Dynamic Line Integral Convolution for Visualizing Electromagnetic Phenomena

by

Andreas Sundquist

Submitted to the Department of Electrical Engineering and Computer Science

and to the Department of Physics

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Physics

Bachelor of Science in Electrical Engineering and Computer Science

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 23, 2001

Author _____

Department of Electrical Engineering and Computer Science, and Department of Physics
May 23, 2001

Certified by _____

Professor John W. Belcher
Thesis Supervisor, Department of Physics

Accepted by _____

Professor Arthur C. Smith
Chairman, Department Committee on Graduate Theses, Department of E.E.C.S.

Accepted by _____

Professor David E. Pritchard
Senior Thesis Coordinator, Department of Physics

Dynamic Line Integral Convolution for Visualizing Electromagnetic Phenomena

by

Andreas Sundquist

Submitted to the

Department of Electrical Engineering and Computer Science

and the Department of Physics

May 23, 2001

In Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Physics

Bachelor of Science in Electrical Engineering and Computer Science

and Master of Engineering in Electrical Engineering and Computer Science

Abstract

Vector field visualization is a useful tool in science and engineering, giving us a powerful way of understanding the structure and evolution of the field. A fairly recent technique called *Line Integral Convolution* (*LIC*) has improved the level of detail that can be visualized by convolving a random input texture along the streamlines in the vector field. This thesis extends the technique to time-varying vector fields, where the motion of the field lines is specified explicitly via another vector field. The sequence of images generated is temporally coherent, clearly showing the evolution of the fields over time, while at the same time each individual image retains the characteristics of the LIC technique. This thesis describes the new technique, entitled *Dynamic Line Integral Convolution*, and explores its application to experiments in electromagnetism.

Thesis Supervisor: John W. Belcher

Title: Professor and Class of 1960 Faculty Fellow, MIT Department of Physics

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

In science and engineering, vector fields often play a central role in describing things such as electromagnetism or fluid flow. For the very simplest fields, it is usually possible to analyze its structure on a global scale, and to create a mental image of what it looks like. However, we are often presented with complicated, time-varying fields whose properties are not intuitive or easily analyzable. With the aid of modern technology, we can use computer graphics to create illustrative representations of the vector field, giving us tremendous insight into its structure.

In a fairly recent paper by Cabral and Leedom [3], a general, high-quality method was introduced for visualizing vector fields, called *Line Integral Convolution* (*LIC*). A number of papers published thereafter have extended this technique in different ways. One important generalization is to be able to visualize a time-varying vector field, maintaining a natural correspondence between successive frames of the visualization. None of the solutions presented by other researchers can satisfactorily do so for time-varying electromagnetic fields.

This thesis describes a new method called *Dynamic Line Integral Convolution* (*DLIC*) developed in conjunction with the TEAL / Studio Physics project that can effectively visualize dynamic electromagnetic fields. It extends LIC by allowing the vector field to vary over time and using a second vector field to describe how the field line structures vary over time. As a result, we have been able to create several animation sequences of experiments in electromagnetism that exhibit all the wonderful detail of the LIC method as well as the evolution of the field over time.

1.1 DYNAMIC VECTOR FIELD VISUALIZATION PROBLEM

Before choosing a method for visualizing vector fields, we need to decide what features of the vector field we would like the image to represent. For example, in some problems only the magnitude of the vector field is important. In this case, a natural representation is to modulate the color/intensity of a pixel on the screen by the magnitude of the vector field at each point. Of course, since a computer display is only a two-dimensional device, this is much more difficult for vector fields of three dimensions of higher.

In other problems, we might instead be interested in the direction of the vector field at each point. Again, we could conceivably modulate the pixel's color/intensity by the direction of the field. This could even be done for three-dimensional fields, since a pixel's color is a three-

dimensional quantity. Unfortunately, color is not a natural way of representing direction. A better solution is to use local directional indicators throughout the vector field. For example, we could establish an ordered grid in a two-dimensional vector field and draw tiny arrows along the direction of the field at those points. The maximum resolution of the visualization is limited, however, because the grid must be coarse enough to provide spacing between the arrows.

Another objective of the visualization may be to observe streamlines (integral lines) in the vector field. One possible solution is to use the above technique to denote the direction at each grid point, and let the observer mentally integrate curves in the vector field. Unfortunately, what the observer perceives is often inaccurate or incomplete, and it is difficult to analyze the features in the field. Instead, we could have the computer automatically integrate along the streamlines and display them. However, this gives rise to the question of *which* streamlines should be displayed. There are methods for automatically selecting a reasonable distribution of streamlines, but they cannot guarantee that all the important features will be represented. Fortunately, texture-based methods such as LIC are straightforward and robust alternatives that faithfully represents the streamlines with as much detail as possible.

Finally, extending the visualization to dynamic fields creates a new set of challenges. In some applications, it might suffice to render each frame independently via one of the previous techniques, in which case the solution is straightforward. However, other problems, including electromagnetism, have complex inter-frame dependencies since we expect to be able to understand how field lines evolve over time. Thus, one difficulty is how to describe the correspondence between vector field points or streamlines over time in the first place. Once we have defined their motion, we face the challenge of producing an animation sequence that depicts this evolution in an intuitive fashion. Another problem is maintaining a consistent level of detail throughout the image, even as streamlines are compressed and expanded. LIC by itself cannot be used in general for time-varying vector fields, since it does not use any information about how the field lines move, nor does it attempt to maintain any sort of inter-frame coherence. The DLIC method introduced here offers a solution to these challenges.

1.2 THESIS OVERVIEW

This thesis is organized into four major sections. Chapter 2 introduces the basic concepts that are needed to discuss vector fields, streamlines, and some of the computational methods used to work with them. Chapter 3 is an overview of related work, discussing the Line Integral

Convolution technique and some of its important improvements, as well as other attempts to extend the method to dynamic vector fields. Chapter 4 describes the new Dynamic Line Integral Convolution algorithm and some of the details of our implementation. Finally, chapter 5 illustrates the application of DLIC to two experiments in electromagnetism.

2 BASIC CONCEPTS

In this chapter, some of the fundamental ideas used in vector field visualization will be introduced, along with some practical computational techniques.

2.1 NOTATION

2.1.1 VECTOR FIELDS

We define a *vector field* $\boldsymbol{f}$ as a function mapping a Euclidean domain $\boldsymbol{D} \subset \boldsymbol{R}^n$ to another space $\boldsymbol{R}^n$ of the same dimension:

$$\boldsymbol{f}(\boldsymbol{x} \in \boldsymbol{D}) = \boldsymbol{y} \in \boldsymbol{R}^n \qquad (2\text{-}1)$$

For our purposes, we only use two- and three-dimensional vector spaces, since higher dimensions are not natural to visualize.

For time-dependent vector fields, we write $\boldsymbol{f}$ as a function mapping $\boldsymbol{D} \times \boldsymbol{T}$ to $\boldsymbol{R}^n$, where $\boldsymbol{T}$ is some scalar time interval:

$$\boldsymbol{f}(\boldsymbol{x} \in \boldsymbol{D}, t \in \boldsymbol{T}) = \boldsymbol{y} \in \boldsymbol{R}^n \qquad (2\text{-}2)$$

2.1.2 STREAMLINES AND PATH LINES

A *streamline* $\boldsymbol{\sigma}$ is defined as an integral curve in a stationary vector field:

$$\frac{d}{du}\boldsymbol{\sigma}(u) = \boldsymbol{f}(\boldsymbol{\sigma}), \quad \boldsymbol{\sigma}(u_0) = \boldsymbol{\sigma}_0 \qquad (2\text{-}3)$$

In this case, the magnitude of the vector field determines how $\boldsymbol{\sigma}$ is parameterized. We can reparameterize the curve by arc length s to obtain $\hat{\boldsymbol{\sigma}}$:

$$ds = \|\boldsymbol{f}\| du \quad \Rightarrow \quad \frac{d}{ds}\hat{\boldsymbol{\sigma}}(s) = \frac{d\boldsymbol{\sigma}}{du}\frac{du}{ds} = \frac{\boldsymbol{f}(\hat{\boldsymbol{\sigma}})}{\|\boldsymbol{f}\|} = \hat{\boldsymbol{f}}(\hat{\boldsymbol{\sigma}}) \qquad (2\text{-}4)$$

Care must be taken, as this parameterization breaks down where the vector field vanishes. The notation $\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(s)$ will often be used to indicate the streamline for which $\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(0) = \boldsymbol{x}$.

For time-dependent vector fields, a streamline is similarly defined at a constant time:

$$\frac{d}{du}\boldsymbol{\sigma}(u) = \boldsymbol{f}(\boldsymbol{\sigma}, t_0), \quad \boldsymbol{\sigma}(u_0) = \boldsymbol{\sigma}_0 \qquad (2\text{-}5)$$

In chapter 4, to describe such a streamline, the following notation will be used instead:

$$\frac{d}{du}\boldsymbol{\sigma}_{t_0}^{u}(\boldsymbol{\sigma}_0) = \boldsymbol{f}(\boldsymbol{\sigma}, t_0), \quad \boldsymbol{\sigma}_{t_0}^{0}(\boldsymbol{\sigma}_0) = \boldsymbol{\sigma}_0, \tag{2-6}$$

where $\boldsymbol{\sigma}_{t_0}^{u}(\boldsymbol{\sigma}_0)$ indicates the streamline seeded at $\boldsymbol{\sigma}_0$ when $u = 0$ at the constant time t_0.

Note that this is different from a *path line*, which is an integral path taken over time:

$$\frac{d}{dt}\boldsymbol{\lambda}(t) = \boldsymbol{f}(\boldsymbol{\lambda}, t), \quad \boldsymbol{\lambda}(t_0) = \boldsymbol{\lambda}_0 \tag{2-7}$$

Again in chapter 4, an alternative notation will be used for such an integral path:

$$\frac{d}{dt}\boldsymbol{\lambda}_{t_0}^{t}(\boldsymbol{\lambda}_0) = \boldsymbol{f}(\boldsymbol{\lambda}, t_0 + t), \quad \boldsymbol{\lambda}_{t_0}^{0}(\boldsymbol{\lambda}_0) = \boldsymbol{\lambda}_0, \tag{2-8}$$

where $\boldsymbol{\lambda}_{t_0}^{t}(\boldsymbol{\lambda}_0)$ is the path line seeded at $\boldsymbol{\lambda}_0$ at time t_0 when $t = 0$.

