-L}^{s_0+L} k(s-s_0)T(\sigma(s))ds \qquad (3.1)$$

Where

Curve σ(s) is a stream-line parameterized by arc-length s.

I(x0) is the intensity of a pixel located at x0 = σ (s0).

L is convolution length in each direction. It determines how much the texture is smeared in the direction of vector field.

k denotes the filter kernel of length *2L*.

T is texture (for example, white noise) value at point σ(s).

The mathematical meaning of the function I is that it is a convolution of input texture with the kernel function, along each stream-line. The effect of this convolution is illustrated in Figure 3.5.

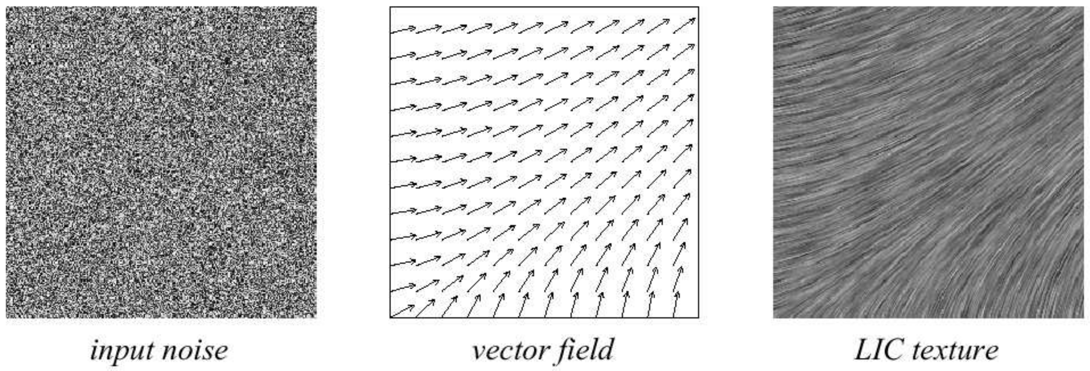


Figure 3.5 Linear integral convolution effect on white noise input texture, Helgeland, 2002.

4. Post-processing: up to this step, each pixel of output image is assigned with a value for intensity. Since the LIC process can be seen as a special kind of averaging operation, output intensity values are very close to each other (the longer the integral length 2L is, the smaller is their difference) and details are not recognizable (low contrast).

   To solve this problem, the contrast of output image needs to be enhanced. That is, it is necessary to stretch the range of output image pixel intensity values in such a way that they cover the full available intensity range (from zero to maximum intensity). After this enhancement, the output image is a grayscale image, with a texture representing the direction of the field at each point. In many applications, displaying the magnitude of the field is also desirable, and coloring techniques can be used to incorporate this information in the output image. The latter technique is discussed in section 3.5.

3.3 Fast LIC

Regular LIC is a computationally intensive operation; for every pixel in output image a stream-line has to be calculated, then a convolution operation has to be applied between values on the stream-line and kernel. This means the amount of computation for each output image is at least proportional to the number of pixels multiplied by the kernel length. For an 850 x 850 input image with kernel length of 85, a typical PC (Intel 2.53 GHz) takes about 23 seconds to calculate a Regular LIC image. This speed is not enough to make animations and as we will see in section 4.3, the computation time increases faster than linear with number of input pixels.

Many pixels lie on each stream-line; (Stalling and Hege, 1995) proposed LIC calculation can be accelerated by changing the order of operations. The resulting procedure is called Fast LIC. Instead of scanning pixels in an arbitrary fashion and calculating stream-lines for each of them, Fast LIC takes advantage of pixels already computed on each stream-line and convolves the kernel with the section of the stream-line adjacent to them. In this manner, long stream-lines are computed spanning several times the kernel length, eliminating the need to recalculate shared stream-line segments for adjacent pixels. This method reduces total computation time by making it linearly proportional to the number of pixels in input image while computation time for Regular LIC is proportional to a power of input pixels number that is more than one (See Figure 4.8 for a quantities comparison).

Additional performance improvements can be achieved for certain kernels using iterative integral stencil. For example, in case of ‘Box’ kernel (which performs an averaging operation over a section of length 2L around each pixel along stream-line), the calculated value of the adjacent pixel on the stream-line can be used to evaluate intensity of each pixel: Imagine there are two points on the same stream-line x1 = σ (s1) and x2 = σ (s2) close to each other with small distance Δs = s2 – s1 for a box filter kernel *k* the convolution integral (Equation 3.1) for x2 can be written as

$$I(x_2) = I(x_1) - k\int_{s_1-L}^{s_1-L+\Delta s} T(\sigma(s))ds + k\int_{s_1+L}^{s_1+L+\Delta s} T(\sigma(s))ds \quad (3.2)$$

The convolution integral for point x0 can be written as

$$I(x_0) = k\sum_{i=-n}^{n} T(x_i) \quad (3.3)$$

Inserting equation 3.3 into 3.2 we can calculate the integral for other pixels on the field line by moving in both directions:

$$I(x_{m+1}) = I(x_m) + k[T(x_{m+1+n}) - T(x_{m-n})], m = 0,1,..., M$$
$$I(x_{m-1}) = I(x_m) + k[T(x_{m-1-n}) - T(x_{m+n})], m = 0,-1,...,-M$$

This formula dramatically increases the performance by using an iterative method instead of full integration. It is equivalent to the conversion of a linear filter from convolution form to an equivalent recursive form. To calculate the convolution for N points, ignoring pixels close to borders, we just need 2N summations instead of 2 x (kernel length) x N summations needed for non-recursive algorithm.

3.4 ANIMATED LIC

LIC-generated images have ambiguous field directions: LIC efficiently incorporates the field vector tangents in the output picture, but since the kernel convolution is symmetric, directional information is lost. (Freeman et al., 1991) describe a technique to suggest the directional information by simulating motion along the streamlines by use of special convolutions. This method involves animation of successive LIC images using varying phase shifted filter kernels. Instead of using a constant or 'box' filter, a periodic kernel is applied; then by changing the phase of the filter, apparent motion in the direction of the field vector is created.

A natural first candidate for a periodic filter kernel is sine function. It produces a distinguishable sense of motion along the direction of field lines. After I made several experimental animations using sine kernel, they seem to include a considerable amount of high frequency noise, decreasing temporal coherency. It is possible to create a better periodic filter to blur the texture in the direction of the vector field. A Hanning filter, $\frac{1 + Cos(w + b)}{2}$, can achieve this goal: it is periodic, has a simple analytical form and produces more coherent animations (Cabral and Leedom, 1993). Another name for this function is *ripple* filter function (Figure 3.6 shows ripple function in different phases).

LIC algorithm operates on a limited length so any kernel function has to be windowed; that is, it has to be made local, even though it is originally defined over an infinite length. For a constant filter this is done by making a 'box' and cutting off the rest of the function; applying the same operation on the Hanning function results in abrupt cut-offs (Figure 3.6).

These cut-offs are noticeable as spatio-temporal artifacts in animation that vary phase in time. Gabor (Chui, 1992) suggested using a Gaussian function to solve this problem: Multiplying (windowing) the Ripple function by a Gaussian function, cut-offs diminish at the boundaries. Unfortunately a Gaussian windowed Hanning function does not have a simple closed form integral. Looking for a window function with windowing properties similar to Gaussian and having a simple form integral, Cabral (Cabral and Leedom, 1993) stresses that the Hanning function itself meets these criteria (Figure 3.7 shows a Hanning windowed ripple function in different phases). Multiplying the Hanning filter functions in Figure 3.6 by Hanning window function from Figure 3.7 top results in five phase shifted Hanning functions shown on Figure 3.7 bottom. These functions smoothly go to zero on window boundaries; hence they do not create artifacts in the animation.

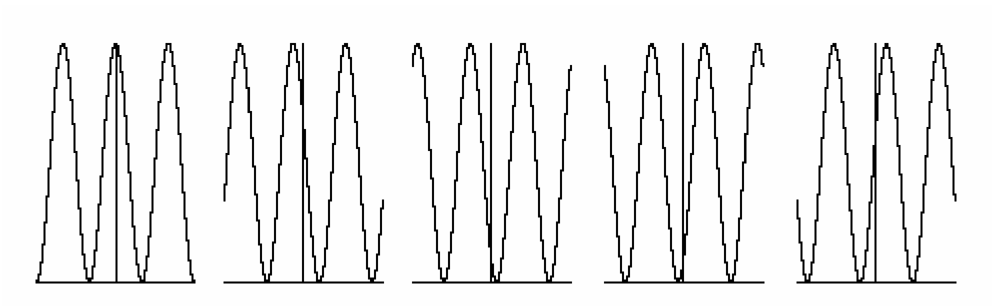


Figure 3.6 Phase shifted Hanning (ripple) function, from Cabral and Leedom, 1993.

3.5 ADDING COLOR

LIC produces an intensity output image, which may be regarded as a gray-scale picture; this image usually only includes tangential information of vector field and is stripped of information on vector magnitude. By post-processing the LIC intensity image, we can return this information back into the output image.

A color-map is a function from real numbers to colors-space. It can be used to depict the magnitude of the vector at each point. A colored magnitude image can be made by first calculating the length of vectors in the field at every point, then applying the color-map to the calculated vector magnitudes.

LIC conveys direction by modifying image intensity and for human eye a natural way to understand intensity is by shades of gray. Color can be projected on three axis of Hue,

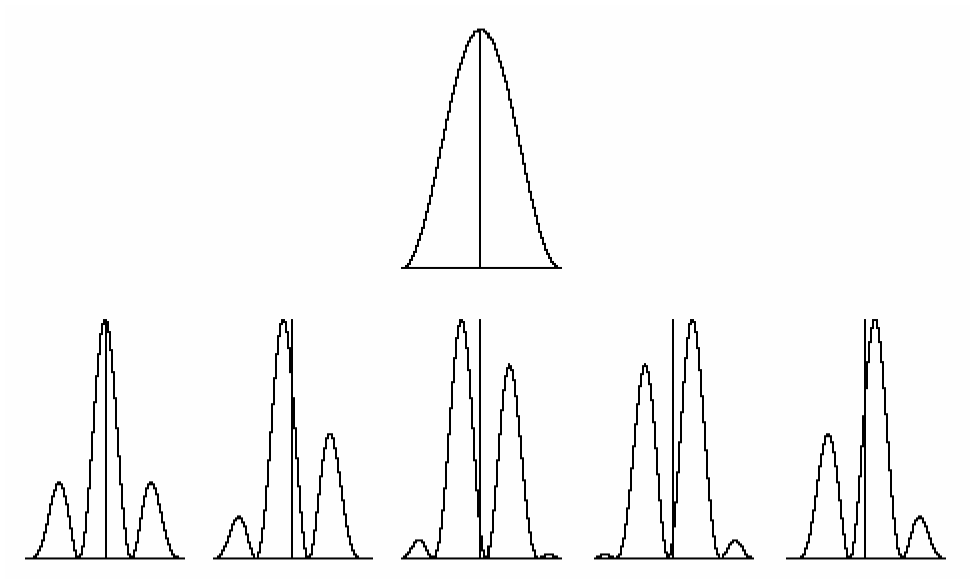


Figure 3.7 (top) Hanning window function (bottom) Hanning (ripple) function multiplied by Hanning window function, from Cabral and Leedom, 1993.

Saturation and Value (Luminescence) or HSV (Figure 3.8 shows HSV color space). One way to mix colored magnitude image and LIC gray-scale image is to put them on different color axis.

It is desirable to use a color-map that maps magnitude to different Hue values (i.e. minimum to blue, maximum to red) and put LIC intensity image on Value axis of the final image. This way since Value changes luminescence of the final image, LIC texture remains distinguishable. Figure 3.9 illustrates this process: The LIC intensity image is placed at the top; intensity is shown by the gray-scale in the picture. The magnitude image is placed in the middle of figure, where lower magnitudes are colored by shades of blue and higher magnitudes by shades of yellow and red. To compose the final image, the LIC intensity values are put on the Value channel of the final image and the Hue values are copied from the Hue channel of the magnitude image to the Hue channel of the final image. In the final step, Saturation for all pixels of the final image is set to 1 to produce the brightest colors. The final image shown at the bottom of Figure 3.9 depicts tangent and magnitude of vectors but it

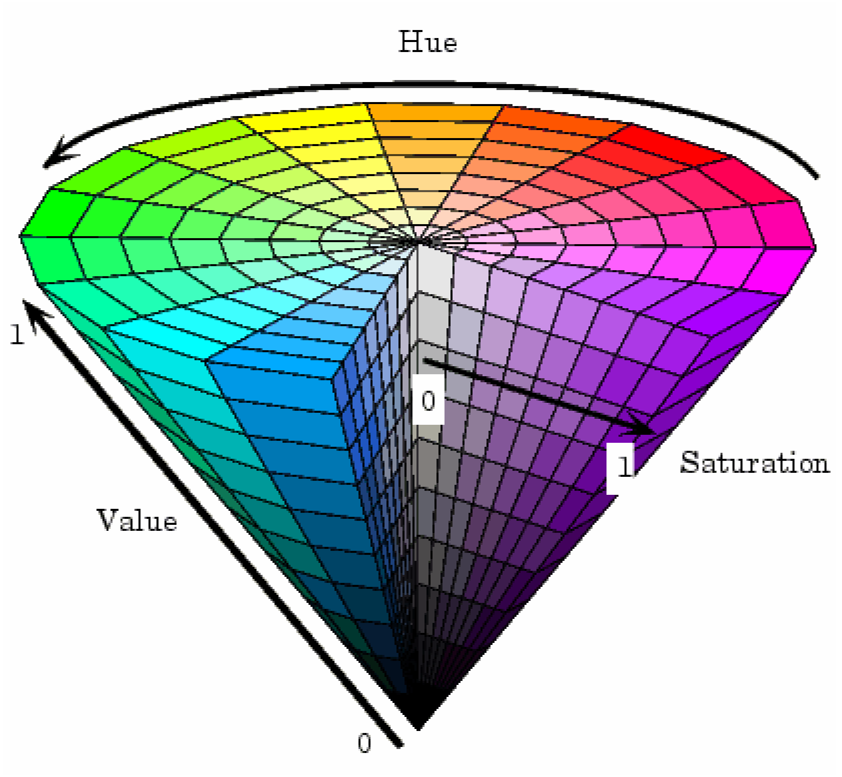


Figure 3.8 HSV axes of color, from Matlab help.

still lacks directional information. A simple way to add direction is to draw small arrows on the picture. Figure 3.10 show the vector field from Figure 3.9 bottom with added arrows. Since human vision tends to search for regular patterns, care should be taken to not put arrows in a rectangular grid; such positioning would impose an artificial rectangular structure on the picture and divert attention from other possible patterns. Instead, I have found, on the basis of experiments done for this study, that better perception of the underlying patterns in the data is possible when arrows are positioned at semi-random coordinates (no two of them are closer than a fixed distance) . To produce such a pattern, I have found the following simple procedure to work effectively: one can pick random points and accept them only if their distance to their closest accepted point is more than a certain length.