2.2 NUMERICAL METHODS

2.2.1 VECTOR FIELD INTERPOLATION

In computer programs, vector fields are often represented as samples over a discrete grid, either as a result of a grid-based field computation method or as a way to speed up vector field evaluations. In order to reconstruct the continuous field function, a number of different interpolation methods can be employed. A general technique is to convolve the discrete samples with a continuous kernel. For example, on a two-dimensional integer grid $\boldsymbol{g}\{i, j\}$ with a kernel $\chi(x, y)$ we define the vector field $\boldsymbol{f}$ by:

$$\boldsymbol{f}(x, y) = \iint \boldsymbol{g}\{\lfloor x + x' \rfloor, \lfloor y + y' \rfloor\}\chi(x', y')dx'dy' \tag{2-9}$$

For our purposes, bilinear interpolation is sufficient because the vector fields are generally smooth. The bilinear convolution kernel in two dimensions is:

$$\chi(x, y) = \begin{cases} 1, & 0 < x < 1 \text{ and } 0 < y < 1 \\ 0, & \text{otherwise} \end{cases} \tag{2-10}$$

The resulting expression for the vector field simplifies to

$$\boldsymbol{f}(x, y) = (1 - y_f)((1 - x_f)\boldsymbol{g}_{0,0} + x_f \boldsymbol{g}_{1,0}) + y_f((1 - x_f)\boldsymbol{g}_{1,0} + x_f \boldsymbol{g}_{1,1}), \text{ where} \tag{2-11}$$

$$x_f = x - \lfloor x \rfloor, \quad y_f = y - \lfloor y \rfloor, \quad \boldsymbol{g}_{m,n} = \boldsymbol{g}\{\lfloor x \rfloor + m, \lfloor y \rfloor + n\} \tag{2-12}$$

This expression can be evaluated very quickly.

2.2.2 HERMITE CURVE INTERPOLATION

Another place that interpolative methods are useful is for reconstructing a continuous streamline from discrete samples. Suppose we have a sequence of samples parameterized by arc length:

$$\hat{\boldsymbol{\sigma}}(s_0),\ \hat{\boldsymbol{\sigma}}(s_1),\ \hat{\boldsymbol{\sigma}}(s_2),\ ...,\ \hat{\boldsymbol{\sigma}}(s_n) \tag{2-13}$$

In addition to the position, we know the tangent of the curve at those parameters from the vector field:

$$\frac{d}{ds}\hat{\boldsymbol{\sigma}}(s_k) = \hat{\boldsymbol{f}}(\hat{\boldsymbol{\sigma}}(s_k)) \tag{2-14}$$

Between two adjacent sample points k and $k+1$, we can fit a cubic Hermite curve by using the two positions and the two tangents:

$$\tilde{\boldsymbol{\sigma}}_k(s) = \begin{bmatrix} s'^3 & s'^2 & s' & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{\sigma}}(s_k) \\ \hat{\boldsymbol{\sigma}}(s_{k+1}) \\ h\hat{\boldsymbol{f}}(\hat{\boldsymbol{\sigma}}(s_k)) \\ h\hat{\boldsymbol{f}}(\hat{\boldsymbol{\sigma}}(s_{k+1})) \end{bmatrix} \tag{2-15}$$

$$s' = \frac{s - s_k}{h}, \quad h = s_{k+1} - s_k$$

This curve has the desired properties at the endpoints

$$\begin{aligned} \tilde{\boldsymbol{\sigma}}_k(s_k) &= \hat{\boldsymbol{\sigma}}(s_k), & \tilde{\boldsymbol{\sigma}}_k(s_{k+1}) &= \hat{\boldsymbol{\sigma}}(s_{k+1}), \\ \frac{d}{ds}\tilde{\boldsymbol{\sigma}}_k(s_k) &= \hat{\boldsymbol{f}}(\hat{\boldsymbol{\sigma}}(s_k)), & \frac{d}{ds}\tilde{\boldsymbol{\sigma}}_k(s_{k+1}) &= \hat{\boldsymbol{f}}(\hat{\boldsymbol{\sigma}}(s_{k+1})) \end{aligned} \tag{2-16}$$

and smoothly interpolates between the sample points as we'd expect, illustrated in *Figure 2-1*.

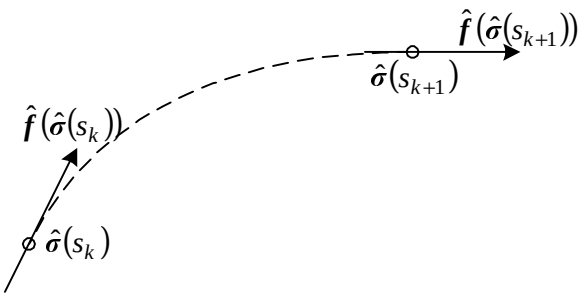

Figure 2-1: Cubic Hermite curve interpolation

Unfortunately, even though the tangents at the endpoints are unit magnitude, in general this will not be true in between. However, as long as the streamline does not vary excessively between two successive sample points, Hermite curves are a good approximation. In addition, because a Hermite curve is a cubic polynomial, it can be evaluated at equi-spaced points along *s* very quickly using forward differences. First, the initial values of four variables are computed:

$$\tilde{\boldsymbol{\sigma}}(s) = \boldsymbol{a}_3 s^3 + \boldsymbol{a}_2 s^2 + \boldsymbol{a}_1 s + \boldsymbol{a}_0 = ((\boldsymbol{a}_3 s + \boldsymbol{a}_2)s + \boldsymbol{a}_1)s + \boldsymbol{a}_0$$
$$\boldsymbol{d}_1(s) = \tilde{\boldsymbol{\sigma}}(s) - \tilde{\boldsymbol{\sigma}}(s - \Delta s), \quad \boldsymbol{d}_2(s) = \tilde{\boldsymbol{\sigma}}(s) - 2\tilde{\boldsymbol{\sigma}}(s - \Delta s) + \tilde{\boldsymbol{\sigma}}(s - 2\Delta s), \quad \Delta \boldsymbol{d} = 6(\Delta s)^3 \boldsymbol{a}_3$$

Then, we can compute the next sample point in succession with only three additions:

$$\begin{aligned} \tilde{\boldsymbol{\sigma}}(s + \Delta s) &= \tilde{\boldsymbol{\sigma}}(s) + \boldsymbol{d}_1(s + \Delta s) \\ \boldsymbol{d}_1(s + \Delta s) &= \boldsymbol{d}_1(s) + \boldsymbol{d}_2(s + \Delta s) \\ \boldsymbol{d}_2(s + \Delta s) &= \boldsymbol{d}_2(s) + \Delta \boldsymbol{d} \end{aligned} \qquad \textit{(2-17)}$$

2.2.3 INTEGRATING FIRST ORDER ODES

In scientific computing, one of the most basic types of problems is integrating first order ordinary differential equations. Given the differential equation and initial condition

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t), \quad \boldsymbol{x}(t_0) = \boldsymbol{x}_0, \qquad \textit{(2-18)}$$

how do we compute the value of $\boldsymbol{x}(t_1)$? If the vector field $\boldsymbol{f}$ is continuous and satisfies the Lipschitz condition, then a unique solution exists. Often, this is not the case, and care must be taken to understand the consequences of picking one particular solution over another.

Most methods for evolving the system by the required time step $t_1 - t_0$ use one-step or multi-step techniques. The simplest possible (and also the least accurate) is the *Simple Euler* method. To evolve the system by a time step *h*, we use the rule:

$$\boldsymbol{x}(t + h) = \boldsymbol{x}(t) + h\boldsymbol{f}(\boldsymbol{x}(t), t) \qquad \textit{(2-19)}$$

A much more accurate and robust technique devised by Runge and Kutta is the standard *RK4* integrator, which uses four evaluations of $\boldsymbol{f}$ to estimate the next step:

$$
\begin{aligned}
\boldsymbol{x}(t+h) &= \boldsymbol{x}(t) + \frac{h}{6}[\boldsymbol{f}_a + 2\boldsymbol{f}_b + 2\boldsymbol{f}_c + \boldsymbol{f}_d] \\
\boldsymbol{f}_a &= \boldsymbol{f}(\boldsymbol{x}(t), t) \\
\boldsymbol{f}_b &= \boldsymbol{f}(\boldsymbol{x}(t) + h\boldsymbol{f}_a/2, t + h/2) \\
\boldsymbol{f}_c &= \boldsymbol{f}(\boldsymbol{x}(t) + h\boldsymbol{f}_b/2, t + h/2) \\
\boldsymbol{f}_d &= \boldsymbol{f}(\boldsymbol{x}(t) + h\boldsymbol{f}_c, t + h)
\end{aligned}
\qquad \textit{(2-20)}
$$

Although there are more sophisticated, higher-order methods for integrating ODEs, such as the Adams predictor-corrector methods, the added complexity does not provide significant gains for our application. Indeed, for the fourth-order RK4 method the error goes as $O(h^5)$, so to obtain the desired accuracy we can simply reduce the step size h.

If we have an implicit equation for the exact integral curve (for example, as a result of a potential function), then we can use it to automatically adjust the step size to keep the results of the integrator within the desired error tolerance. Or, we can more generally construct an error metric by comparing the results of two integrators of different order. As discussed in Stalling dissertation [19], Fehlberg came up with a technique to use the intermediate results of the RK4 integrator to construct a third-order result to be used as an error estimate:

$$
\begin{aligned}
\tilde{\boldsymbol{x}}(t+h) &= \boldsymbol{x}(t) + \frac{h}{6}[\boldsymbol{f}_a + 2\boldsymbol{f}_b + 2\boldsymbol{f}_c + \boldsymbol{f}(\boldsymbol{x}(t+h))] \\
\varepsilon &= \left\| \boldsymbol{x}(t+h) - \tilde{\boldsymbol{x}}(t+h) \right\| = \frac{h}{6} \left\| \boldsymbol{f}_d - \boldsymbol{f}(\boldsymbol{x}(t+h)) \right\|
\end{aligned}
\qquad \textit{(2-21)}
$$

Given an error tolerance ε_{tol}, and knowing the error goes as $O(h^5)$, we can estimate the largest step size h^* that will remain within the tolerance as

$$
h^* = \min\left(\sqrt[5]{\frac{\rho\varepsilon_{tol}}{\varepsilon}},\ h_{\max} \right), \qquad \textit{(2-22)}
$$

where $\rho < 1$ is a safety factor to compensate for higher-order errors. The step size needs to be limited to a maximum of $h_{\max}$ since the error can become arbitrarily small. When a step is taken whose error is too large, the step size h is adjusted to h^* and the step is recomputed. When the step taken falls within the error tolerance, the next step is computed with the new step size h^*. This way, the integrator's performance is maximized by taking the largest steps possible.

2.2.4 STREAMLINE INTEGRATION

When the adaptive error-controlled RK4 integrator is combined with a cubic Hermite polynomial interpolator, the result is a fairly robust and efficient method for producing samples spaced evenly along a streamline $\hat{\boldsymbol{\sigma}}$. To summarize the algorithm:

1. Starting at the initial point $\hat{\boldsymbol{\sigma}}(s_0)$, use the adaptive error-controlled RK4 on the unit-magnitude vector field to produce a sequence of sample points and their parameters:

$$\hat{\boldsymbol{\sigma}}(s_0), \hat{\boldsymbol{\sigma}}(s_1), \hat{\boldsymbol{\sigma}}(s_2), \hat{\boldsymbol{\sigma}}(s_3), \ldots$$

2. For each equi-spaced point $s_0 + n\Delta s$, let k_n be the index of the integrator sample point immediately preceding it (i.e. $s_{k_n} \le s_0 + n\Delta s < s_{k_n+1}$). Compute the sample point by fitting a cubic Hermite polynomial to the integrator points $\hat{\boldsymbol{\sigma}}(s_{k_n}), \hat{\boldsymbol{\sigma}}(s_{k_n+1})$ and the tangents $\hat{\boldsymbol{f}}(\hat{\boldsymbol{\sigma}}(s_{k_n})), \hat{\boldsymbol{f}}(\hat{\boldsymbol{\sigma}}(s_{k_n+1}))$. A contiguous subsequence of equi-spaced points between two adjacent integrator sample points should be computed quickly via forward differences. The result is another sequence:

$$\tilde{\boldsymbol{\sigma}}(s_0), \tilde{\boldsymbol{\sigma}}(s_0 + \Delta s), \tilde{\boldsymbol{\sigma}}(s_0 + 2\Delta s), \tilde{\boldsymbol{\sigma}}(s_0 + 3\Delta s), \ldots$$

With these more or less evenly spaced samples, we can approximate line integrals taken along streamlines. For example, the line integral over the scalar field $r : \boldsymbol{D} \to \boldsymbol{R}$ can be approximated as a discrete sum:

$$\int r(\hat{\boldsymbol{\sigma}}(s))ds \cong \Delta s \sum_n r(\tilde{\boldsymbol{\sigma}}(s_0 + n\Delta s)). \qquad (2\text{-}23)$$

A line integral convolution can be computed similarly:

$$[(r \circ \hat{\boldsymbol{\sigma}}) \otimes \kappa](s_0) = \int r(\hat{\boldsymbol{\sigma}}(s_0 + s))\kappa(s)ds \cong \Delta s \sum_n r(\tilde{\boldsymbol{\sigma}}(s_0 + n\Delta s))\tilde{\kappa}(n\Delta s), \qquad (2\text{-}24)$$

where $\tilde{\kappa}(s)$ is a discrete approximation of the convolution kernel $\kappa(s)$. Note that the kernel should have a limited extent in order for the summation to be feasible.

A streamline integrator is not complete if it does not deal with streamline abnormalities, however. For example, what is the correct behavior when the streamline leaves the vector field domain $\boldsymbol{D}$ or terminates in a sink? What should the integrator do if the vector field does not obey the Lipschitz condition, and there isn't a unique solution? These questions can only be answered by examining the particular problem at hand.

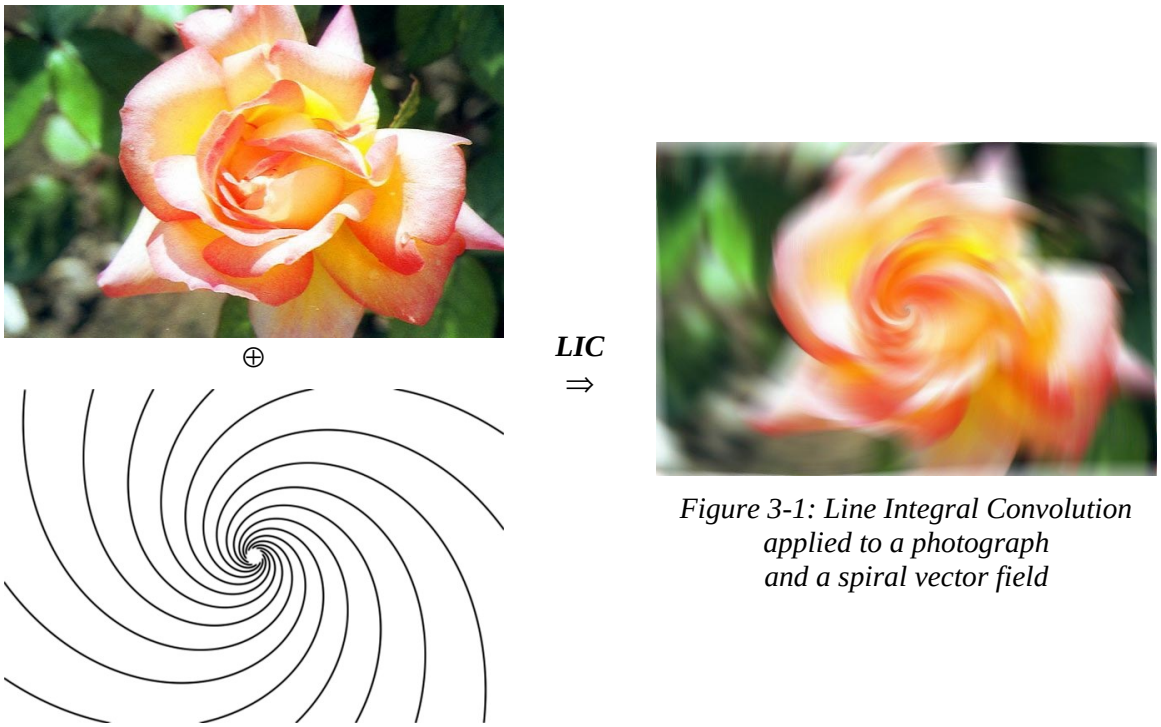

Figure 3-1: Line Integral Convolution applied to a photograph and a spiral vector field

3 PREVIOUS WORK

3.1 LINE INTEGRAL CONVOLUTION

In a seminal paper published by Cabral and Leedom entitled "Imaging Vector Fields Using Line Integral Convolution" [3], a new technique was introduced that allows us to represent vector fields with a level of detail as fine as the pixels on a computer screen, without causing confusion by being so densely packed. *Line Integral Convolution* (not to be confused with merely convolving via a line integral) can be viewed as an operation on an input texture and a vector field to produce an output image that represents the field.

Ideally, we can describe the result of this operation as:

$$I(\boldsymbol{x}) = [(T \circ \hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}) \otimes \kappa](0) = \int T(\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(s))\kappa(s)ds\,, \qquad (3\text{-}1)$$

where $I(\boldsymbol{x})$ is the output image intensity at the continuous location $\boldsymbol{x}$, $T(\boldsymbol{x})$ is the input texture intensity at $\boldsymbol{x}$, $\kappa(s)$ is the convolution kernel, and $\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(s)$ is the arc-length parameterized streamline such that $\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(0) = \boldsymbol{x}$. The above *Figure 3-1* was generated by applying LIC separately to each of the red, green, and blue color components of the color photograph, using a constant

convolution kernel of finite extent. We can see that the image has been "blended" along the direction of the spiral vector field.

One interesting effect of this operation is the correlation between output intensities along the same streamline, for example between $\boldsymbol{x}$ and $\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(\Delta s)$. The integral for the intensity at $\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(\Delta s)$ can be rewritten as:

$$\begin{aligned} I(\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(\Delta s)) &= \int T\left(\hat{\boldsymbol{\sigma}}_{\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(\Delta s)}(s)\right)\kappa(s)ds = \int T(\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(\Delta s+s))\kappa((\Delta s+s)-\Delta s)ds \\ &= \int T(\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(s'))\kappa(s'-\Delta s)ds' \end{aligned} \qquad \textit{(3-2)}$$

This identity uses the fact that $\hat{\boldsymbol{\sigma}}_{\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(\Delta s)}(s) = \hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(\Delta s+s)$. The resulting difference in intensity can therefore be expressed as:

$$I(\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(\Delta s)) - I(\boldsymbol{x}) = \int T(\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(s))[\kappa(s-\Delta s)-\kappa(s)]ds \qquad \textit{(3-3)}$$

If the difference between κ and itself shifted by Δs is small, then the output intensities will be very similar.

For a two-dimensional field, in order to compute the output intensity of a particular pixel $\boldsymbol{x}_0$, ideally we would convolve $I(\boldsymbol{x})$ with the spatial response curve $\chi(\boldsymbol{x})$ of a pixel:

$$O(\boldsymbol{x}_0) = \iint I(\boldsymbol{x}_0+\boldsymbol{x})\chi(\boldsymbol{x})d\boldsymbol{x} \qquad \textit{(3-4)}$$

Using something like a radially-symmetric Gaussian response curve would give high-quality results, but computing the full integral is too time-consuming. In practice, if the input texture $T(\boldsymbol{x})$ comes from pixels with a resolution at least as coarse as $O(\boldsymbol{x})$, then $I(\boldsymbol{x})$ will have frequency components that are only as high as those in $T(\boldsymbol{x})$. Therefore, we can get results that are almost as good by using a much simpler filter. For example, we could use a constant response curve

$$\chi(\boldsymbol{x}) = \begin{cases} \dfrac{1}{(\Delta x)^2}, & -\dfrac{\Delta x}{2} < x_1 < \dfrac{\Delta x}{2} \text{ and } -\dfrac{\Delta x}{2} < x_2 < \dfrac{\Delta x}{2} \\ 0, & \text{otherwise} \end{cases} \qquad \textit{(3-5)}$$

where Δx is the width and height of a square pixel. Usually, it is simplified even further by approximating the convolution integral as the sum of a few samples; often even a single sample will suffice.