3.6 Iterative LIC

Using a box kernel is the equivalent of calculating the average (over a streamline segment) of the on white noise input image. Therefore, calculated output intensity approaches a constant value as kernel length increases. Since variance of a measured quantity

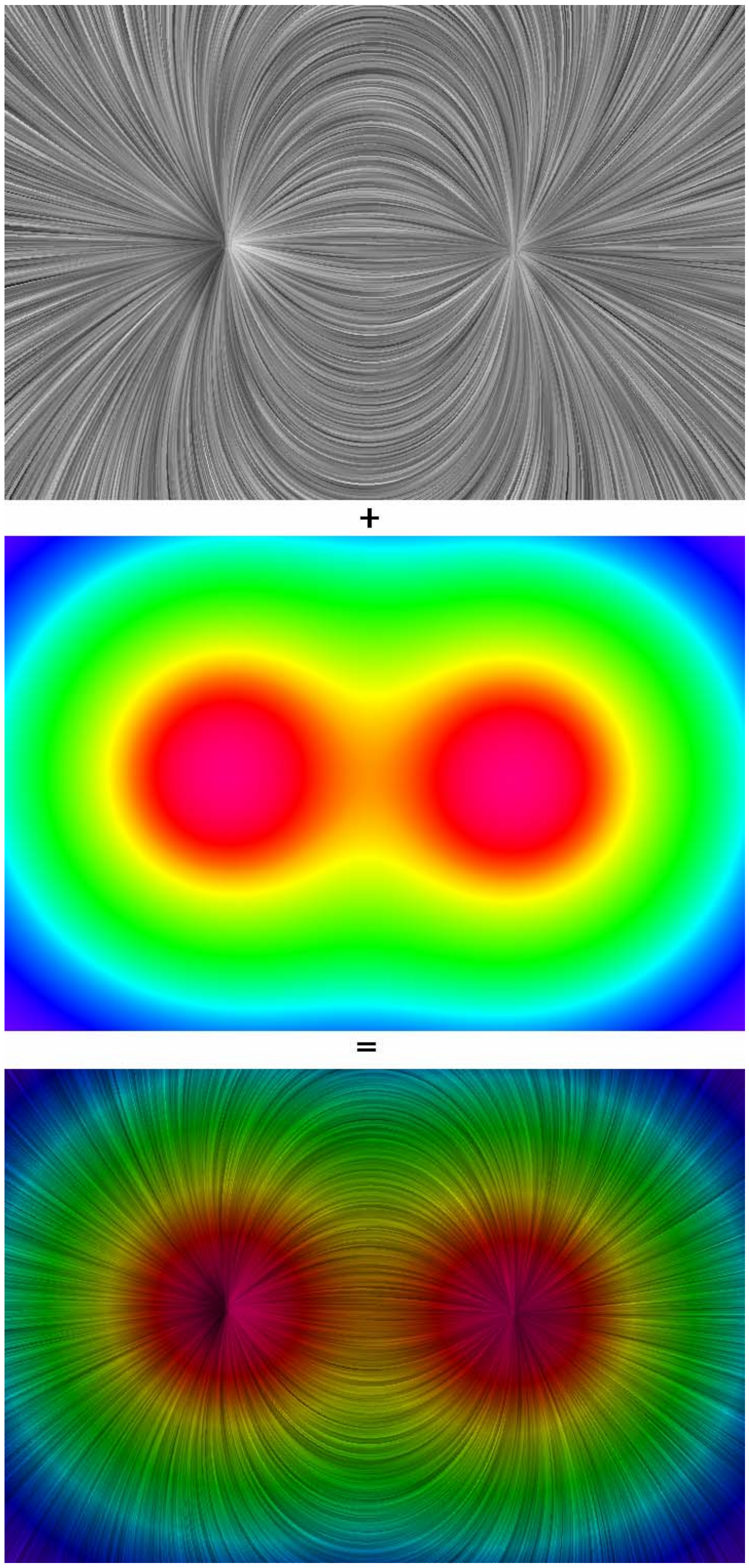


Figure 3.9 (Top) LIC intensity image (middle) magnitude image (bottom) final post-processed LIC image.

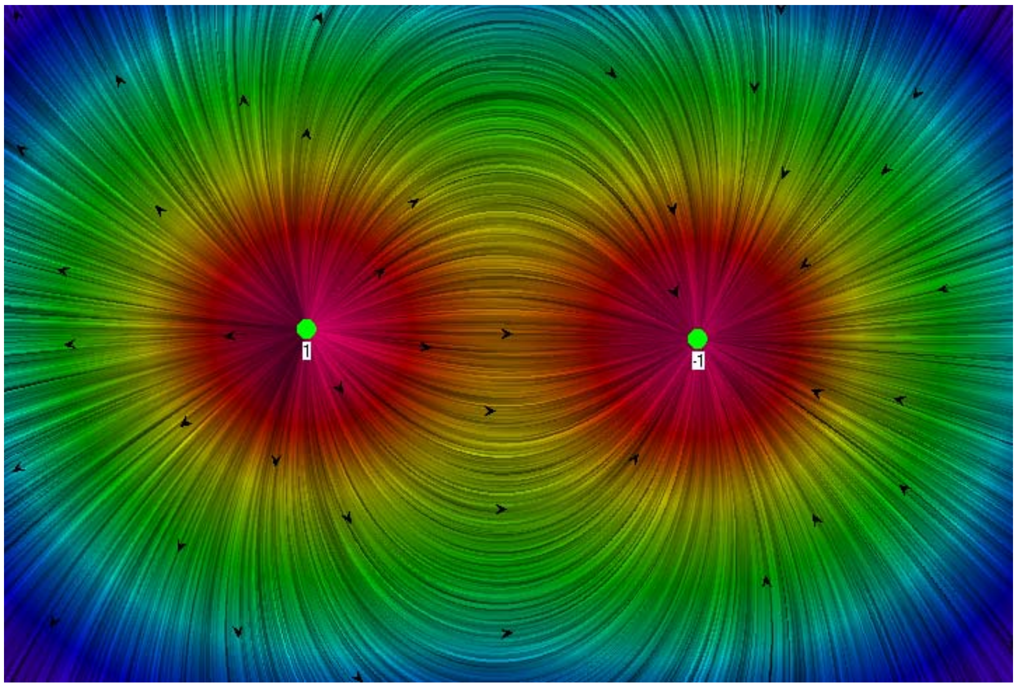


Figure 3.10 Direction is shown by small arrows at random positions.

decreases by factor of $\frac{1}{\sqrt{N}}$ (N is the number of measurements), output image contrast is reduced by factor of $\frac{1}{\sqrt{KernelLength}}$ .It is desirable to choose a relatively large kernel length (about 1/10 of largest image dimension) to improve image coherence. Unfortunately, long kernel length also reduces output image contrast; in some cases it may not be feasible to achieve a reasonable contrast even using contrast enhancement methods (if variations become small enough to be comparable to defined accuracy of floating point data type, i.e. round off error).

According to equation 3.1, for output image I_1 after applying LIC on texture T we have:

$$I_1(x_1) = \int_{s_1-L_1}^{s_1+L_1} k(s-s_1)T(\sigma(s))ds \qquad (3.4)$$

Now, if we apply another LIC run but this time using the output image I_1 from the last run as input texture:

$$I_2(x_2) = \int_{s_2-L_2}^{s_2+L_2} k(s'-s_2)I_1(\sigma(s'))ds' \qquad (3.5)$$

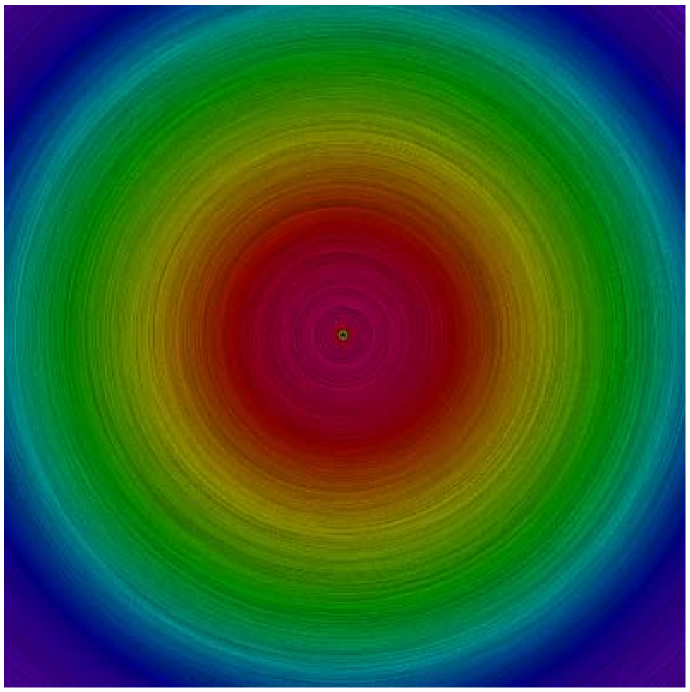
Inserting 3.4 into 3.5 we get

$$I_2(x_2) = \int_{s_2-L_2}^{s_2+L_2} k(s'-s_2) \int_{s'-L_1}^{s'+L_1} k(s-s')T(\sigma(s))dsds'$$

Changing the order of integration and combining the limits, this can be simplified to

$$I_2(x_2) = \int_{s_2-L_1-L_2}^{s_2+L_1+L_2} k(s'-s_2)T(\sigma(s'))ds' \qquad (3.6)$$

Comparing equation 3.6 to equation 3.1 we can see equation 3.6 is equivalent to applying one LIC run with combined kernel lengths. Iterative LIC maintains image contrast by breaking LIC convolution into several rounds, enhancing contrast at end of each LIC run then using the output image on each run as the input image for the next LIC round. Overall effect is similar to having a large kernel length (high coherence) but sustaining a reasonable contrast. Mathematically, it does not make a difference whether one enhances contrast in the final stage or between rounds, but since numbers have finite accuracy in computer operations, enhancing the contrast between rounds prevents irreversible loss of contrast after long kernel lengths. Figure 3.11 top shows an LIC image generated from magnetic vector field with kernel length of 50 pixels after contrast enhancement; Figure 3.11 bottom shows the same vector field after two LIC iterations each with length 25 pixels and contrast enhancement after each run. Comparing these two images, a noticeable improvement in contrast and spatial coherence can be seen in Iterative LIC image.



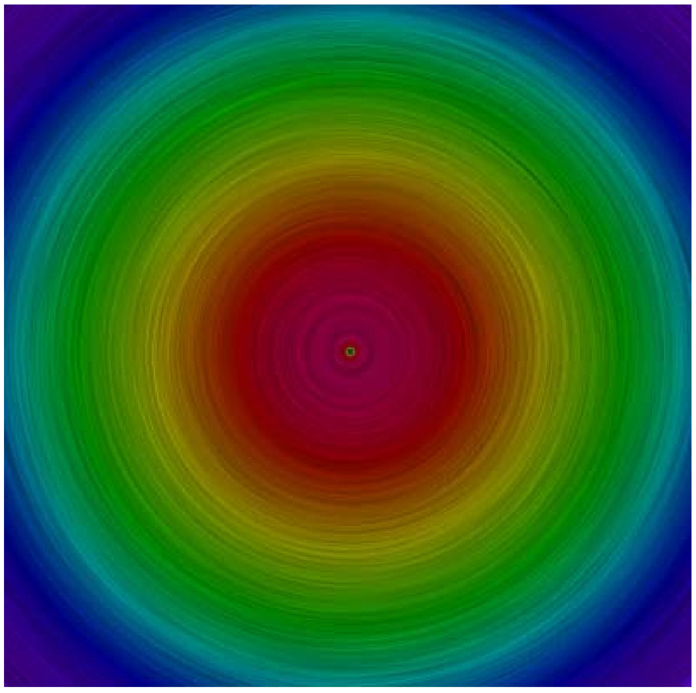


Figure 3.11 (top) LIC image after one iteration of length 50 pixels and contrast enhancement (bottom) LIC image after two iterations of length 25 pixels and contrast enhancement after each iteration.

CHAPTER 4

IMPLEMENTATION

In order to visualize vector field data produced by crack propagation simulations, I implemented LIC technique in several computer languages; each of these implementations has unique features which make it suitable for a particular task.

4.1 DELPHI

The first implementation of LIC was in Delphi (Pascal language) using Regular LIC method. The reasons for this choice were personal proficiency in the language, speed of execution and readability of the Pascal language. Also since Delphi provides numerous tools for rapid application development, building a graphical user interface (GUI) was an easy task. The code is robust; calculating a 720 x 320 output image with a 10 pixel length box kernel takes less than 3 seconds on a home PC (2.52 GHz Pentium 4).

Several animations are made using this code, demonstrating velocity, ground displacement and energy flux calculated from crack propagation codes. Figure 4.1 shows a frame from one on these animations; it visualizes velocity field for bimaterial crack propagation (Harris and Day, 1997). It is important to notice that these animations are different from Animated LIC discussed in section 3.4. Here each LIC frame corresponds to a different time-frame of simulation code, but in Animated LIC directional information of a single time-frame is shown by apparent motion in animation.