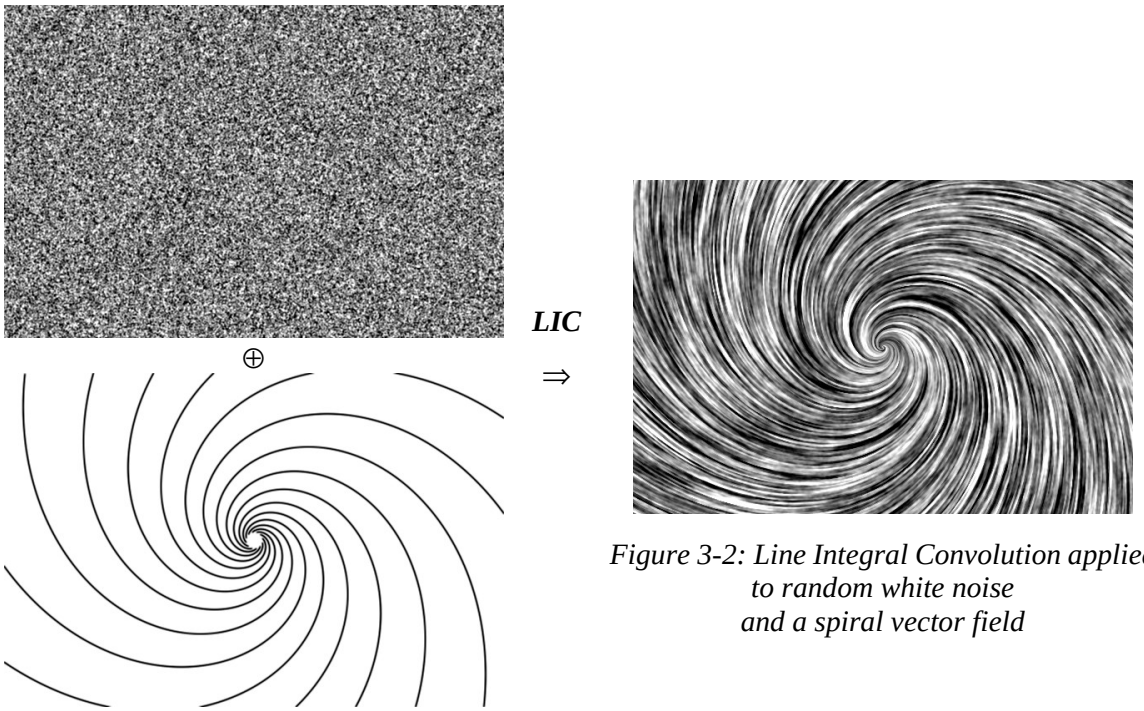

Figure 3-2: Line Integral Convolution applied to random white noise and a spiral vector field

Though the LIC technique can be viewed as an image operator, in order to visualize vector fields, random white noise is typically used as the input texture. This way, the net effect of LIC is that output intensities along a streamline tend to vary very slowly in intensity, while pixels in the perpendicular direction have no correlation. Thus, an observer can easily discern the direction of the vector field – an example can be seen in *Figure 3-2*. Although the above description of LIC has focused on two-dimensional vector fields, the method extends naturally to higher dimensions. In particular, three-dimensional fields can be successfully imaged, although it is more difficult to present the results on a two-dimensional computer display.

Images produced in this way have a strong resemblance to particle streak images produced by real-life flow experiments. The reason is because the input texture can be viewed as an instantaneous image of the particles at one point in time, and the consequent application of LIC spreads the particles along their streamlines via the convolution kernel $\kappa(s)$. The following discussion of the particle image-formation process is similar to the model described in Stalling's dissertation [19]. For one particle, given its initial position $\boldsymbol{p}_i(0)$, we can describe its advection path as:

$$\boldsymbol{p}_i(t) = \boldsymbol{\sigma}_{\boldsymbol{p}_i(0)}(t) \qquad (3\text{-}6)$$

Suppose the particle has an intrinsic intensity level a_i, and that over time its "exposure" on the output image is proportional to $\bar{\kappa}(t)$. Then, the resulting image due to that particle is:

$$I_i(\boldsymbol{x}) = \int a_i \delta(\boldsymbol{x} - \boldsymbol{p}_i(t))\bar{\kappa}(t)dt, \qquad (3\text{-}7)$$

where we represent the image of the particle via a delta-function. Taking the contributions of all the particles together, the output image is:

$$I(\boldsymbol{x}) = \sum_i I_i(\boldsymbol{x}) = \sum_i \int a_i \delta(\boldsymbol{p}_i(t) - \boldsymbol{x})\bar{\kappa}(t)dt = \sum_i \int a_i \delta(\boldsymbol{\sigma}_{\boldsymbol{p}_i(0)}(t) - \boldsymbol{x})\bar{\kappa}(t)dt \qquad (3\text{-}8)$$

Now, suppose all the particles at time $t = 0$ densely cover the domain $\boldsymbol{D}$. Then, we can turn the sum over the particles into an integral over $\boldsymbol{D}$:

$$I(\boldsymbol{x}) = \int_{\boldsymbol{D}}\int \delta(\boldsymbol{\sigma}_y(t) - \boldsymbol{x})a_y\bar{\kappa}(t)dt\,d\boldsymbol{y} = \int_D \int \delta(\boldsymbol{\sigma}_y(t) - \boldsymbol{x})a_y\bar{\kappa}(t)d\boldsymbol{y}\,dt \qquad (3\text{-}9)$$

where a_y is the intensity of the particle that is at $\boldsymbol{y}$ at time $t = 0$. Assuming the particle paths are unique, the delta function is non-zero when:

$$\boldsymbol{\sigma}_y(t) = \boldsymbol{x} \quad \Leftrightarrow \quad \boldsymbol{y} = \boldsymbol{\sigma}_{\boldsymbol{x}}(-t) \qquad (3\text{-}10)$$

In order to integrate out the delta function, we need to consider what happens as we vary the integration variable $\boldsymbol{y}$ near the point where the delta function is non-zero. If we vary $\boldsymbol{y}$ an amount ε along the direction of the streamline, then $\boldsymbol{\sigma}_y(t)$ will move by an amount

$$\varepsilon \cdot \left(\left.\frac{ds}{dt}\right|_0\right)^{-1}\left(\left.\frac{ds}{dt}\right|_t\right), \qquad (3\text{-}11)$$

where s is the arc-length along the streamline. Similarly, if we vary $\boldsymbol{y}$ in a direction perpendicular to the streamline, $\boldsymbol{\sigma}_y(t)$ will also move, scaled some relative expansion or contraction of the streamlines. This can be an extremely complicated effect, and so I will simply denote it by the symbol β. Thus, taking these variances into account, we can integrate out the delta function and rewrite the intensity of a point as:

$$I(\boldsymbol{x}) = \int a_y\bar{\kappa}(t)\left[\beta^{-1} \cdot \left(\left.\frac{ds}{dt}\right|_0\right)\left(\left.\frac{ds}{dt}\right|_t\right)^{-1}\right]dt, \qquad (3\text{-}12)$$

where **y** comes from *Equation 3-10*. Next, we define a_y as the input texture $T(\boldsymbol{y})$, giving us:

$$I(\boldsymbol{x}) = \int T(\boldsymbol{\sigma}_{\boldsymbol{x}}(-t))\,\overline{\kappa}(t)\left[\beta^{-1}\left(\left.\frac{ds}{dt}\right|_0\right)\left(\left.\frac{ds}{dt}\right|_t\right)^{-1}\right]dt \qquad (3\text{-}13)$$

If we parameterize the particle paths by distance instead of time, then we would have $t = s$ and $ds/dt = 1$, which would dramatically simplify the equation. Indeed, this is desirable if we are more interested in the direction of the vector field than the magnitude. Finally, although β may have a strong effect on the intensity, for visualization purposes it makes sense to set this to one. For example, a large β means that the streamlines have "spread apart", and that the particles have a overly small effect on the output intensity. I argue that particles should not be affected by how the streamlines expand and contract, but that their advection should simply represent the directions of the streamlines. Finally, reversing the direction of the integral, the result is the expression:

$$I(\boldsymbol{x}) = \int T(\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(s))\,\overline{\kappa}(-s)\,ds \qquad (3\text{-}14)$$

This is identical to the output intensity function for Line Integral Convolution, if $\kappa(s) = \overline{\kappa}(-s)$. Intuitively, the above particle-advection model can be seen as "scattering" the particles along their streamlines, while Line Integral Convolution "gathers" particles along the streamlines that would have contributed to each output point.

Typically, as discussed earlier, streamlines used in LIC are reparameterized by arc-length, which is important in order to control the degree of streamline correlation and maintain a consistent level of detail despite large variances in the vector field magnitude. Unfortunately, information about the vector field's magnitude is by this reparameterization. A simple method for reintroducing this information is color-coding the output intensities by the local field magnitude, as seen in *Figure 3-3.* Another technique due to Kiu and Banks [12] uses multiple input textures with different frequency extents, blending them according to the local vector magnitude. This results in images that have very coarse detail where the magnitude is small and fine detail where the magnitude is large (or vice versa).

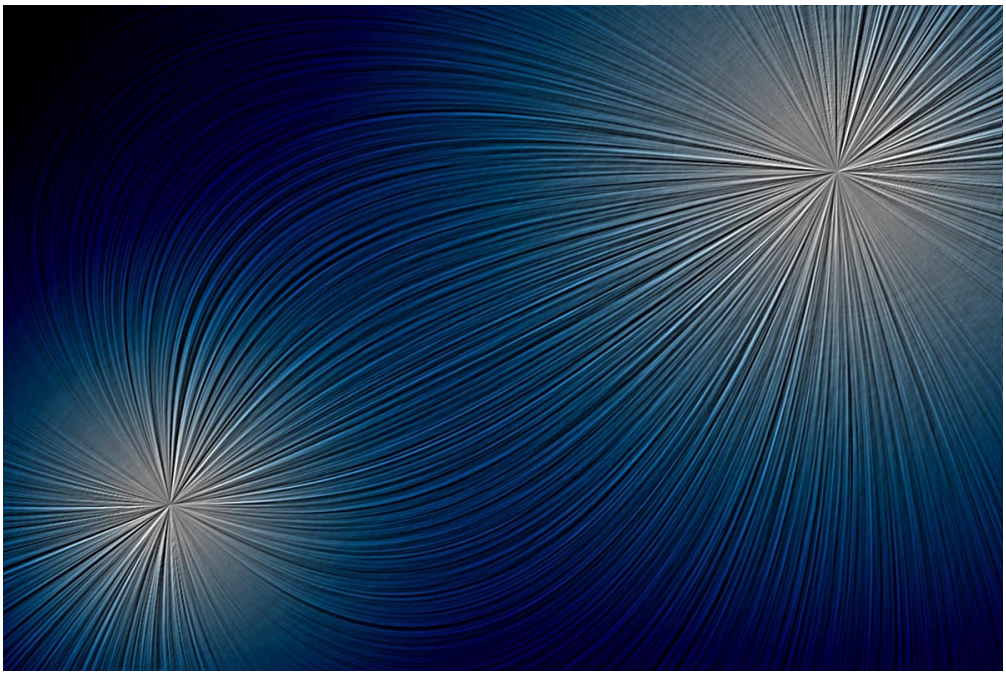

Figure 3-3: LIC rendering of two electric charges color coded by electric field intensity

3.2 EXTENSIONS

A number of papers published after the introduction of LIC have extended it in important ways. In the article "Fast and Resolution Independent Line Integral Convolution" by Stalling and Hege [18], a method was described to decouple the input and output resolutions, and also to accelerate the LIC computation by an order of magnitude for box-filter convolutions. We can make the output resolution independent of the input resolution simply by introducing a transformation $\boldsymbol{\Phi}$ from the output pixels $O(\boldsymbol{x})$ to the output intensity field $I(\boldsymbol{x})$:

$$O(\boldsymbol{x}_0) = \iint I(\boldsymbol{\Phi}(\boldsymbol{x}_0 + \boldsymbol{x}))\chi(\boldsymbol{x})d\boldsymbol{x} \qquad (3\text{-}15)$$

Indeed, we could also have separate coordinate systems for the input texture and the vector field. For example, if the input texture is made coarser than the vector field, then the resulting output image will be a coarser rendition of the vector field. In her paper "Visualizing Flow Over Curvilinear Grid Surface Using Line Integral Convolution" [6], Forsell extended it further, allowing LIC to work on more general surfaces.

A number of other methods have been developed whose results are similar to those achieved by the LIC method. For example, the concept of "splatting" was used to visualize vector fields by Crawfis and Max [4] and similarly by de Leeuw and van Wijk [14]. Another paper by de

Leeuw and van Liere, "Comparing LIC and Spot Noise"[13], describes how the two relate, while Verma, Kao, and Pang attempt to consolidate LIC with the idea of drawing streamlines in their article on a hybrid method called *Pseudo-LIC* [20]. Wegenkittl, Gröller and Purgathofer use an asymmetric LIC convolution kernel in order to provide additional information about the orientation of the vector field in their paper on *Oriented Line Integral Convolution* [21]. Surprisingly, a radically different diffusion method developed by Diewald, Preußer, and Rumpf [5] produces results very similar to LIC – in fact they claim it is more general than LIC. Also, there exist several methods for approximating LIC in order to speed it up substantially. For example, the document "Adaptive LIC Using a Curl Field and Stochastic LIC Approximation using OpenGL" by Bryan [2], takes advantage of hardware acceleration.

3.2.1 FAST LINE INTEGRAL CONVOLUTION

Returning to the paper by Stalling and Hege on *Fast Line Integral Convolution* (*FLIC*) [18], suppose that the convolution kernel $\kappa(s)$ takes on the form:

$$\kappa(s) = \begin{cases} \dfrac{1}{2S}, & -S < s < S \\ 0, & \text{otherwise} \end{cases} \qquad \textit{(3-16)}$$

Then, there is an interesting relationship between the output intensities of two points along a streamline separated by $\Delta s < 2S$:

$$\begin{aligned} I(\hat{\sigma}_{\boldsymbol{x}}(\Delta s)) - I(\boldsymbol{x}) &= \int T(\hat{\sigma}_{\boldsymbol{x}}(s))[\kappa(s-\Delta s) - \kappa(s)]ds \\ &= \frac{1}{2S}\left[\int_{S}^{S+\Delta s} T(\hat{\sigma}_{\boldsymbol{x}}(s))ds - \int_{-S}^{-S+\Delta s} T(\hat{\sigma}_{\boldsymbol{x}}(s))ds\right] \end{aligned} \qquad \textit{(3-17)}$$

Next, we turn the continuous integrals into discrete sums taken with a step size of Δs, using the discrete version of the convolution kernel:

$$\bar{\kappa}(n\Delta s) = \begin{cases} \dfrac{1}{2N\Delta s + 1}, & -N \le n \le N \\ 0, & \text{otherwise} \end{cases} \qquad \textit{(3-18)}$$

The relationship between adjacent output intensities along the streamline becomes much simpler:

$$\begin{aligned} I(\hat{\sigma}_{\boldsymbol{x}}(\Delta s)) - I(\boldsymbol{x}) &\cong \sum_{n} T(\hat{\sigma}_{\boldsymbol{x}}(n\Delta s))[\tilde{\kappa}((n-1)\Delta s) - \tilde{\kappa}(n\Delta s)] \\ &= \frac{1}{2N\Delta s + 1}[T(\hat{\sigma}_{\boldsymbol{x}}((N+1)\Delta s)) - T(\hat{\sigma}_{\boldsymbol{x}}(-N\Delta s))] \end{aligned} \qquad \textit{(3-19)}$$

Thus, we can incrementally compute the output intensities along a streamline in constant time per step. In fact, this method can be extended to speed up piece-wise polynomial convolution kernels as well [10] [19].

In the original LIC algorithm, the discrete convolution that produces the output pixels $O(\boldsymbol{x}_0)$ dictates which output intensity points $I(\boldsymbol{x})$ are computed. For the Fast LIC algorithm, in order to take advantage of the above relationship, we cannot insist on sampling the output intensities according to the discrete convolution filter $\chi(\boldsymbol{x})$. In fact, if we pick random starting points for the incremental streamline convolutions, the resulting set of output intensity samples are themselves pseudo-randomly distributed.

However, consider the convolution filter

$$\chi_{\boldsymbol{x}_0}(\boldsymbol{x}) = \frac{1}{N_{\boldsymbol{x}_0}} \sum_{i=1}^{N_{\boldsymbol{x}_0}} \delta\left(\boldsymbol{x} - \boldsymbol{z}_{\boldsymbol{x}_0}^{(i)}\right), \qquad \textit{(3-20)}$$

which is specific to each output pixel $\boldsymbol{x}_0$. The number of samples $N_{\boldsymbol{x}_0}$ is small, and the sample locations $\boldsymbol{z}_{\boldsymbol{x}_0}^{(i)}$ are scattered randomly around the pixel center. As is well known in computer graphics, as long as the frequency components in $I(\boldsymbol{x})$ are not higher than the frequencies that can be represented by $O(\boldsymbol{x}_0)$, a random sampling gives a reasonable approximation to a continuous convolution with a kernel proportional to the probability density of the sample points. Therefore, given a set of samples $I(\boldsymbol{x}_i)$, the uniform square convolution kernel (*Equation 3-5*) can be approximated by accumulating each sample in the appropriate output pixel, and then dividing each output pixel by the number of samples that hit it.

Thus, FLIC is typically implemented as follows:

1. Pick a random sample point $\boldsymbol{x}$ and compute $I(\boldsymbol{x})$ by a discrete streamline convolution. Accumulate this result in the appropriate output pixel.
2. Produce a sequence of new sample points $I(\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(k\Delta s))$ via the incremental method and accumulate those pixels. Continue along the streamline for some maximum distance $M\Delta s$.
3. If "most" of the output pixels have been hit at least as often as some specified minimum, then continue to the next step. Otherwise, go back to step 1.

4. For the remaining pixels that have not been hit often enough, sample the output intensity at those pixels and accumulate the result until every pixel has been hit the minimum amount.

5. Normalize the output pixels by dividing out the number of samples that have hit each pixel.

This algorithm produces results that are almost the same as those produced by LIC. For a streamline convolution of length $N\Delta s$, if we follow it for a distance $M\Delta s$, the ratio between the number of input textures sampled for LIC and for FLIC is roughly

$$\theta\left(\frac{N \cdot M}{N+M}\right) \approx \theta(N), \qquad (3\text{-}21)$$

where M is typically some factor larger than N. Since the cost of computing one streamline convolution is amortized over the entire length that the streamline is followed incrementally, FLIC achieves an order of magnitude speed-up.

Note the one downside of FLIC is that its results are somewhat dependent on how the streamlines are sampled. Two different random samplings will produce variations in the output images that are distinguishable upon careful examination. Thus, if the same vector field (or one that is similar) is to be visualized more than once using FLIC, care must be taken to preserve the order of the sampling.

3.2.2 DYNAMIC FIELD VISUALIZATION

There have been a number of articles describing methods for animating vector field images in the style of LIC. Most of them fall into one of two categories: depicting cyclic motion along streamlines for static vector fields, or simulating fluid motion along the streamlines of time-dependent flow fields.

The original paper by Cabral and Leedom on LIC [3] described a method for animating the field lines in the direction of the vector field by varying the convolution kernel over time. This method is based on a technique first described by Freeman, Adelson, and Heeger in "Motion Without Movement" [8]. By shifting a pseudo-periodic kernel in space over time, the output image appears to move along the direction of the kernel translation, in this case along the field lines. Jobard and Lefer reduced the time it takes to render an animation sequence by precomputing a "motion map", from which all the frames can be quickly extracted [11]. Forssell and Cohen improved LIC for variable-speed animation on curvilinear grids [7].

Other research has been done that allows the vector field itself to vary over time, but the application has almost always been fluid-flow models. In the same paper by Forssell and Cohen [7], they describe a modification to LIC whereby *streaklines* instead of streamlines are followed, where streaklines are defined as the set of points that pass through a seed point when advected through time. Unfortunately, because convolutions are performed over space as well as time, images exhibit behavior that looks "ahead" in time, which is unphysical and can be misleading. Shen and Kao modify this algorithm by feeding forward advected streakline paths and the results of the convolution instead of using future streakline immediately in their two publications on *Unsteady Flow LIC* [16] [17].

Evidently, vector field visualization techniques are most often used in flow problems, where the field indicates the direction of flow. Thus, the very same vector field that is being visualized also indicates how it should "flow" over time. In electromagnetism, this is no longer the case. We are visualizing electric or magnetic fields, but their "motion" is not generally along the field lines – in fact, motion along streamlines has no meaning. Visualizing electromagnetic fields presents us with a more general problem than that of visualizing fluid flow, since the vector field and the "motion" of the vector field lines themselves are different.

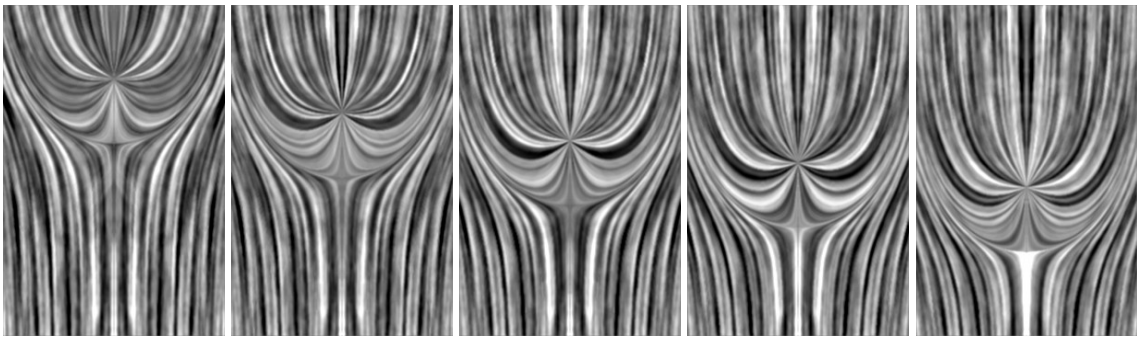

Figure 4-1: DLIC animation of a charge moving through a constant electric field

4 DYNAMIC LINE INTEGRAL CONVOLUTION

4.1 PROBLEM FORMULATION

Given two time-dependent vector fields $\boldsymbol{f}, \boldsymbol{d} : \boldsymbol{D} \times \boldsymbol{T} \rightarrow \boldsymbol{R}^n$, where $\boldsymbol{f}$ is the vector field we would like to visualize, and $\boldsymbol{d}$ describes how its field lines evolve over time, how do we generate a sequence of images that represents this information effectively? The field $\boldsymbol{f}$ is the same as before, except that it is allowed to vary over time, while the new field $\boldsymbol{d}$ is the instantaneous velocity of the motion of every streamline point in $\boldsymbol{f}$. The sequence of images must intuitively depict the streamlines in $\boldsymbol{f}$ evolving over time as indicated by $\boldsymbol{d}$. Each individual image must by itself be an accurate and detailed representation of the vector field $\boldsymbol{f}$ at a particular time. When viewed as a sequence, it should be clear how its streamlines evolve over time. *Figure 4-2* below is an example of the two fields for an electric charge moving against a uniform electric field.