After using Delphi code to animate earthquake simulation data, the need to develop a more elaborate method for color-map construction became evident. Each color-map is composed of an internal variable that spans a limited range in which colors are defined (for example this variable can go from 0 for blue to 1 for red, see Figure 4.2) and a mapping from data to the range of internal value. To assign a color to each input number, the number is first mapped to the internal variable range using a mapping function. The simplest color-map uses a linear function for mapping spanning the range from minimum to maximum of the input variable. Figure 4.3 shows a linear mapping function from input values to internal color-map variable.

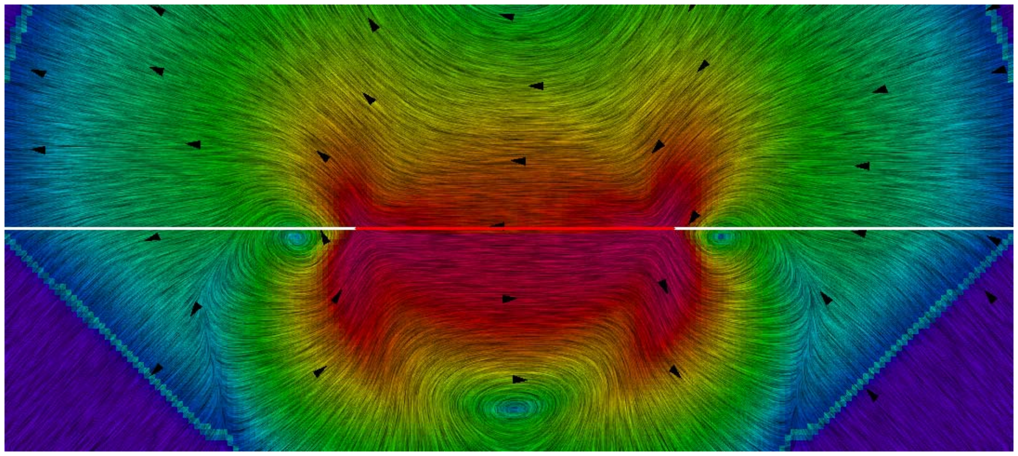


Figure 4.1 Delphi code visualizes velocity field produced by a crack propagating simulation.

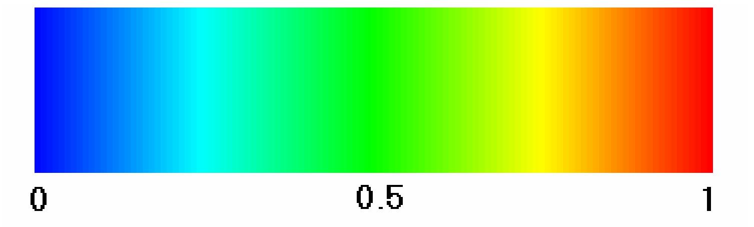


Figure 4.2 Color map internal variable.

Linear mapping does not guarantee the best results. Consider the following example: let most of the input values lie on a small section, say in middle of maximum and minimum. In this case, only a few pixels contain values near the minimum or maximum (see Figure 4.4 for an example). Also assume there are 100 distinct colors assigned to values of internal color-map variable. A linear mapping function allocates these colors uniformly to input value range. Since there are more pixels with values in the middle than pixels with values near the boundaries, linear mapping results in lower color contrast for pixels in the middle (and, unfortunately, most of the pixels lie in that range).

A better way is to maximize color contrast using histogram equalization: instead of evenly distributing color-map colors between input values, this method assigns them evenly between input pixels in such a way that there are approximately same number of pixels with each color in the output image.

Histogram equalization is performed by first calculating the histogram of input values, then using the normalized cumulative histogram (ratio of the number of pixels with

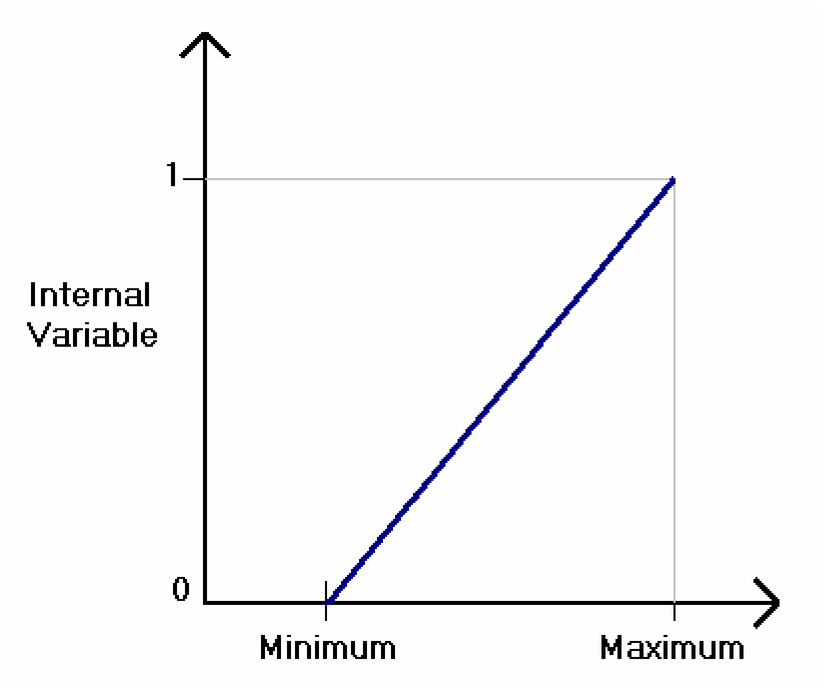


Figure 4.3 Linear mapping from input values to internal color-map variable.

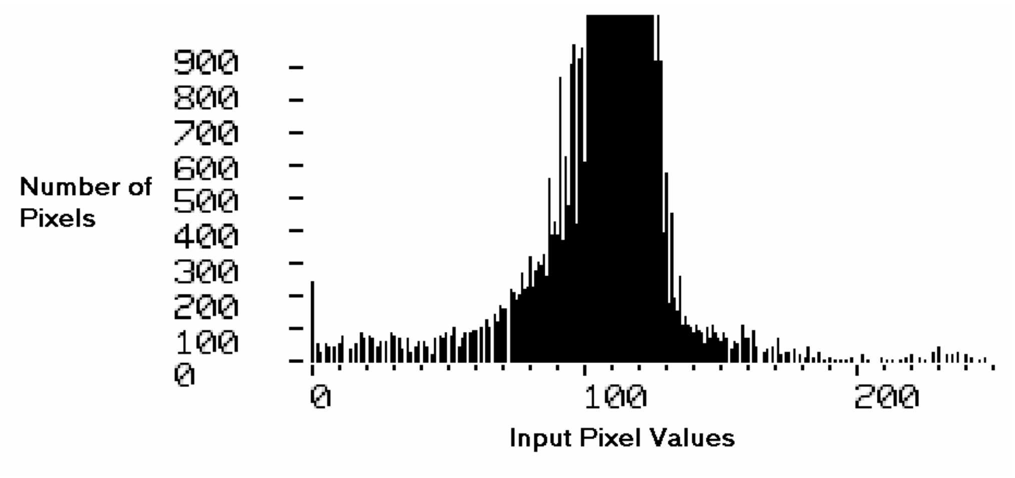


Figure 4.4 Example of input value histogram, Devir and Tridensky.

values less than a certain input value to the total number of pixels) and linearly mapping it to color-map internal variable (Devir and Tridensky). The equation for this mapping is

$$HE(g) = \frac{\left|\{x \in I : x \leq g\}\right|}{|I|} \qquad (4.1)$$

where I contains the intensity values for all pixels in the image, g is an input (intensity) value and x is the intensity value of a pixel from the image. HE(g) is the ratio of the number of pixels that are associated values below the value of g to the total number of pixels. HE is also called accumulated histogram of g. For example, if 65 percent of pixels have values less than 50 and internal variable goes from 0 to 1, the input value of 50 is mapped to color associated with internal variable 0.65.

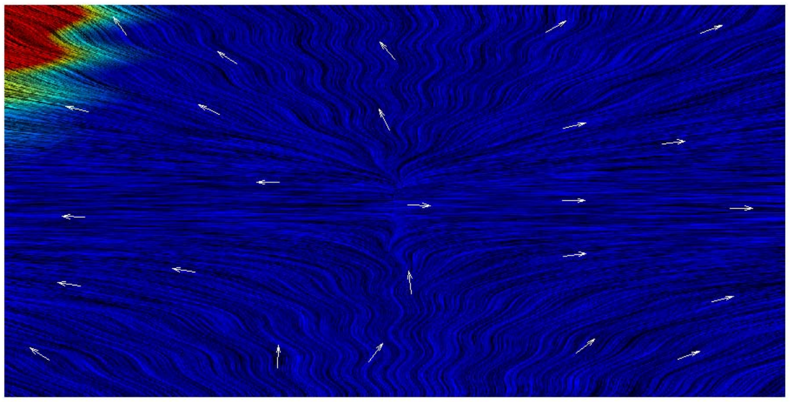
Both images in Figure 4.5 show the same data (generated by an analytical equation) visualized using two types of color-map mapping functions. Comparing Figure 4.5 top and bottom, the difference between these two color-mapping methods is clear. In Figure 4.5 top, most of the colors are assigned to high values on the top left corner of the image, and the majority of pixels are assigned with only one color. On the other hand, in Figure 4.5 bottom, colors are properly distributed to pixel values, resulting in a high color contrast image which is better suitable for scientific investigation. In this example, the histogram equalization image reveals the presence of oscillations in the field that are not evident from the linear color-map image.

Considering that a good portion of interesting phenomena often happen around the minimum and maximum points, it is sometimes desirable to gain more color contrast in those regions so they can be more easily investigated. A Gaussian distribution function can provide such characteristics. Histogram determination is an addition to the histogram equalization method, modifying the mapping function after histogram equalization (4.1) by the inverse Gaussian error function (Devir and Tridensky)

$$HD(g) = \frac{erf^{-1}((g-\frac{1}{2}).2erf\sigma}{2\sigma} + \frac{1}{2} \qquad (4.2)$$

In equation 4.2 σ is the standard deviation of the Gaussian curve. As σ increases, colors shift toward the middle of the spectrum and contrast reduces. Figure 4.6 shows several histograms; notice how increasing σ accumulates values closer to the center of the graph thus decreasing the image contrast.

After LIC with histogram determination was implemented in the Delphi environment, results using several values for σ (in equation 4.2) were compared to each other. For



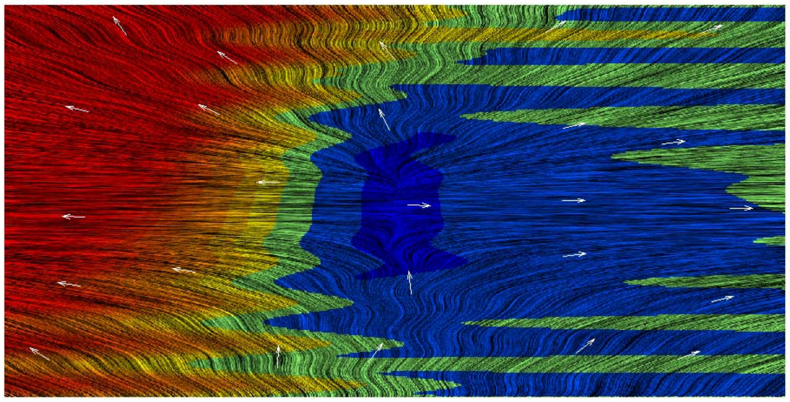


Figure 4.5 (Top) LIC image colored using linear color-map function (bottom) same image colored using a histogram equalized color-map function.

available earthquake simulation data, σ = 1.2 was found to be the best choice since near-singular behavior of crack tip and wave fronts were visibly distinguishable but overall contrast was still in an acceptable range. In general, I have observed that it is better to keep σ value less than 4 to minimize the decrease in image contrast.

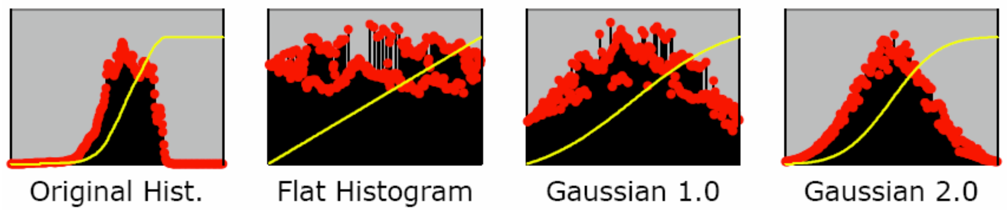


Figure 4.6 Histogram equalization and histogram determination, Devir and Tridensky.

In order to produce animation from LIC images of simulation time-frame data, white-noise input texture has to be kept constant for all the frames to preserve temporal coherency. Also since LIC images cannot convey the direction of the vector field by themselves, arrows can be placed on fixed locations during animation.

A problem specific to the animation problem is that even though histogram determination significantly improves color contrast in each frame; it also makes it hard to compare colors from different frames, since each is colored using a different mapping function. Based on numerous experiments with images from an earthquake rupture simulation, I found that this problem can be avoided through a form of moving window averaging. In this approach, we replace the histogram of each frame used in histogram determination algorithm by the average histogram of several animation frames. These frames can form a *moving window* behind the current frame resulting in a smooth change in the mapping function or they may span the whole length of animation resulting in a constant mapping function for all frame. Based on numerical experiments, I found that performance can be increased by use of frame sampling (averaging over a handful of sample frames behind the current frame) and iterative averaging (similar to iterative averaging method described in section 3.3, but using adjacent frame histograms instead of pixels).

An optimized iterative averaging method for a sampled moving window is implemented for Delphi visibly improving quantitative comparison among Animated LIC images. It calculates the average histogram from a sample chosen as every 10 frames in a 70 time-frame window before the current time-frame. Average histogram in each step is calculated by subtracting the earliest time-frame histogram from last average histogram and adding current-frame histogram to it.

4.2 MATLAB

The second implementation of LIC was in Matlab language. Codes for both regular and fast methods are written using Matlab internal language commands and Image Processing toolbox.

Due to Matlab data structure and operations, random access to array values is time-consuming. Since LIC manipulates arrays in that manner, Matlab implementations of LIC using internal functions are noticeably slow. For example a 750 x 375 output image with box kernel of length 10 pixels takes about 13 seconds with the Regular LIC method and 34 seconds with the Fast LIC method; the reason that ‘fast’ method takes longer is it has a larger overhead in Matlab (uses a lot more array manipulation) and it actually runs slower for small output and kernel lengths.