At this point, the notation $\hat{\boldsymbol{\sigma}}_t^s(\boldsymbol{x})$ will be used to specify streamlines in $\boldsymbol{f}$ at constant time t

$\boldsymbol{f}$ vector field: the background vertical field lines "bend" around the electric charge

$\boldsymbol{d}$ vector field: as the charge moves down, the field lines are "pushed" apart horizontally, and near the charge move down along with it

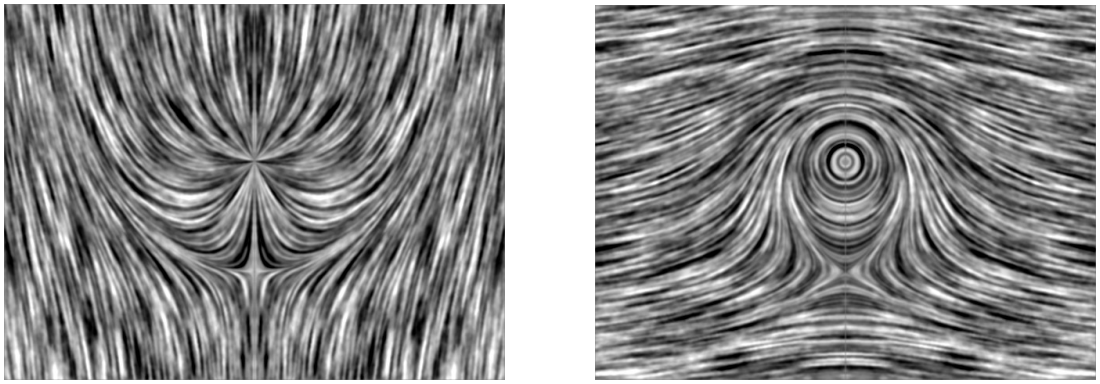

Figure 4-2: $\boldsymbol{f}$ and $\boldsymbol{d}$ fields of a charge moving against a uniform electric field

seeded at $\boldsymbol{x}$ and parameterized by arc-length s, while $\boldsymbol{\lambda}_t^{\Delta t}(\boldsymbol{x})$ will denote path lines in $\boldsymbol{d}$ starting at $\boldsymbol{x}$ at time t, evolved by a time delta Δt. Assuming the streamlines are all unique and can be labeled by some parameter $\boldsymbol{\alpha}$, every point must move such that

$$\boldsymbol{x} = \hat{\boldsymbol{\sigma}}_t^s(\boldsymbol{\alpha}) \quad \Rightarrow \quad \boldsymbol{x} + \boldsymbol{d}(\boldsymbol{x}, t)dt = \hat{\boldsymbol{\sigma}}_{t+dt}^{s'}(\boldsymbol{\alpha}), \tag{4-1}$$

for some parameters s and s' and an infinitesimal time delta dt. In other words, a point $\boldsymbol{x}$ must move with a velocity specified by $\boldsymbol{d}$ to another point on the same streamline at a different time.

It may seem that $\boldsymbol{d}$ and $\boldsymbol{f}$ encode redundant information, or that the above condition is excessively constrained. Indeed, for an n-dimensional domain $\boldsymbol{D} \subset \boldsymbol{R}^n$, we could label the streamlines by an $(n-1)$-dimensional parameter $\boldsymbol{\alpha}$, and thus a function $\boldsymbol{d} : \boldsymbol{R}^{n-1} \times \boldsymbol{T} \to \boldsymbol{R}^{n-1}$ could describe the evolution of the streamlines over time. However, not only is this representation very difficult to work with, it does not specify how different points on a particular streamline evolve along the streamline. Using two fields $\boldsymbol{f}$ and $\boldsymbol{d}$ is a straightforward way of representing all the information.

Note that fluid flow visualization becomes a subclass of this problem, where we take $\boldsymbol{d} = \boldsymbol{f}$. Since the streamlines visualized in $\boldsymbol{f}$ depict the direction of flow, it can likewise be used to describe how particles move along streamlines.

4.2 SOLUTION

The solution presented in this thesis, called *Dynamic Line Integral Convolution*, allows us to visualize time-varying vector fields represented by the $\boldsymbol{f}$ and $\boldsymbol{d}$ fields described above. Although it makes use of the LIC algorithm to render the images, the method is actually more general in that other texture-based vector visualization techniques could be used as well. *Figure 4-1* is an example of DLIC, which shows a sequence of images of a charge moving against a uniform electric field.

4.2.1 ALGORITHM OVERVIEW

Conceptually, the technique employed by DLIC is fairly simple. Going back to the model of the particle image-formation process in *Equation 3-7*, the output intensity can be seen as a sum of contributions from a number of particles:

$$I(\boldsymbol{x}) = \sum_i \int \delta(\boldsymbol{x} - \boldsymbol{p}_i(s)) a_i \bar{\kappa}(s) ds \qquad (4\text{-}2)$$

Now, instead of interpreting the streamlines in $\boldsymbol{f}$ physically as the motion of particles over time, we think of it simply as an imaging effect. Instead, we allow the particles to advect over time in the $\boldsymbol{d}$ field:

$$\boldsymbol{p}_i(t) = \lambda_0^t(\boldsymbol{p}_i) \qquad (4\text{-}3)$$

At any particular time, we form streamlines that are seeded at the location of the particle at that time:

$$\boldsymbol{p}_i(s,t) = \hat{\boldsymbol{\sigma}}_t^s(\boldsymbol{p}_i(t)) = \hat{\boldsymbol{\sigma}}_t^s\left(\lambda_0^t(\boldsymbol{p}_i)\right) \qquad (4\text{-}4)$$

Thus, the image-formation process over time can now be described as:

$$I(\boldsymbol{x},t) = \sum_i \int \delta(\boldsymbol{x} - \boldsymbol{p}_i(s,t)) a_i \bar{\kappa}(s) ds \qquad (4\text{-}5)$$

This two-phase evolution of the particle position, first via $\boldsymbol{d}$ from time 0 to t, and second via $\boldsymbol{f}$ by a distance s along the streamline, is illustrated in *Figure 4-3*.

Once again, assuming the particles cover the domain $\boldsymbol{D}$ for all time, we can rewrite the sum as an integral and replace the intrinsic intensity a by a time-varying texture T:

$$I(\boldsymbol{x},t) = \int_{\boldsymbol{D}} \int \delta\left(\boldsymbol{x} - \hat{\boldsymbol{\sigma}}_t^s(\boldsymbol{y})\right) a_{\boldsymbol{y}}^t \bar{\kappa}(s) ds\, d\boldsymbol{y} \cong \int T\left(\hat{\boldsymbol{\sigma}}_t^s(\boldsymbol{x}), t\right) \kappa(s) ds\,, \qquad (4\text{-}6)$$

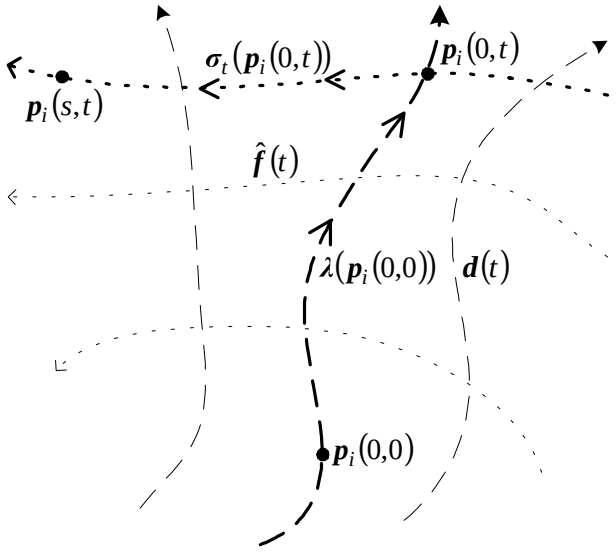

Figure 4-3: (s,t) parameterization of a particle's position via advection along $\boldsymbol{d}$ and $\boldsymbol{f}$ fields

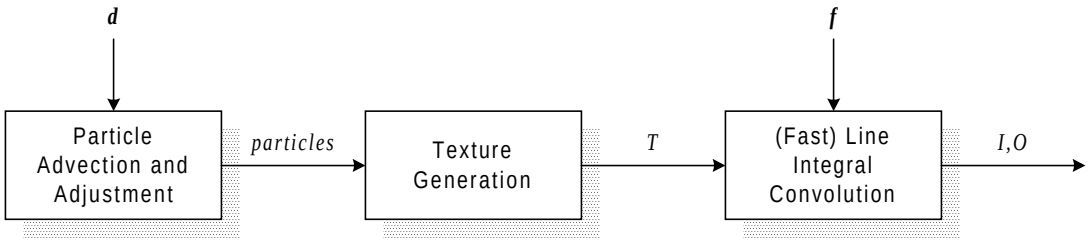

Figure 4-4: Organization of DLIC

where the index *i* has been replaced via the mapping

$$\boldsymbol{y} = \boldsymbol{p}_i(0,t) = \hat{\boldsymbol{\sigma}}_t^s(\boldsymbol{x}), \tag{4-7}$$

and the direction of the integral has been reversed.

The above derivation introduced a time-varying texture $T(\boldsymbol{x},t)$ that represents the intrinsic intensities of the particles as they move over time. Once we have such a texture at a particular time, the standard LIC algorithm will yield the output image for that time:

$$I(\boldsymbol{x},t) = \int T\left(\hat{\boldsymbol{\sigma}}_t^s(\boldsymbol{x}),t\right)\kappa(s)ds \tag{4-8}$$

Thus, the basic idea of the algorithm is to first evolve the particles via $\boldsymbol{d}$ to their locations at a particular time, then generate a texture representing those particles, and finally compute the output intensities $I(\boldsymbol{x},t)$ (and $O(\boldsymbol{x},t)$) using the texture and vector field $\boldsymbol{f}$. This process is illustrated in *Figure 4-4*, and the result is an image representing the field at that particular time that evolves naturally from the previous frame. After an image is rendered, the current time is incremented by a small amount *dt* and the process is repeated for the next frame of the sequence.

4.2.2 TEMPORAL CORRELATION

From a visualization standpoint, we would like the intensity of a point on a streamline at a certain time to remain highly correlated with itself as the point moves via $\boldsymbol{d}$ over time. Comparing the intensity of a point as it moves over a small time delta *dt*:

$$I\left(\lambda_t^{dt}(\boldsymbol{x}),t+dt\right) - I(\boldsymbol{x},t) = \int T\left(\hat{\boldsymbol{\sigma}}_{t+dt}^{s'}\left(\lambda_t^{dt}(\boldsymbol{x})\right),t+dt\right)\kappa(s')ds' - \int T\left(\hat{\boldsymbol{\sigma}}_t^s(\boldsymbol{x}),t\right)\kappa(s)ds \tag{4-9}$$

Now, since each point on the streamline at time *t* moves to another point on the streamline at time $t+dt$ via $\boldsymbol{d}$, we can come up with a function $u(s)=s'$ mapping these points, as illustrated in *Figure 4-5*. We can then rewrite the first integral as:

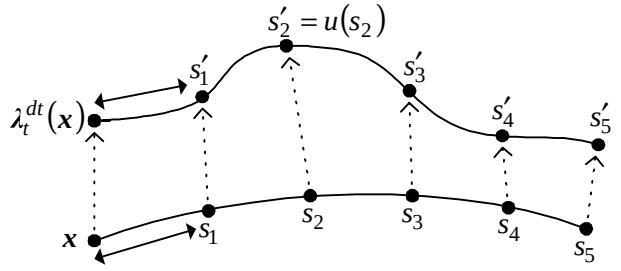

Figure 4-5: Temporal arc-length parameter mapping

$$\int T\left(\hat{\boldsymbol{\sigma}}_{t+dt}^{s'}\left(\lambda_t^{dt}(\boldsymbol{x})\right), t+dt\right)\kappa(s')ds' = \int T\left(\hat{\boldsymbol{\sigma}}_{t+dt}^{u(s)}\left(\lambda_t^{dt}(\boldsymbol{x})\right), t+dt\right)\kappa(u(s))\frac{du}{ds}ds \qquad (4\text{-}10)$$

Since the particles do not change their intrinsic intensity, we assert that the texture is the same at corresponding points in time:

$$T\left(\hat{\boldsymbol{\sigma}}_{t+dt}^{u(s)}\left(\lambda_t^{dt}(\boldsymbol{x})\right), t+dt\right) = T\left(\hat{\boldsymbol{\sigma}}_t^{s}(\boldsymbol{x}), t\right) \qquad (4\text{-}11)$$

This allows us to combine the two integrals:

$$I\left(\lambda_t^{dt}(\boldsymbol{x}), t+dt\right) - I(\boldsymbol{x}, t) = \int T\left(\hat{\boldsymbol{\sigma}}_t^{s}(\boldsymbol{x}), t\right)\left[\kappa(u)\frac{du}{ds} - \kappa(s)\right]ds \qquad (4\text{-}12)$$

Because the relationship between s and s' may be incredibly complex, this result may not seem very useful. However, consider what happens if we had a well-behaved relationship with a simple convolution kernel:

$$\kappa(s) = \begin{cases} \dfrac{1}{2S}, & -S < s < S \\ 0, & \text{otherwise} \end{cases}$$
$$u(s) = (1+\mu)s \quad \Rightarrow \quad \frac{du}{ds} = 1+\mu \qquad (4\text{-}13)$$

The mapping function I have chosen is linear, with the required condition $u(0) = 0$, though other functions that are close to $u(s) = s$ would yield similar results. The integral can then be simplified to:

$$I\left(\lambda_t^{dt}(\boldsymbol{x}), t+dt\right) - I(\boldsymbol{x}, t) =$$
$$\frac{\mu}{2S}\int_{-S/(1+\mu)}^{S/(1+\mu)} T\left(\hat{\boldsymbol{\sigma}}_t^{s}(\boldsymbol{x}), t\right)ds - \frac{1}{2S}\left[\int_{-S}^{-S/(1+\mu)} T\left(\hat{\boldsymbol{\sigma}}_t^{s}(\boldsymbol{x}), t\right)ds + \int_{S/(1+\mu)}^{S} T\left(\hat{\boldsymbol{\sigma}}_t^{s}(\boldsymbol{x}), t\right)ds\right] \qquad (4\text{-}14)$$

In other words, for a small "stretching" parameter μ, the first term of the difference is approximately μ times smaller than the intensity $I(\boldsymbol{x},t)$, and the remaining two terms are small corrections at the ends of the streamline.

Qualitatively, when $(du/ds-1)$ is small and $\kappa(s)$ is smooth, the difference in intensity of a particular point on a streamline as it evolves over time will also be small. Thus, we can recognize how points on a streamline move over time by their intensity correlation. In fact, since different points along a streamline are themselves close in intensity, the entire streamline is highly correlated as its points evolve over time.

4.2.3 TEXTURE GENERATION

If the output image has a pixel size of Δx, then by Nyquist's sampling theorem we know that the maximum frequency it can exhibit is $1/2\Delta x$. Since spatial convolutions are equivalent to multiplication in the frequency domain, and the output image $O(\boldsymbol{x}_0)$ is computed from the texture $T(\boldsymbol{x})$ via the convolution kernels $\kappa(s)$ and $\chi(\boldsymbol{x})$, the frequencies present in *O* are only as high as the lowest of the frequency extents of *T*, κ, and χ. Therefore, if the maximum frequency in *T* is $\omega < 1/2\Delta x$, then *O* will also have a frequency extent limited to ω. On the other hand, additional detail in the input texture beyond a frequency of $1/2\Delta x$ is clearly lost in the output. Therefore, we could reduce the frequency extent of *T* and still achieve similar results in the output *O*. This is wonderful from an implementation standpoint, since it would have been difficult to work with a function *T* that had an arbitrary frequency extent.

Returning to the model of discrete particles, we can realistically only work with a finite number. Unfortunately, a finite number of particles represented by delta functions will no longer fill the texture domain. Instead, we let the particles contribute to the texture via a shape distribution function $A(\boldsymbol{x})$. The output texture can then be written as a convolution over the sum of particle distribution functions scaled by their intrinsic intensity:

$$T(\boldsymbol{x},t) = \int\int \left[\sum_i a_i A(\boldsymbol{x}' + \boldsymbol{x} - \boldsymbol{p}_i(t)) \right] \chi(\boldsymbol{x}') d\boldsymbol{x}' \qquad (4\text{-}15)$$

This process is depicted in *Figure 4-6*. Since the particles are typically very small, their actual shape does not significantly affect the result of the output image. Similarly, the particulars of the convolution kernel are not critical. Thus, for simplicity, in my implementation I choose a square shape function and kernel:

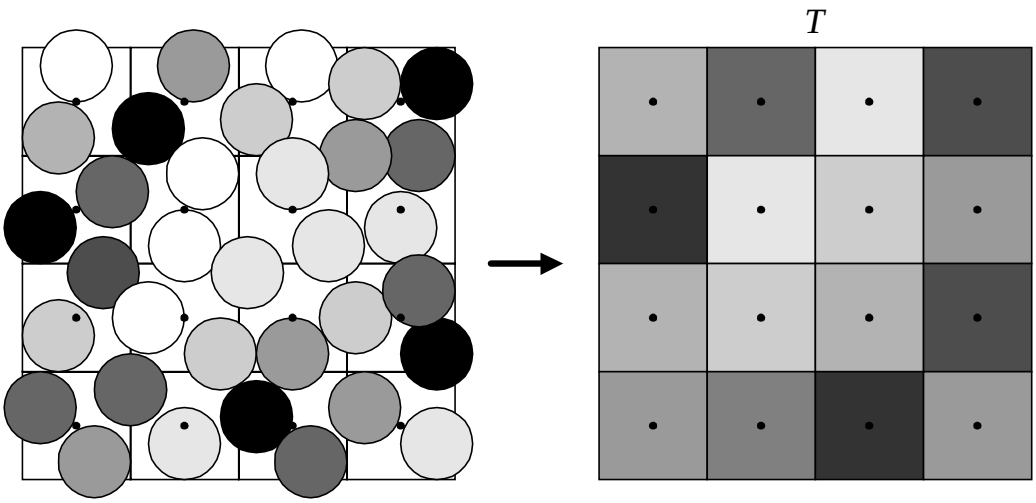

Figure 4-6: Texture generation from particles

$$A(\boldsymbol{x}) = \begin{cases} 1, & -\frac{\Delta x}{2} < x_1 < \frac{\Delta x}{2} \text{ and } -\frac{\Delta x}{2} < x_2 < \frac{\Delta x}{2} \\ 0, & \text{otherwise} \end{cases}$$
$$\chi(\boldsymbol{x}) = \begin{cases} \frac{1}{(\Delta x)^2}, & -\frac{\Delta x}{2} < x_1 < \frac{\Delta x}{2} \text{ and } -\frac{\Delta x}{2} < x_2 < \frac{\Delta x}{2} \\ 0, & \text{otherwise} \end{cases} \qquad (4\text{-}16)$$

The above integral then simplifies tremendously:

$$T(\boldsymbol{x},t) = \sum_i a_i u(x - p_{ix}(t)) u(y - p_{iy}(t))$$
$$u(x) = \begin{cases} 1 - |x|/\Delta x, & |x| < \Delta x \\ 0, & \text{otherwise} \end{cases} \qquad (4\text{-}17)$$

To compute $T(\boldsymbol{x})$ on the pixel grid, for each particle we compute the bilinear interpolation coefficients of its four surrounding pixels, scale by the particle's intrinsic intensity, and accumulate the results in the four pixels.

Although it is clear that the particle shapes need to mostly cover the texture domain D, it is not as obvious how densely packed they should be. If the particles are separated by a distance roughly equal to the pixel size Δx, then the maximum frequencies in the texture $T(\boldsymbol{x})$ would be on the order of $1/2\Delta x$, the frequency cut-off of the output image $O(\boldsymbol{x}_0)$. If the particle density was greater, then the frequencies in the texture would also be greater, but this could not improve the level of detail in the output image. On the other hand, if the particles were separated by much more than Δx, we would see a degradation in the output detail. Therefore, distributing particles on an ordered grid of pixel size Δx (perhaps jittered) gives the best results.

For vector field visualization purposes, the intrinsic intensities of the particles are usually chosen randomly. In this case, the above process is somewhat wasteful, since we could have simply chosen random values for $T(\boldsymbol{x})$ at each pixel. In fact, filling the texture randomly has the desirable property that we can control the distribution of random values in texture. In contrast, the distribution of texture pixel intensities computed from particles depends on the spatial distribution of the particles. Unfortunately, generating the texture from a set of particles is absolutely critical for the operation of DLIC.