As the performance of the code written within Matlab, using internal Matlab functions was not satisfying, I decided to rewrite it as an external Matlab function (‘.MEX’ file) in C language. Doing so significantly improved the performance of the code, as Regular LIC written in C calculates the same 750 x 375 output image with box kernel of length 10 pixels in less than 3 seconds. Fast LIC also offers a better performance when implemented in as an external function The performance comparisons will be given in more detail in Section 4.3.

A robust Fast LIC code implemented in external C functions wrapped in internal Matlab programs combined with histogram equalization, Animated LIC and zoom capabilities is packaged as Matlab VFV toolbox (VFV stands for Vector Field Visualization) which is developed by this author.

To implement a histogram equalized color-map, first the *histeq* function from the Matlab Image Processing Toolbox is used to equalize the histogram of an intensity image containing the magnitude of vector field at each point. Then a Matlab color-map matrix produces a color image based on the equalized magnitude image.

To this point, output image only consists of information regarding the magnitude of vector field points; to incorporate an LIC image into it, it is transformed into HSV color space and the LIC image is put in the Value color channel. This modifies image colors, reflecting both LIC and magnitude information on the one output image.

Directional information can be added either by drawing arrows on select positions or by making an animation using Animated LIC technique discussed in section 3.4. Both options now exist in the VFV Matlab toolbox. It also facilitates exploration of vector fields by zoom-in, zoom-out features in a GUI environment. In Figure 4.7 left you can see a snapshot of toolbox GUI. An example of toolbox visualization is shown on Figure 4.7 right. The visualization shows a surface snapshot of horizontal velocity vector field from a simulation of a magnitude 7.7 earthquake on the southern San Andreas Fault courtesy of the Southern California Earthquake Center TeraShake earthquake simulation team.

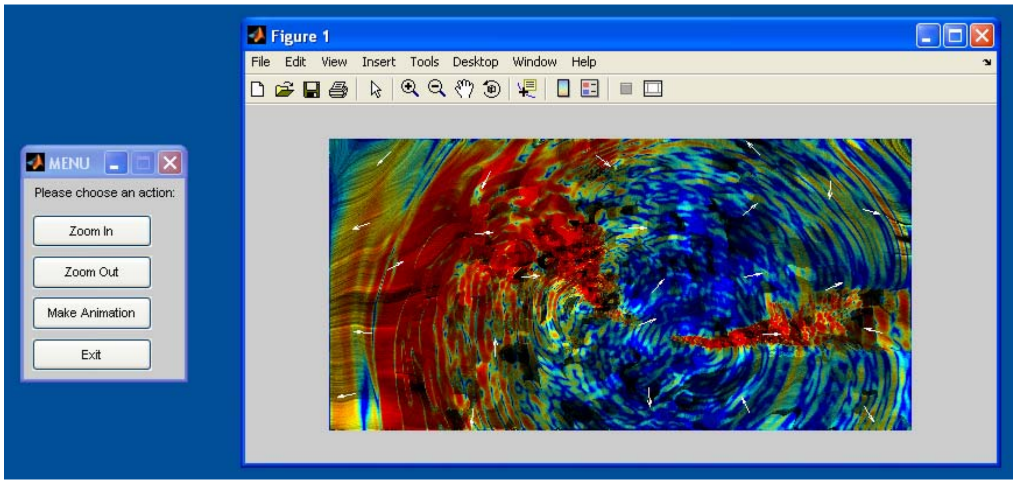


Figure 4.7 Snapshot of developed matlab toolbox, data is the courtesy of the southern California earthquake center Terashake earthquake simulation team and it represents a surface snapshot of horizontal velocity vector field from a simulation of a magnitude 7.7 earthquake on the southern San Andreas fault.

4.3 Performance Analysis

It is useful to compare performance of different LIC implementations. Table 4.1 contains execution time of a 724 x 324 pixel vector field with box kernel LIC length of 10 pixels. Interestingly, in this case all Fast LIC methods are slower than Regular LIC, according to Table 4.1. This is because with the short LIC kernel length of 10 used in this benchmark, here the overhead of Fast LIC method has more effect on performance than its speed in handling large input textures.

Figure 4.8 shows how Fast LIC takes over in speed as the number of input pixels (and proportionately kernel length) increase. Kernel length is set to one tenth of the largest image dimension in this figure. As the figure shows, execution time forms a line in logarithmic axis for both algorithms, the order or 'power-law exponent' of these methods can be calculated from the slopes of these lines. Based on Figure 4.8, Regular LIC execution time increases with input pixels with power 1.36, whereas this power for Fast LIC is just 1.00. This means that the number of operations in Fast LIC is linearly dependent on the number of input pixels. Figure 4.8 shows a cross-over between execution time of Fast LIC and Regular LIC methods for input values at about 95296 input pixels. This corresponds to input images with size of 308 x 308. For images smaller than this size Regular LIC is faster, but Fast LIC takes over for larger images. Also since increasing kernel length slows down Regular LIC more than

Fast LIC, cross-over input pixel number decreases with increase in the ratio of kernel length to largest image dimension.

Table 4.1 LIC Execution Time with LIC Kernel Length = 10 Pixels

	Delphi Regular LIC	Delphi Fast LIC	Internal Matlab Regular LIC	Internal Matlab Fast LIC	External Matlab in C with Regular LIC	External Matlab in C with Fast LIC
LIC execution time in seconds	0.845±0.013	1.34±0.02	13.718±0.022	77.883±1.195	1.886±0.016	2.61±0.01

Table 4.2 shows execution times for kernel length of 50, for all the LIC codes developed for this project. Each Fast LIC code, with the exception of the internal Matlab Fast LIC code, now runs faster than the corresponding Regular LIC code. A possible reason that this does not apply to the internal Matlab implementation of LIC is because Matlab has a low performance in executing ‘for’ loops and random access to matrix indices, while Fast LIC extensively uses these internal commands (much more than Regular LIC).

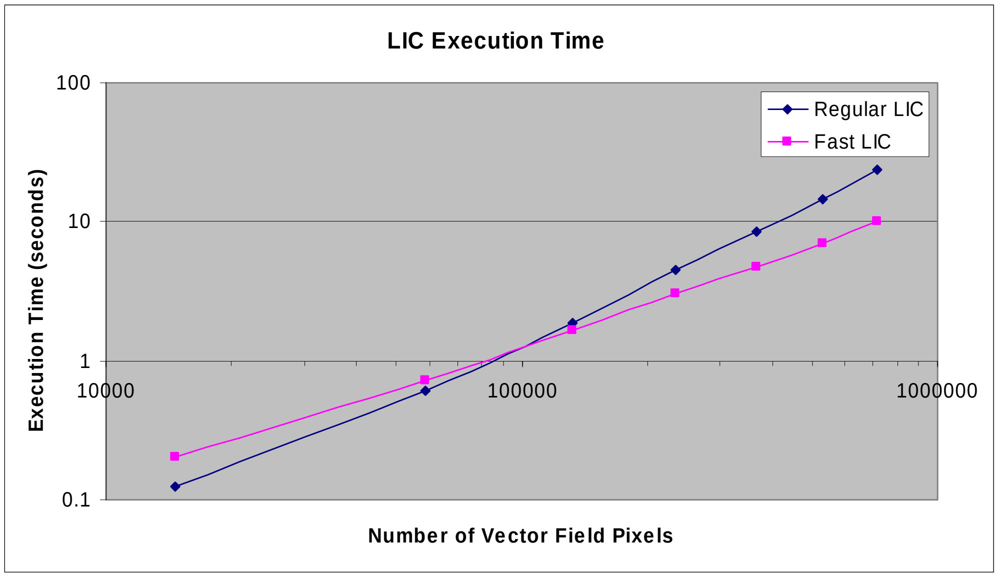


Figure 4.8 Executions time versus number of vector field pixels shown in logarithmic axis.

As discussed in section 3.4, longer kernel lengths result is improved coherency in output image. But the comparison of execution times in Tables 4.1 and 4.2 shows that the longer

Table 4.2 LIC Execution Time with LIC Kernel Length = 50 Pixels

	Delphi Regular LIC	Delphi Fast LIC	Internal Matlab Regular LIC	Internal Matlab Fast LIC	External Matlab in C with Regular LIC	External Matlab in C with Fast LIC
LIC execution time in seconds	3.28 ±0.01	2.18± 0.013	72.25±0.01	84±3	8.89±0.04	3.38±0.17

kernel also significantly increases the execution time. Traditionally one tenth of the largest image dimension is chosen as kernel length, but if the image is large so kernel length should exceed 50 pixels, Based on my experiments, it is advisable to use iterative methods and break the process up to two consecutive LIC runs with contrast enhancement applied in between (Iterative LIC) to improve contrast and coherence in the output image (see Figure 3.11 which shows a comparison between these two methods for kernel length of 50 pixels).

CHAPTER 5

APPLICATION

Earthquake simulations usually generate several high resolution vector fields, i.e. velocity, displacement and energy flux. Proper understanding of earthquake dynamics requires meticulous investigation of these fields, in which visualization plays an important rule.

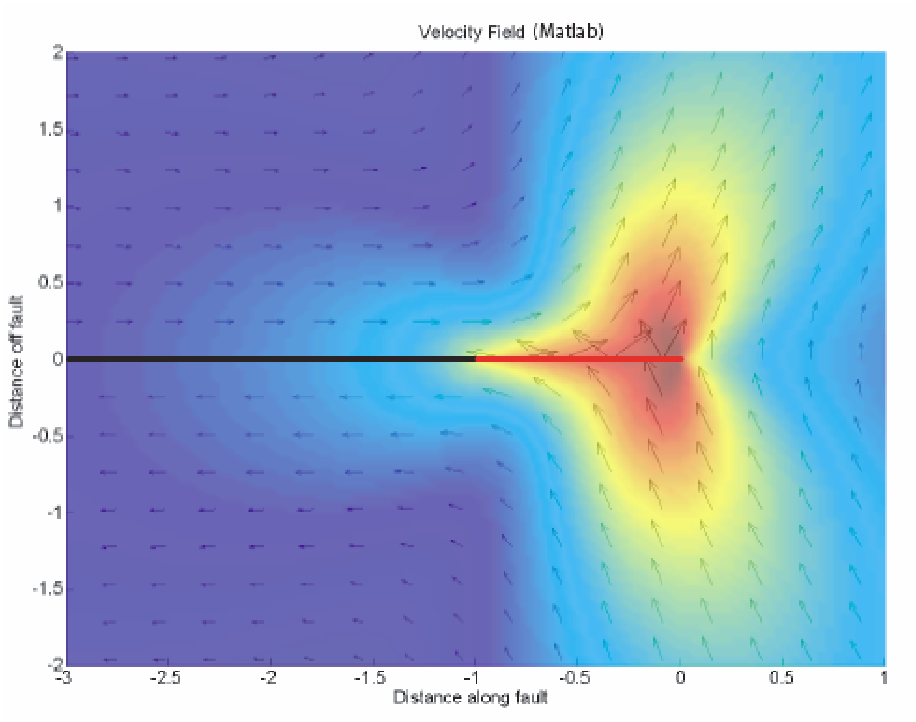
Several datasets has been visualized using the implementation of the LIC algorithm developed here. By comparing these results with traditional visualization methods, the usability of the toolbox can be evaluated.

5.1 Steady State Crack Propagation

A simple, easily calculated model that contains many of the features common to large, complex earthquake rupture propagation simulations is provided by the steady state crack propagation code developed by Dunham, et al. (Dunham and Archuleta, 2004). It calculates physical quantities, including particle velocity and energy flux, for a constant-length crack propagating with a constant speed. Both sub-shear and super-shear propagation speed modes are simulated. Sub-shear means that the crack propagation speed is less than the shear wave speed of the medium, while super-shear means that the crack propagation speed exceeds the shear wave speed.

Figures 5.1 and 5.2 both show the velocity field around the tip of a semi-infinite Mode II (inplane shear mode) crack, propagating to the right at a constant velocity of 0.8 times the shear wave speed. The slip zone, marked by the solid black line extends from 0 to -infinity. The fields are non-singular at the crack tip, which is accomplished by introducing a finite length scale (here set to unity) over which the shear traction breaks down. The breakdown zone is marked by the solid red line, and its length determines the concentration of the fields surrounding the crack tip.

Figure 5.1 top is generated using Matlab original vector field visualization commands. It shows direction by a Glyph based method and demonstrates intensity



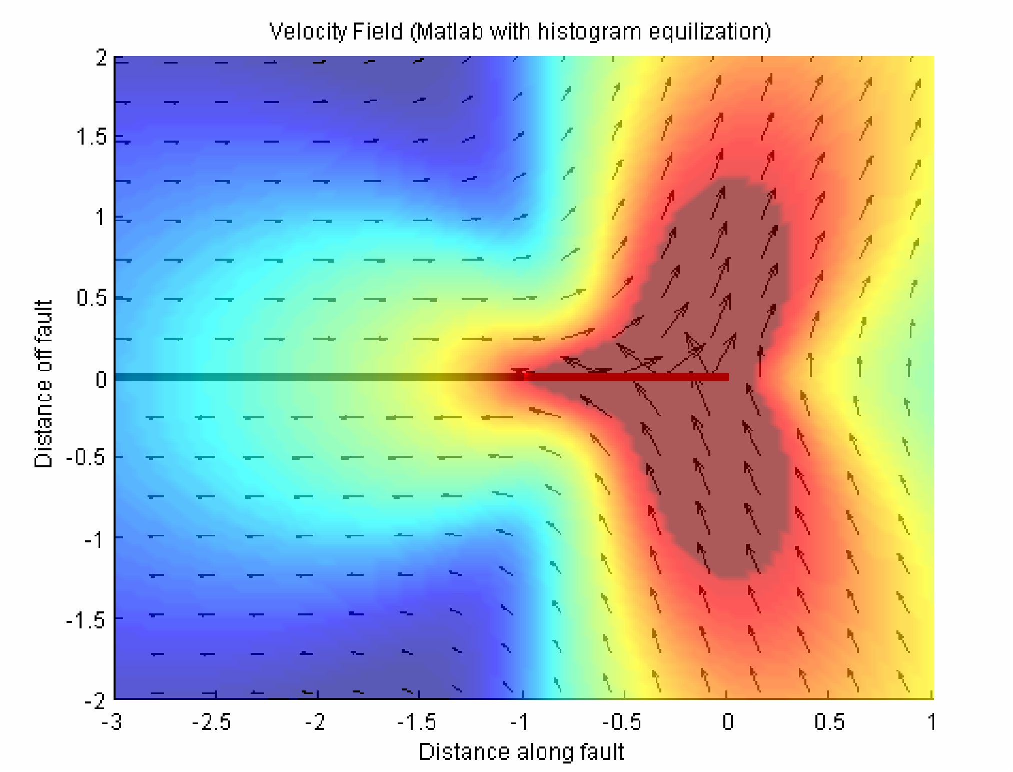


Figure 5.1 Matlab visualization generated from velocity field of a semi-infinite mode ii crack (top) using linear color-map (bottom) using histogram equalized color-map.

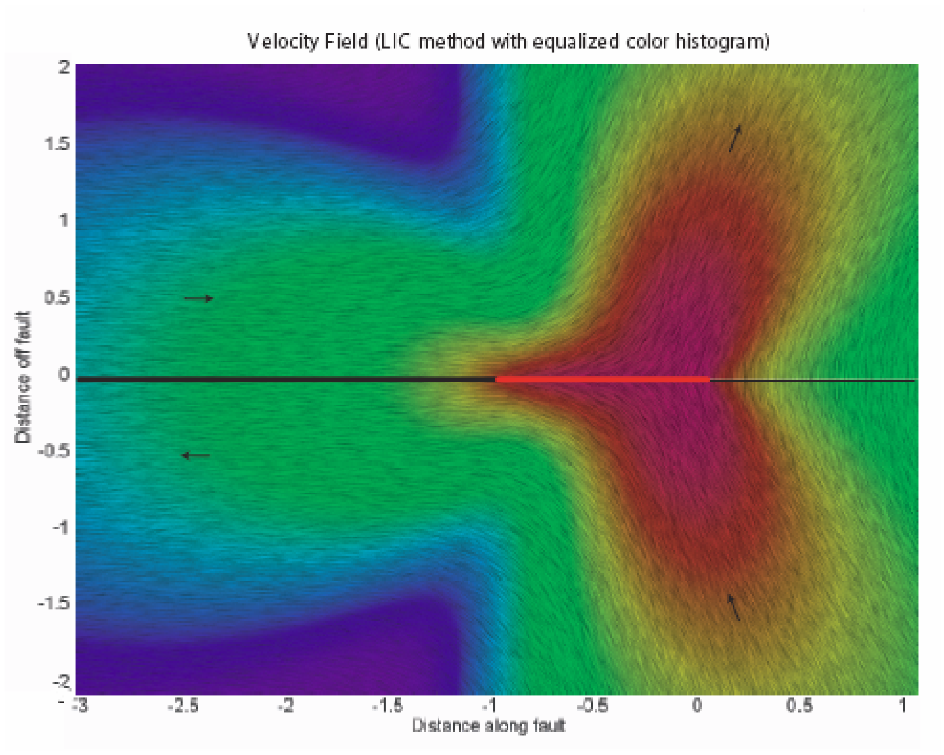


Figure 5.2 LIC visualization generated from velocity field of a semi-infinite mode II crack.

by a linear color-map. Figure 5.1 bottom also uses Matlab original commands but its color-map is histogram equalized. Figure 5.2 is produced by the LIC method with equalized color histogram. Comparing these figures reveals the advantage of LIC in demonstrating the direction of the field at every point. Note in particular how the overlapping arrow glyphs near crack tip in Figure 5.1 obscure the details of the velocity field in the breakdown zone, whereas these can be followed clearly in the LIC image in Figure 5.2.

The energy flux vector field provides additional information on the behavior of crack models. Since the models considered here are linearly elastodynamic (except on the crack plane itself), the energy flux calculation can be simplified by neglecting terms in the energy flux higher than 2nd order in the velocity and displacement-gradient components. This simplification leads to the expression $\varepsilon = -\sigma \cdot v$ where σ is the Cauchy stress tensor, v is the particle velocity vector and ε is the energy flux vector. To provide a second example for comparison of the glyph and LIC approaches, the energy flux vector field for the crack problem has been calculated from this expression. Results are shown in Figures 5.3 and 5.4, for the standard Matlab and LIC codes, respectively.

To see benefits of histogram equalization, please pay close attention to the tip of crack in Figures 5.3 and 5.4. In Figure 5.3 top, which uses a linear color-map, it is marked only by a faint bright blue color, not revealing any significant energy flow structure; flow intensity structure at the tip of crack is noticeably better visualized in Figure 5.3 bottom which is histogram equalized but sparse glyphs on the image are unable to depict directional structure near the crack tip. On the other hand, Figure 5.4 shows the energy flow pattern (both intensity and direction) near crack tip in great detail.

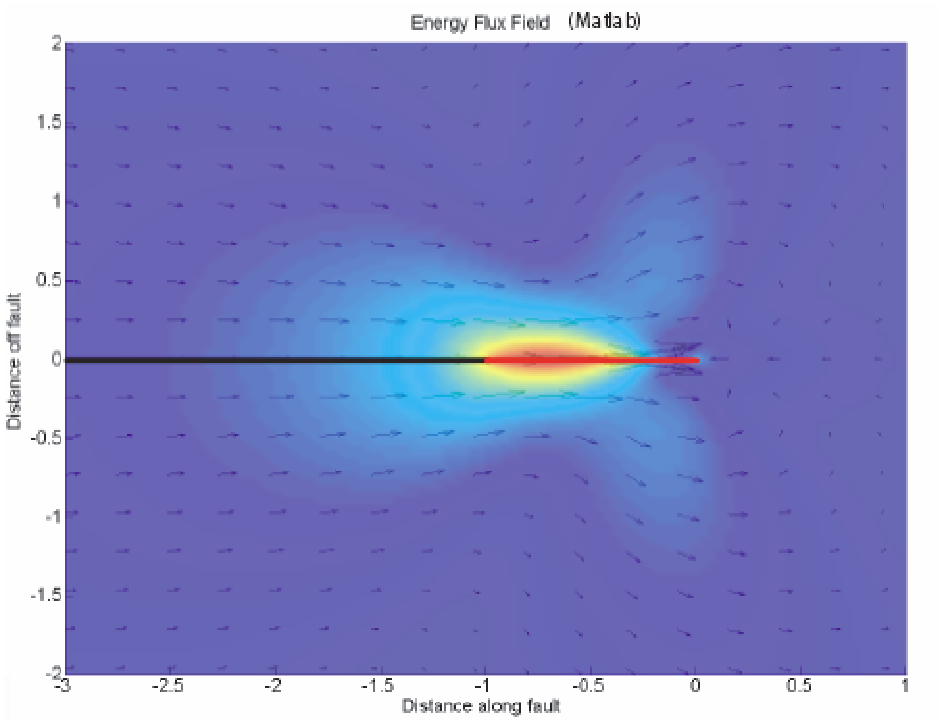
5.2 Variable Speed Crack Propagation

A more elaborate model for crack propagation is provided by the finite difference simulation method of Day (Day, 1982). It simulates a crack propagating with a variable speed on a plane embedded in a homogenous elastic body. The speed is calculated based on realistic physics calculations at each moment.

Two dimensional velocity, displacement and energy flux fields are computed in different time-frame sequences. Then each time-frame is visualized using the LIC method implemented in Delphi with a constant white noise input texture (to preserve coherency between frames). These frames are then put together in an AVI file forming a computer animation showing the dynamics of crack propagation.

Selecting a color-map was found to be a challenge in this case. The problem is that keeping the color map fixed for all animation frames provides the comparability needed for qualitative investigation, but since it has to span from the minimum to maximum magnitude value among all frames, it significantly reduced color-contrast in individual frames. To make adjacent frames comparable, a window averaged histogram color-map is used, as discussed in more detail in section 4.1).

Figures 5.5 (p. 39) and 5.6 (p. 40) both show the velocity field of a crack propagating from left to right in one of the simulation time-frames. The crack is horizontal, running left to right through the center of the frame. Both figures are zoomed onto the region around the tip of the crack, where field magnitude and direction change rapidly. Figure 5.5, which is produced by Matlab default commands, demonstrates the general field structure. Figure 5.6,



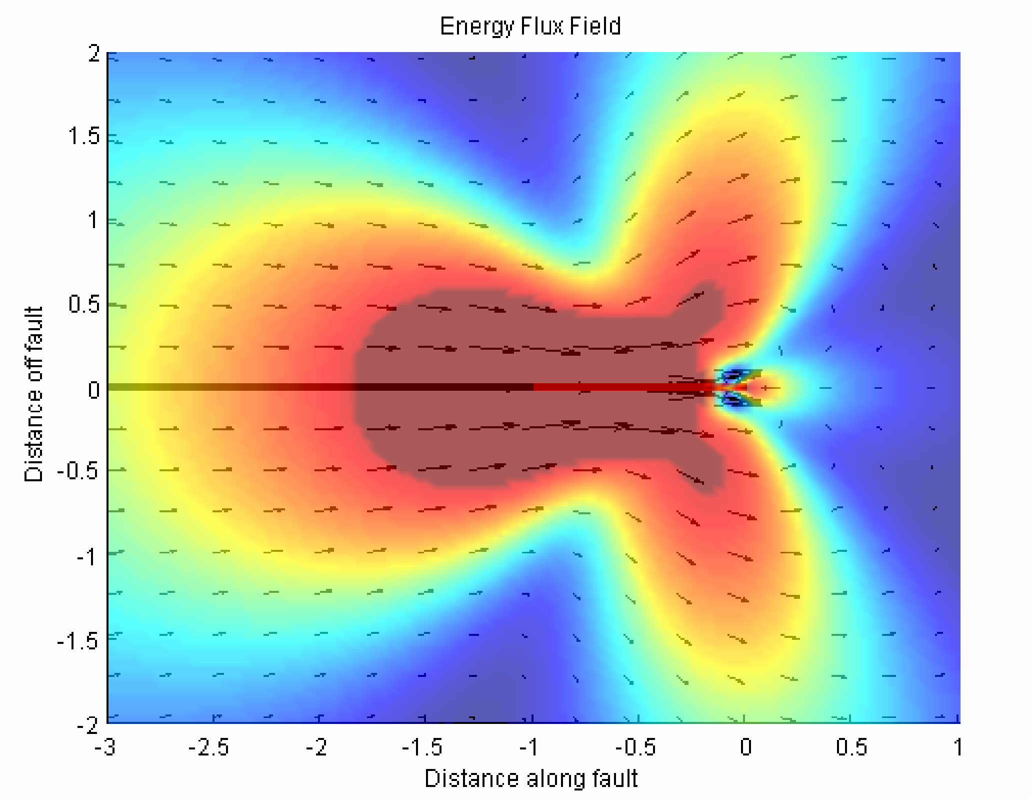


Figure 5.3 Matlab visualization generated from energy flux field of a semi-infinite mode ii crack (top) with linear color-map (bottom) with histogram equalized color-map.

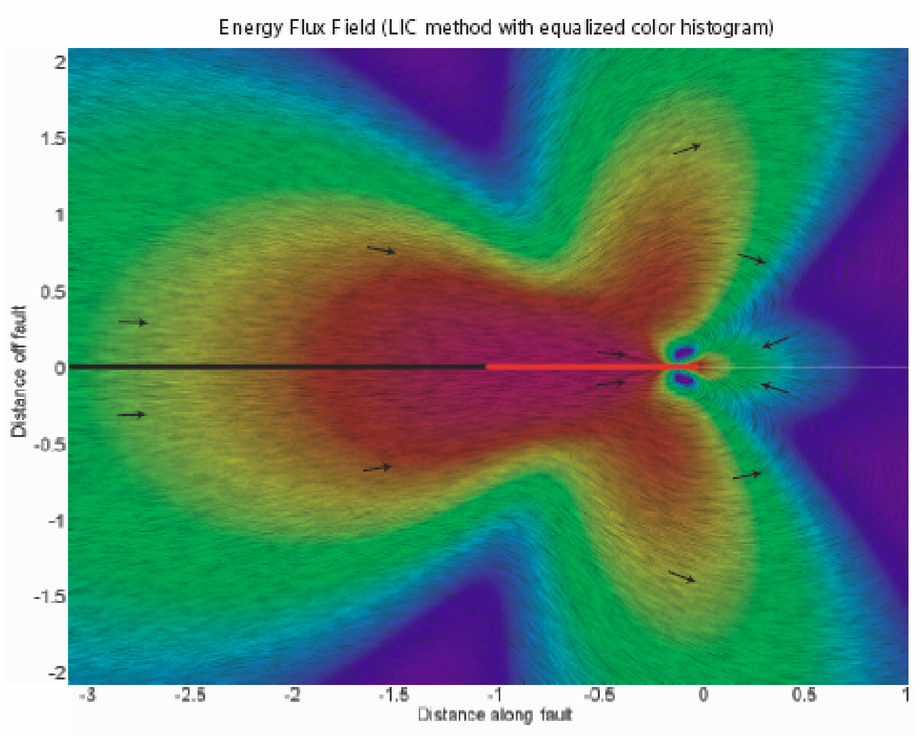


Figure 5.4 LIC visualization generated from energy flux field of a semi-infinite mode II crack.

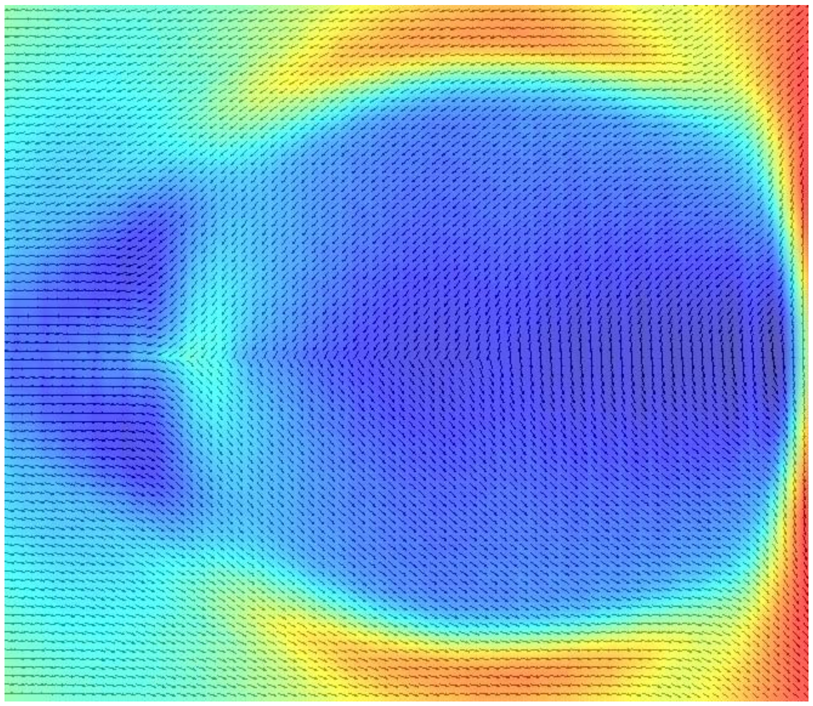


Figure 5.5 Matlab visualization generated from velocity field of variable velocity crack propagating left to right.

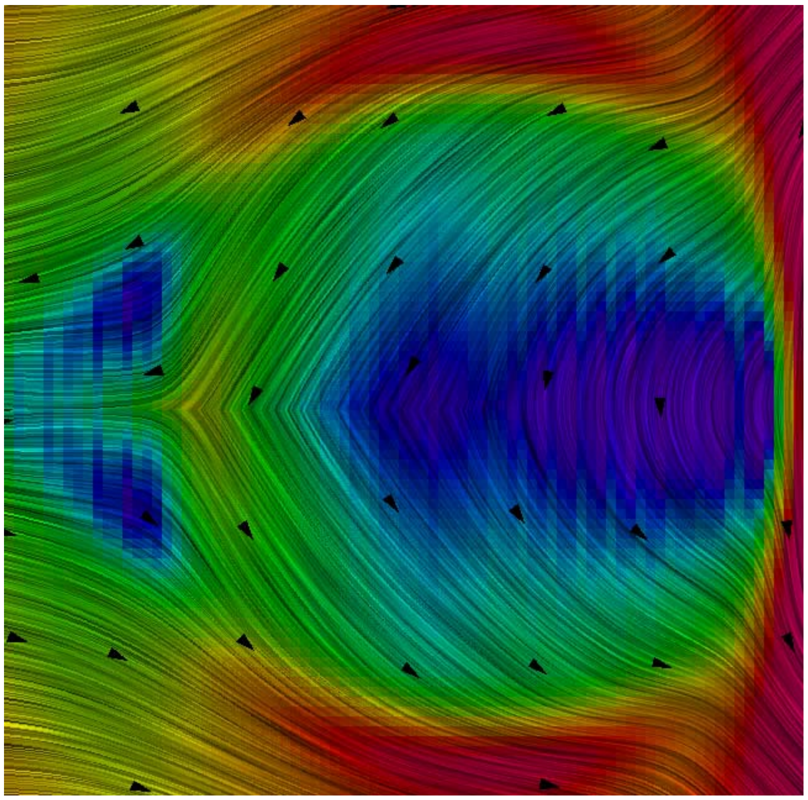


Figure 5.6 LIC visualization generated from velocity field of variable velocity crack propagating left to right.

produced by LIC with histogram equalization noticeably does a better job visualizing both direction and structure of the vector field. There are several features that are visualized more clearly in Figure 5.6: rapid change in direction occurring on the crack plane, about 20% of the way across the figure from left to right—this is right at the crack tip; motion predominantly normal to fault ahead (to right) of crack tip; motion predominantly parallel to fault behind (to left of) crack tip.

5.3 Bimaterial Simulation

Another interesting simulation, using the same numerical method as that in the previous section, is a crack propagating in a heterogeneous environment. In this case, fault line separates two half-spaces with two different wave speeds, as in the work of Harris and Day, 1997. Figures 5.7 (p. 42) and 5.8 (p. 43) show two different computed quantities, velocity and energy flux, from two time-frames. The fault (white line) separates half-spaces with different wave speeds (lower half-space is 33% slower than upper half-space). Red line indicates actively slipping part of the fault. Figure 5.7 demonstrates an early stage of propagation (t =0.89 s), while Figure 5.8 corresponds to a later stage (t = 0.154 s).

Several features are revealed in these images; For example in Figure 5.8 top you can see three vortices in the velocity field, two at each side of the red line (actively slipping part of the fault) and one at the bottom center of the figure. Since velocity-field curl travels with S speed, these features can be used to identify S waves radiating from the crack. Energy field at Figure 5.8 bottom confirms this; since the energy flux is outward, away from the fault, it represents a radiating wave.

5.4 TeraShake

TeraShake is the largest and most detailed simulation of what may happen during a major earthquake on the southern San Andreas fault that has been done to date. It models the earth shaking that would occur in Southern California if a 230 kilometer section of the San Andreas fault ruptured from north to south, beginning near Wrightwood, California and producing a magnitude 7.7 earthquake. This simulation uses a 3,000 by 1,500 by 400 mesh, dividing the volume into 1.8 billion cubes with a spatial resolution of 200 meters on a side, resulting in the maximum frequency of .5 hertz. It is the most detailed simulation of this region to date (Tooby).

Surface velocity is an important factor in study of earthquake impact on structures. Since data generated by TeraShake is complex and in high resolution, the Matlab VFV toolbox developed here can help in exploring it by providing an interface with vector field visualization, zoom and animation features.
Figure 5.9 shows horizontal velocity vector field from a TeraShake time-frame (3000 x 1500 grid). High velocities are painted with red and yellow, while lower velocity magnitudes are colored with blue. Seismic waves can be seen emitting outwards from ruptured section. To further explore velocity patterns, we use zoom-in capability of the Matlab VFV toolbox and magnify a small region around left rupture tip (the white box in Figure 5.9 indicates zoomed area). Figure 5.10 shows the turbulent velocity structure in that region. This turbulent appearance is due to reverberation of waves excited by the rupture tip as it enters the low seismic velocity sediments of the San Bernardino basin. The nature of these reverberating

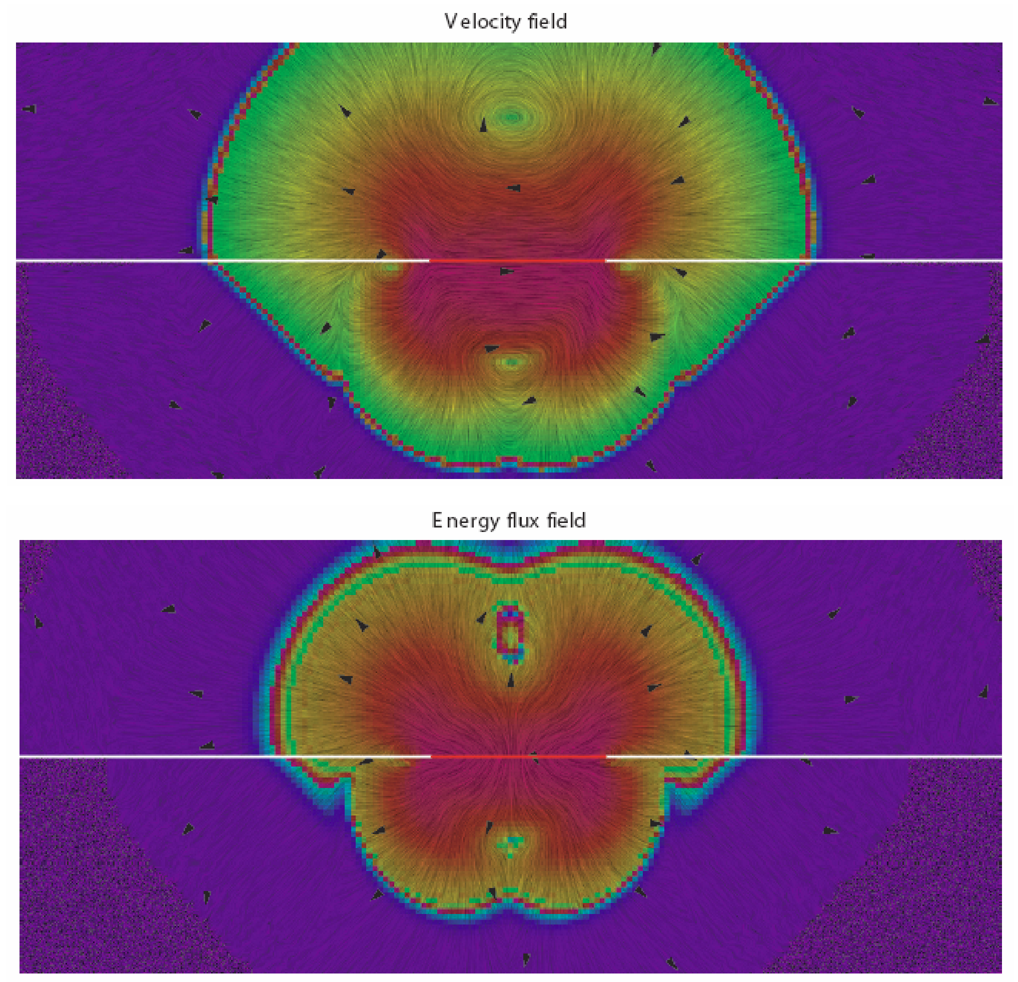


Figure 5.7 LIC Visualization of particle velocity (top) and energy flux (bottom) vector field for a mode II bimaterial simulation at t = 0.89 s.

waves is very complex, and the high-resolution vector images may help seismologists understand how they are excited and how they propagate.

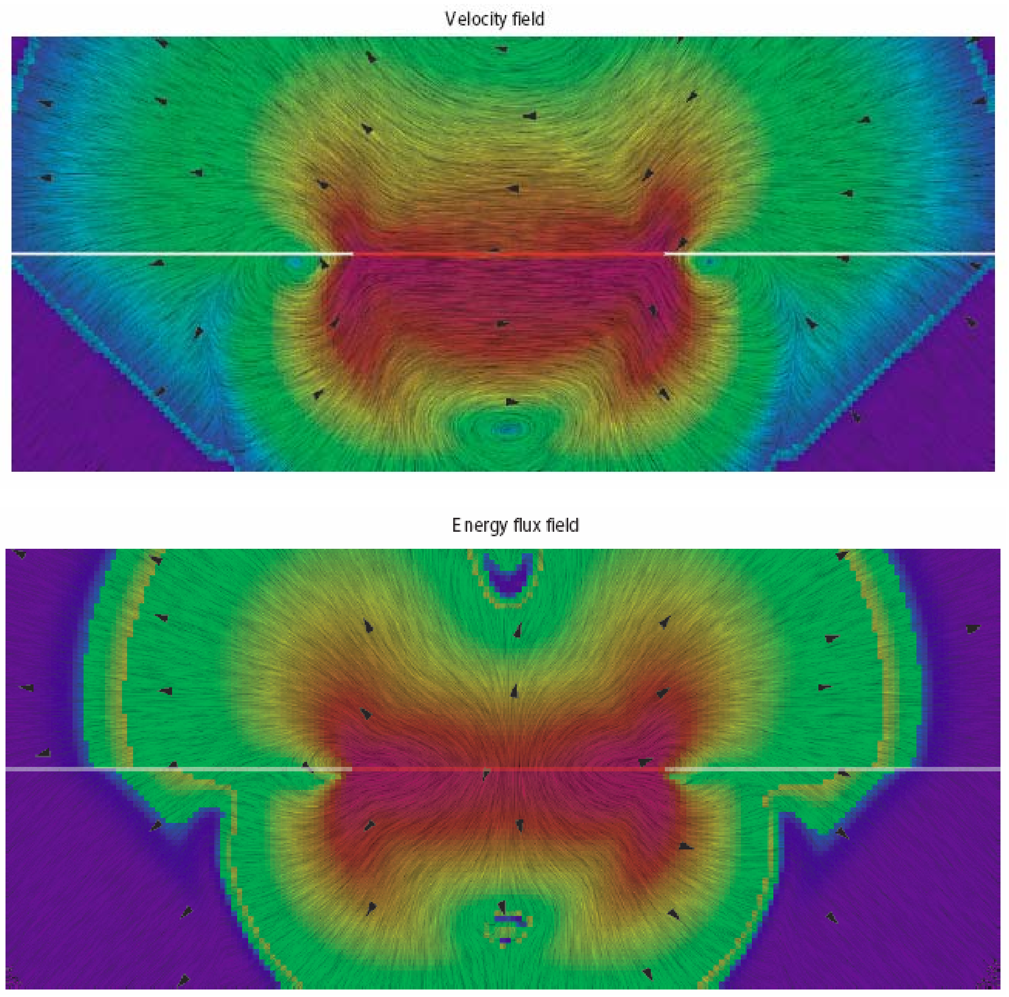


Figure 5.8 LIC Visualization of particle velocity (top) and energy flux (bottom) vector field for a mode II bimaterial simulation at t = 0.154s.

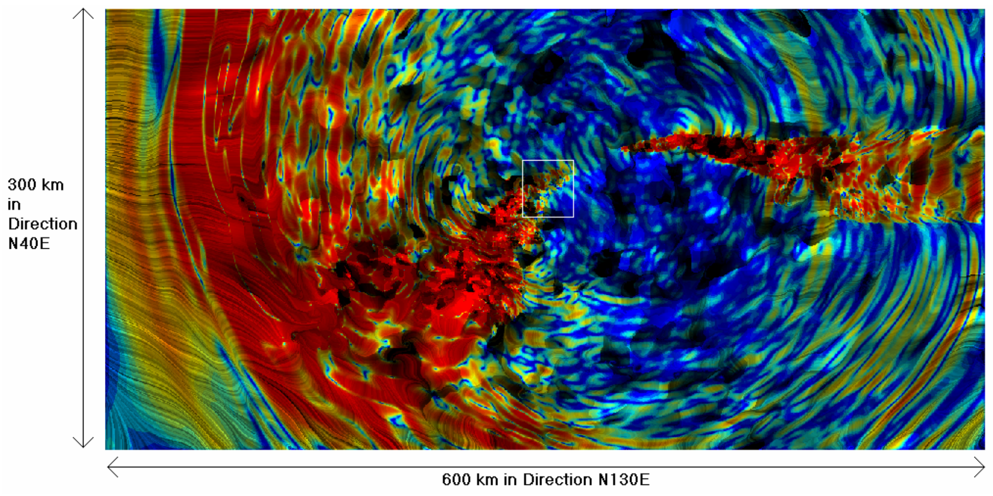


Figure 5.9 Surface horizontal velocity vector field visualized by Matlab VFV toolbox using iterative LIC.

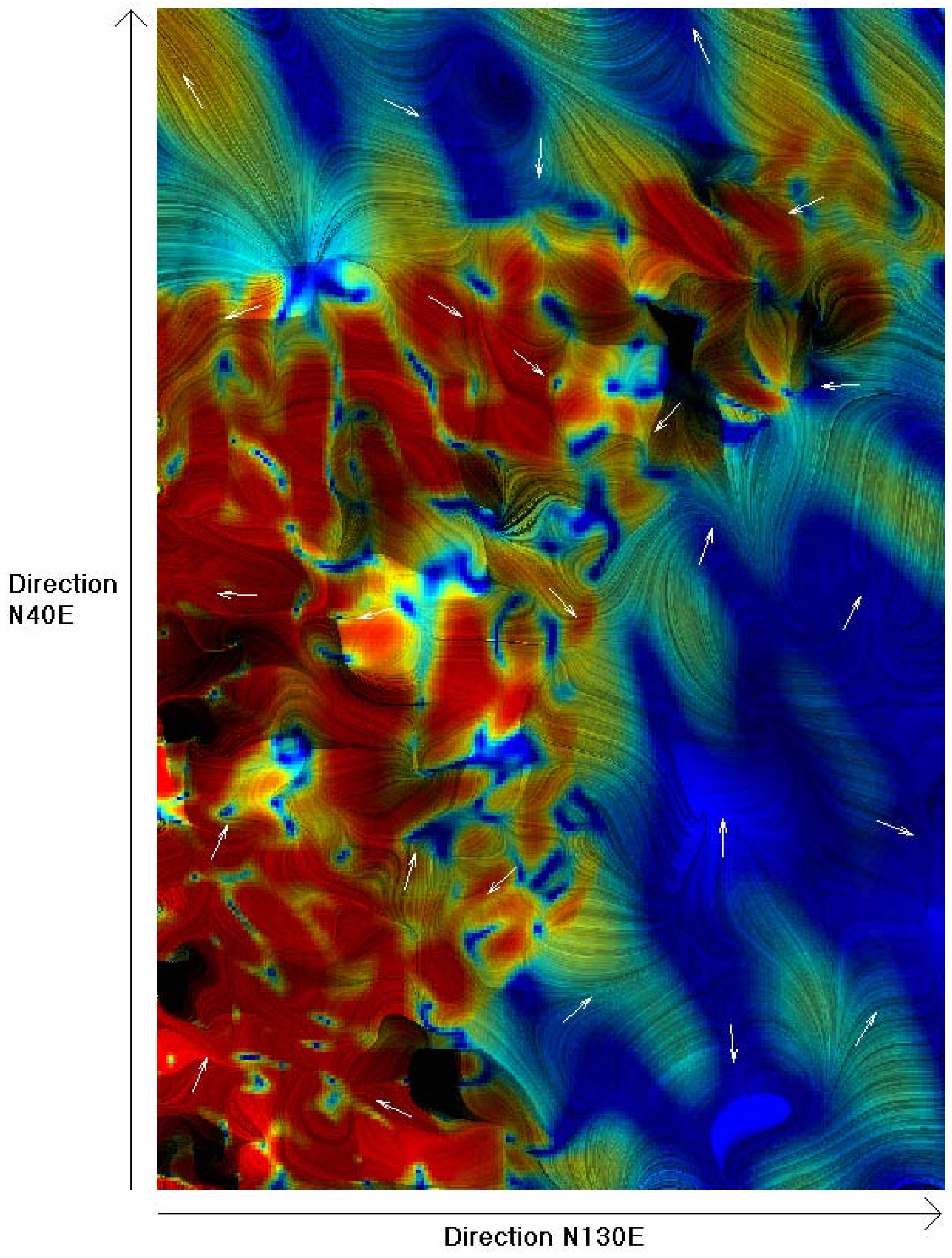


Figure 5.10 Complex velocity pattern near rupture tip.

CHAPTER 6

DISCUSSION AND CONCLUSIONS

In this chapter, a summary of earlier chapters will be presented. Several variations of LIC method for 3D and unsteady-flow visualization that might be of future value for earthquake rupture visualization will also be briefly discussed.

6.1 SUMMARY

As mentioned in Chapter 1, vector field visualization plays an important rule in scientific data interpretation. Among several methods discussed in that chapter, only texture methods are able to generate high resolution results. Availability of texture based method in common software packages used by scientist is investigated in Chapter 2. The lack of these features provided the motive to develop Matlab VFV toolbox.

In Chapter 3 we discussed several texture based methods. We saw LIC provides a better visualization quality than DDA and how LIC can be accelerated using Fast LIC method. An elaborate way to incorporate directional information into LIC images, known as Animated LIC was explained, and we learned how to preserve contrast of the output image using Iterative LIC.

Implementing several variations of LIC in different computer languages and experimenting with earthquake simulation data in Chapter 4, we saw how histogram equalization can improve visualization quality and enhance important features. To preserve comparability between different time-frames, a histogram averaging method was proposed and ways to improve its speed discussed. Finally we compared the performance of different implementations to each other and saw that Fast LIC takes over in speed in large images as its execution time is only linearly dependent to the number of input image pixels (unlike Regular LIC in which execution time is proportional to a large-than-one power of the number of input image pixels). Applying Matlab VFV toolbox on several earthquake simulations, I verified the usability of the toolbox for this class of problems in Chapter 5. LIC images generated by the toolbox provided more insights into crack propagation data than images produced by Matlab internal commands as we observed field structure in more clear detail.

6.2 Future Enhancements

The application of the toolbox is currently limited to 2D steady-flow vector field. Several new features can be added to Matlab VFV toolbox in the future; 3D vector field and unsteady-flow visualization are among the most useful future enhancements.

6.2.1 Three Dimensional LIC

Up to this point, we only discussed visualization of two dimensional vector fields; in many cases there is a need to visualize 3D fields in 2 or 3 dimensional domains. An example of a 3D field in 2D domain is the surface velocity field of an earthquake. Since buildings lie on earth surface, engineers and geologists are often interested to observe the three components of velocity on the ground surface plane. Stream-lines are not suitable for this task, as they tend to reach beyond the surface following the vertical component of velocity field; Glyph arrows cannot be used in high resolutions also, as they easily clutter the picture, as discussed in section 1 and illustrated in Figure 1.4. Overlaying an LIC image created from two horizontal components of the field over a topographic map generated by vertical component, all three components of the field can be visualized.

Another way to handle this problem is to use VolumeLIC. VolumeLIC is an extension of LIC in three dimensions; here a volumetric texture is blurred along field lines, then a volume rendering algorithm is used to generate a 2D image. Figure 6.1 shows Volume LIC pipeline.

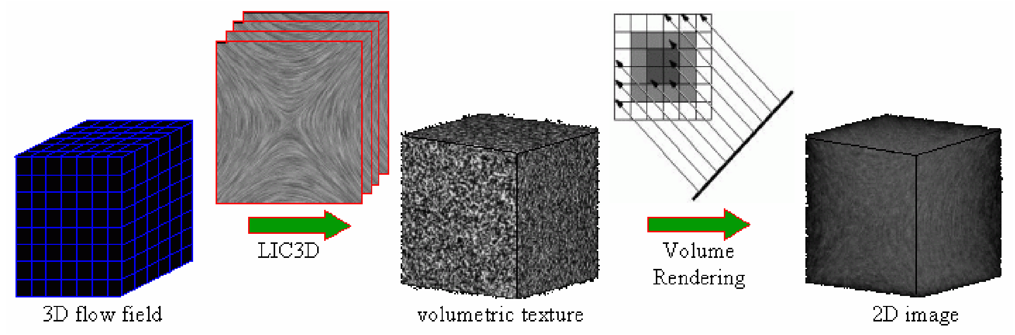


Figure 6.1 Volumetric LIC pipeline, Zhanping Liu, source: http://www.erc.msstate.edu/~zhanping/research/flowvis/lic/volumelic/volumelic.htm.

Several techniques can be used to improve visibility and speed of VolumeLIC (Helgeland, 2002; Helgeland and Anderson, 2004):

- Use of a clip plane to see inside 3D LIC texture. For example, in Figure 6.2 a horizontal plane has clipped the upper section providing a view into the 3D texture.

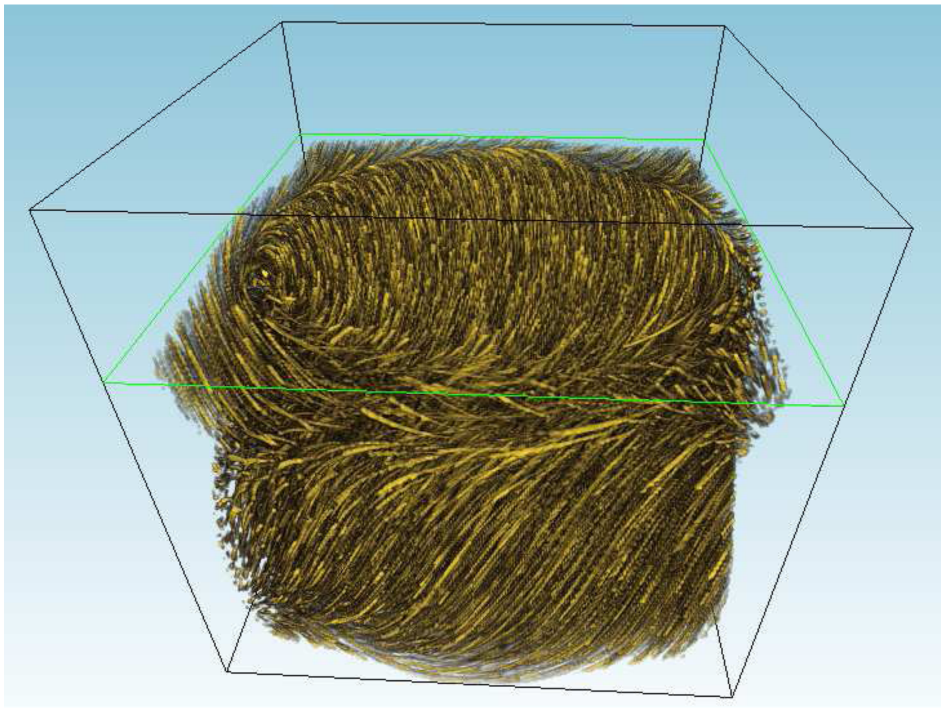


Figure 6.2 Clip plane reveals inside 3d LIC texture, Helgeland, 2002.

- Manipulate the opacity value according to field magnitude. Figure 6.3 shows how changing opacity (Alpha) according to field intensity can be used to see inside the 3D texture.
- Specify a region of interest (ROI): use of a scalar value (Temperature, Velocity magnitude…) to specify a region where we are interested to see information about the vector field. Figure 6.4 top (p. 49) shows white noise located at ROI. In Figure 6.4 bottom (p. 49) final 3D LIC image is shown as it is generated only in ROI.
- SeedLIC: Using seed points instead of input noise texture can significantly reduce calculation time in cost of losing some information. Figure 6.5 (p. 50) compares image quality and speed of Seed LIC and Fast LIC for a sample vector field. Here Seed LIC has significantly increased the speed with a minimum effect on visualization quality.

6.2.2 Unsteady-Flow LIC

As we saw in chapter 3, an LIC image is only capable of incorporating tangential information of the vectors and it fails to include direction. Since Animated LIC algorithm, describe it in section 3.4, employs apparent motion to include direction, it can only be applied to field vectors that are constant in time.

Until now, we had limited our discussion to steady-flow visualization, or we represented

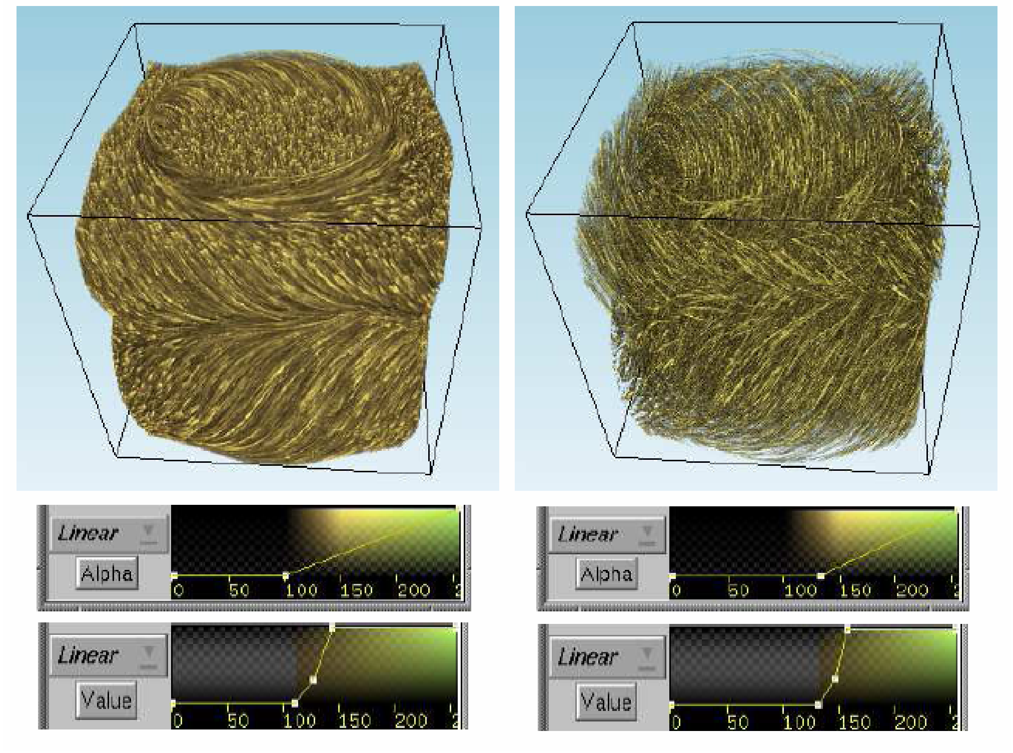


Figure 6.3 Dense LIC texture with different alpha (opacity) and value (magnitude) functions, Helgeland, 2002.

unsteady flow without direction information, by animation of LIC. To produce LIC animation from time-varying vector fields while keeping direction information, unsteady flow LIC algorithm (UFLIC) appears to provide a good alternative (Liu and Moorhead II, 2005), and could in future be tested for 3D extensions of the 2D rupture problems considered the Section 5. In UFLIC, particles are released at each pixel, depositing their property (e.g. texture value) down the stream as they flow over time. On the other hand, each pixel keeps receiving several particles over time and stores their values in a buffer; by convolving these values with frame index, spatial coherence is created in each output frame. Each frame is then high-pass filtered with noise jitter and fed forward as the input texture of next frames to create temporal coherence.

6.3 Conclusion

Linear Integral Convolution is a powerful texture method for vector fieldvisualization when high resolution is required to reveal field structure. Different

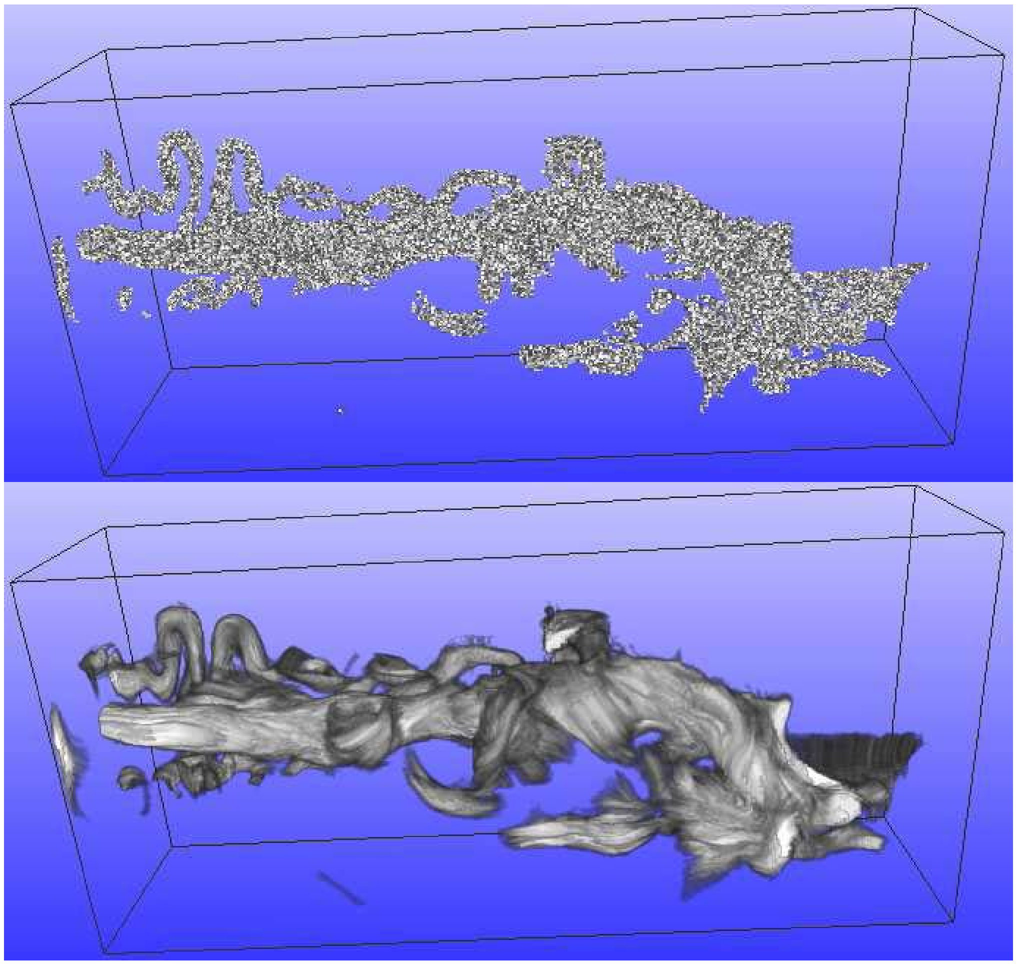


Figure 6.4 (top) White noise located at region of interest (bottom) texture produced by LIC from white noise at region of interest, Helgeland, 2002.

implementations of this algorithm are presented in this article (i.e. Delphi, Matlab VFV toolbox), as each of these provides certain advantages. Matlab VFV toolbox combines the flexibility of Matlab with the power and wide applicability of LIC, forming a portable extension to the Matlab environment. Scientist can now easily run VFV toolbox commands from their Matlab code and explore their data without prior knowledge of the LIC process.

Applying Matlab VFV toolbox to crack propagation problems reveals more structures than Matlab internal commands. In the cost of few seconds that are needed to generate each LIC image, the toolbox provides powerful exploration capabilities that are helpful in understanding crack propagation mechanisms.

Matlab VFV toolbox provides a complete set of commands to generate LIC images and animations arranged in a graphical user interface (GUI) for exploring vector fields with zooming capabilities. Further improvements can be made by adding a GUI for LIC options, commands for visualizing 3D fields over a plane using texture mapping and finally using OpenGL in external functions for VolumeLIC.

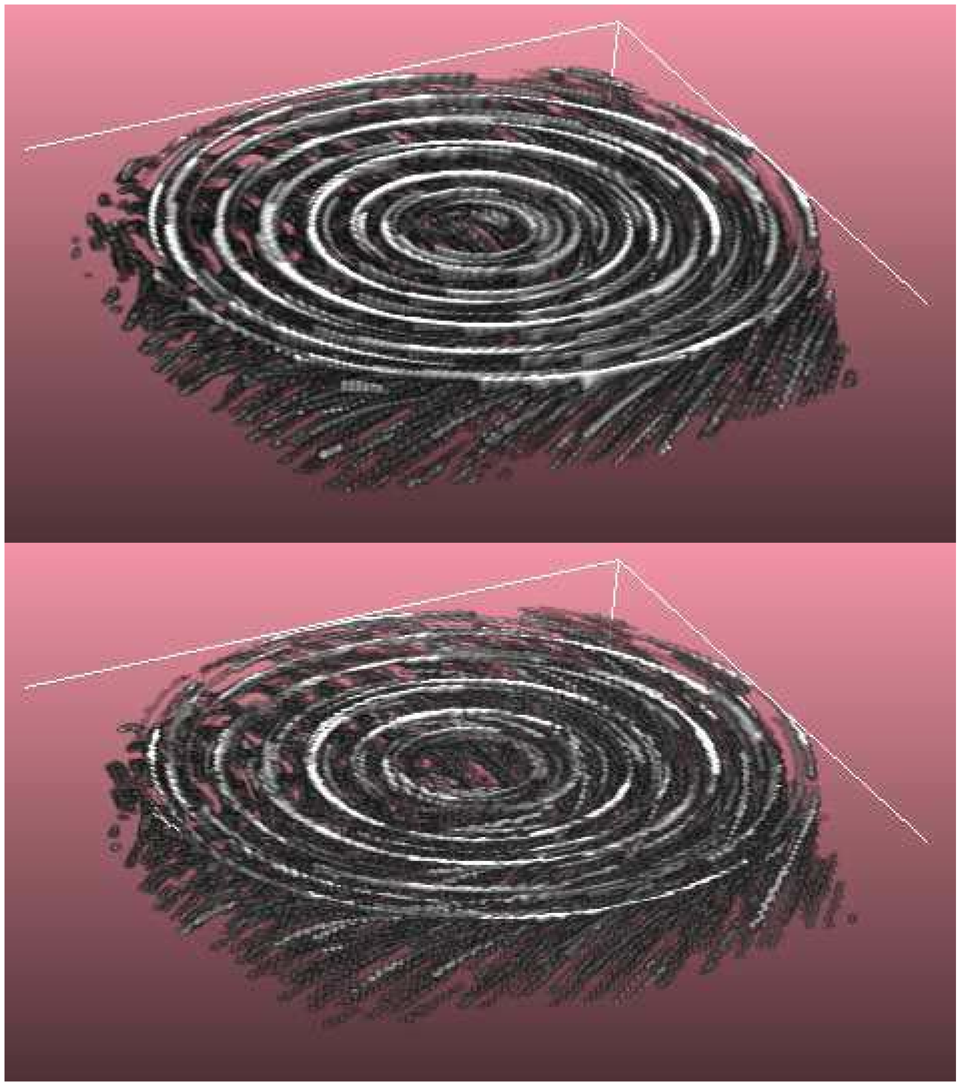


Figure 6.5 (top) Fast LIC takes 753 seconds to compute (bottom) Seed LIC takes only 4.7 seconds, Helgeland, 2002.

REFERENCES

Helgeland, A., 2002, Visualization of vector fields using line integral convolution and volume rendering. [Masters Thesis]: Norway, University of Oslo, Department of Informatics.

Wijk, V. J., 1991, Spot noise texture synthesis for data visualization, *Computer Graphics* v. 25, no. 4, p. 309-318.

Turk, G., 1991, Generating textures on arbitrary surfaces using reaction-diffusion textures, *Computer Graphics* v. 25, no. 4, p. 289-298.

Witkin, A. and Kass, M., 1991, Reaction-diffusion textures, *ComputerGraphics* v. 25, no. 4, p. 299-308.

Cabral, B. and Leedom, L., 1993, Imaging vector fields using line integral convolution, *Proceedings of ACM SigGraph 93*, Aug 2-6, Anaheim, California, p. 263-270.

Klassen, R.V. and Harrington, S. J., 1991, Shadowed hedgehogs: a technique for visualizing 2d slices of 3d vector fields, *Proceedings of IEEE Visualization 91*, Oct 22-25, San Diego, California, p. 148-153.

Kenwright, D. N. and Mallinson, G. D., 1992, A 3-D Srteamline tracking algorithm using dual stream functions, *Proceedings of IEEE Visualization 92*, Oct 19-23, Boston, Massachusetts, p. 62-68.

Laramee, R. S., Hauser, H., Doleisch, H., Vrolijk, B., Post, F. H., and Weiskopf, D., 2004, The state of the art in flow visualization: dense and texture-based techniques, *Computer Graphics Forum*, v. 23, no. 2.

Dunham et al., 2003, A supershear transition mechanism for cracks**,** *Science* 299, p. 1557-1559

Bresenham, J., 1965, Algorithm for computer control of a digital plotter, *IBM Systems Journal,* v. 4, no. 1, p. 25-30.

Perlin, K., 1985, An image synthesizer, *Computer Graphics,* v.19, no. 3, p. 287-296.

Freeman, W., Adelson, E. and Heeger, D., 1991, Motion without movement, *Computer Graphics,* v. 25, no. 4, p. 27-30.

Chui, K., 1992, An introduction to wavelets. Academic Press, Inc., p. 49-60.

Devir, Z., Tridensky, L., Dynamic range compression while improving the visual quality of color images, Retrieved August 2005 from www.cs.technion.ac.il/~gip/projects/s2002/Zvi_Devir/files/report.pdf

Dunham, E. M., and Archuleta, R. J., 2004, Near-source ground motion in earthquakes: the influence of rupture speed on the resolution of fault processes, in preparation.

Tooby, P., TeraShake: SDSC simulates the big one, Retrieved August 20th from www.ttivanguard.com/sfreconn/terashake.pdf

Liu, Z., and Moorhead II, R. J., 2005, Accelerated unsteady flow line integral convolution, *IEEE Transactions on Visualization and Computer Graphics*, v. 11, no. 2, p. 113-125.

Stalling, D., and Hege, H., 1995, Fast and resolution independent line integral convolution, *Proceedings of ACM SigGraph* 95, Aug 6-11, Los Angeles, California, p. 249-256.

Helgeland, A., and Andreassen, O., 2004, Visualization of vector fields using seed LIC and volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 10(6), p. 673-682.

Day, S.M., 1982, three-dimensional simulation of spontaneous rupture: the effect of nonuniform prestress, *Bull. Seism. Soc. Am.,* v. 72, p. 1881-1902.

Harris, R. A., and Day, S. M., 1997, Effects of a low-velocity zone on a dynamic rupture, *Bull. Seism. Soc. Am*., v. 87, p. 1267-1280.

ABSTRACT OF THE THESIS

Matlab Toolbox for High Resolution Vector Field Visualization with Application in Improving the Understanding of Crack Propagation Mechanisms

by
Nima Bigdely Shamlo
Master of Science in Computational Science
San Diego State University, 2005

Traditional vector field visualization methods are unable to demonstrate details in high resolution. In many cases these details are valuable to understand the underlying mechanism. Crack propagation in earthquake rupture dynamics is one of these examples.

Linear Integral Convolution (LIC) is one of the most popular texture methods for vector field visualization. The idea behind LIC is to demonstrate vector directions by blurring a texture along field lines. Several variations of this method (Regular LIC, Fast LIC, and Animated LIC) have been implemented in different computer languages (Delphi, Matlab and C). Evaluating the performance of these implementations, Fast LIC outperforms Regular LIC in large image and kernel lengths since its computation time is linearly dependent on the number of input pixels. A Matlab toolbox is developed that helps geophysicists spot intricate details of a vector field. This toolbox, called VFV, uses Fast LIC method to produce texture and employs histogram determination methods to increase the contrast of output images. Matlab VFV toolbox provides a complete set of commands to generate LIC images and animations. The tools are arranged in a graphical user interface (GUI) for exploring vector fields with zooming capabilities. Demonstrated on several geophysics problems, these tools show the potential to provide new insights on complex mechanisms involved in crack propagation.