There are ways, however, of adjusting the resulting input texture *T* so that its distribution is somewhat controlled. For example, to normalize the standard deviation of pixel intensities, we can use the following scheme. First, we introduce another function, the square coverage $T_S(\boldsymbol{x},t)$:

$$T_S(\boldsymbol{x},t) = \sum_i \left[\iint A(\boldsymbol{x}' + \boldsymbol{x} - \boldsymbol{p}_i(t))\chi(\boldsymbol{x}')d\boldsymbol{x}'\right]^2, \qquad \textit{(4-18)}$$

where the intrinsic intensities have been removed and the contribution from each particle is squared. For every output pixel, we note that the convolution integrals for *T* and T_S are essentially weighted sums:

$$T(\boldsymbol{x},t) = \sum_i w_i^{(x)} a_i, \quad T_S(\boldsymbol{x},t) = \sum_i \left[w_i^{(x)}\right]^2 \qquad \textit{(4-19)}$$

Suppose the intrinsic intensities a_i are independent and have a Gaussian distribution with a mean of zero and a variance of s^2. Then, each output pixel will have a zero mean and a variance of:

$$s_{\boldsymbol{x}}^2 = \sum_i \left(w_i^x\right)^2 s^2 = T_S(\boldsymbol{x},t)s^2 \qquad \textit{(4-20)}$$

Thus, to renormalize the distribution, we divide by the square root of T_S to produce a new texture function $T'(\boldsymbol{x},t)$ whose variance is s^2:

$$T'(\boldsymbol{x},t) = \frac{T(\boldsymbol{x},t)}{\sqrt{T_S(\boldsymbol{x},t)}} \qquad \textit{(4-21)}$$

Each pixel in the resulting texture will have the same Gaussian distribution. Note that because a particle can contribute to more than one pixel, their distributions will not be entirely independent. This, however, is necessary if we want to be able to correlate the movement of particles over time.

A simpler way of controlling the distribution is by simply dividing out the coverage level $T_C(\boldsymbol{x},t)$:

$$T_C(\boldsymbol{x},t)=\sum_i \iint A(\boldsymbol{x}'+\boldsymbol{x}-\boldsymbol{p}_i(t))\chi(\boldsymbol{x}')d\boldsymbol{x}' \quad \Rightarrow \quad T'(\boldsymbol{x},t)=\frac{T(\boldsymbol{x},t)}{T_C(\boldsymbol{x},t)} \tag{4-22}$$

This way, if the particle intensities are limited to the range $-1<a_i<1$, then the texture pixel will also be limited to $-1<T'(\boldsymbol{x},t)<1$. In addition, increasing the texture coverage reduces the standard deviation of the texture pixels. Therefore, texture particles packed closely together tend to "average out", which may or may not be a desirable property.

One unfortunate consequence of these two normalization techniques is that it affects the temporal correlation of the texture pixels over time. Another conceivable solution is to avoid normalizing the texture pixel intensity, and instead control the texture coverage. If we can constrain the coverage to some range, then as a consequence the pixel intensity will be bounded. The next section describes a method for maintaining a consistent separation between particles, resulting in a somewhat uniform texture coverage.

Experimentation with these three methods has shown that no solution is significantly better or worse. As expected, the first method yields a uniform standard of deviation, but the frame-to-frame coherence is not as good. In contrast, the third method produces a more consistent evolution, but the distribution of intensities is not as uniform. The second method seems like a compromise between the other two.

4.2.4 PARTICLE ADVECTION AND ADJUSTMENT

As a minimum, over time the texture function should roughly satisfy

$$T(\boldsymbol{x},t)\cong T\left(\lambda_t^{dt}(\boldsymbol{x}),t+dt\right) \tag{4-23}$$

so that the intrinsic intensities of the particles remain constant as they advect over time. However, this condition is not sufficient to guarantee that the resulting sequence of images is a good representation of the vector field.

For example, consider the simple evolution

$$\boldsymbol{d}(\boldsymbol{x})=\hat{\boldsymbol{x}}. \tag{4-24}$$

This yields the condition

$$T(\boldsymbol{x},t)\cong T(\boldsymbol{x}-t\hat{\mathbf{x}},0) \tag{4-25}$$

for $\|\boldsymbol{x}\| > t$. However, for $\|\boldsymbol{x}\| < t$, the particles have appeared out of a source at the origin, and have no constraint. What intensities should these new particles take on?

In regions where the evolution vector field $\boldsymbol{d}$ has a positive divergence, the particles will "spread out", and the frequency extent of the texture will be reduced. After LIC is applied, the frequency components perpendicular to the direction of the vector field $\boldsymbol{f}$ will therefore be diminished, which reduces the amount of detail that can be discerned. Ideally, we would like the texture's frequency extent to be homogeneous in order to maintain a constant level of detail in the images output by LIC.

The last section described how DLIC generates the texture by using a collection of particles distributed over the pixel grid. When it evolves the particles through time, we no longer have the luxury of choosing the distribution of particles, as they move according to the motion described by $\boldsymbol{d}$. In regions where $\boldsymbol{d}$ diverges and as a result the particle separation is too great, DLIC generates new random detail. In addition, when particles are packed too closely together, they do not produce any benefit in the level of detail in the output image. Therefore, to avoid unnecessarily tracking extra particles, DLIC collapses the particles together to a single particle.

The method I use to determine where new detail needs to be created and where particles need to be merged is by examining the coverage level for each texture pixel:

$$T_C(\boldsymbol{x},t) = \sum_i \iint A(\boldsymbol{x}' + \boldsymbol{x} - \boldsymbol{p}_i(t))\chi(\boldsymbol{x}')d\boldsymbol{x}' \qquad (4\text{-}26)$$

The result of this computation can be seen in *Figure 4-7*, and in my implementation, this occurs at the same time that the texture is generated.

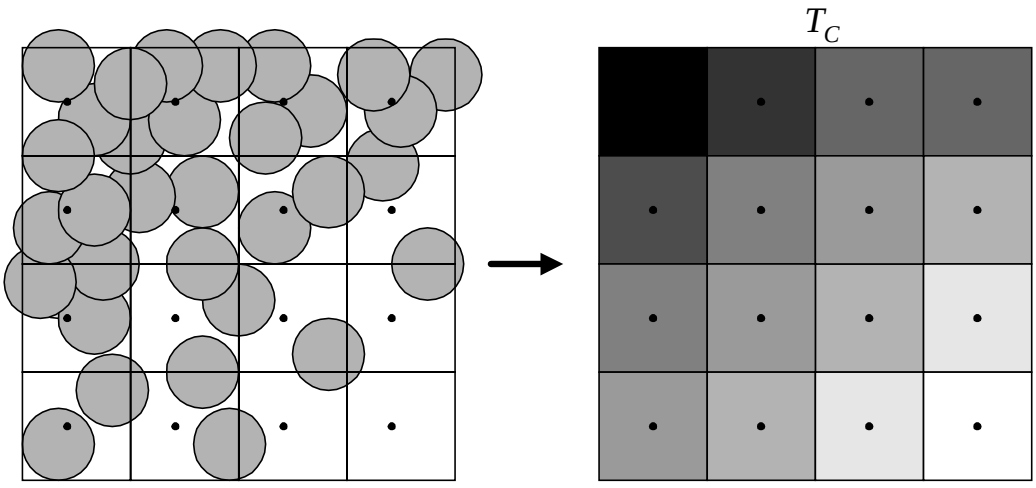

Figure 4-7: Texture coverage

For every texture pixel ***x***, if the coverage is below some minimum threshold, we need to create more detail. One way is to take a particle that contributes to the pixel and replace it with two particles spaced slightly apart, with different intrinsic intensities that average out to the intensity of the original particle. However, in my implementation I use a much simpler scheme, depicted in *Figure 4-8*. For each pixel with a low coverage (the three pixels in T_C to the right and bottom), a new particle is created near the pixel center with a completely random intensity. This results in new random intensities being introduced in the corresponding pixels of *T*. Clearly, this has the potential of introducing variations in the streamlines of the output images that would ruin the frame-to-frame correlation. Fortunately, the number of pixels along a streamline that fall below the minimum coverage level is usually very small. For example, suppose the mapping between points along a streamline between two successive time steps is as in *Equation 4-13*:

$$\kappa(s) = \begin{cases} \dfrac{1}{2S}, & -S < s < S \\ 0, & \text{otherwise} \end{cases} \qquad \textit{(4-27)}$$

$$u(s) = (1+\mu)s \quad \Rightarrow \quad \frac{du}{ds} = 1+\mu$$

Also, suppose the separation between adjacent streamlines expands by a factor $(1+\beta)$. Then, if we want to maintain a minimum particle separation of Δx, the number of new particles created within the streamline convolution range is approximately:

$$\frac{1}{\Delta x} 2S - \frac{1}{\Delta x} \frac{2S}{(1+\beta)(1+\mu)} = \frac{2S}{\Delta x} \frac{\beta + \mu + \beta\mu}{(1+\beta)(1+\mu)} \qquad \textit{(4-28)}$$

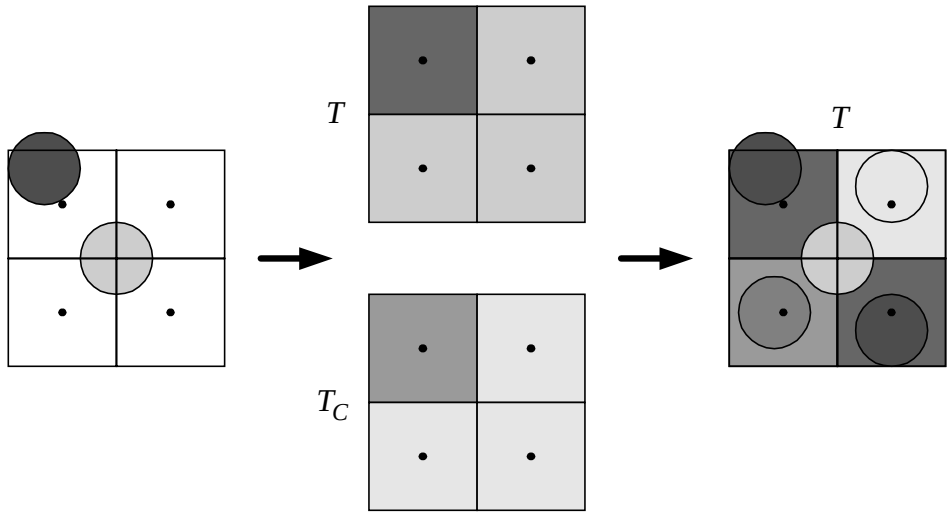

Figure 4-8: Particle creation

For small expansion factors μ and β, very few particles are created relative to the total number of particles in a streamline convolution, $2S/\Delta x$. Therefore, in most cases where streamlines are evolving gradually, the effect of adding random particles to it is very gradual.

One addition benefit is that regions that are not mapped by a time step of the evolution will automatically have new random detail added, since their coverage will be zero, which is presumably below the minimum coverage. Thus, for the example of radial evolution ***d*** field (*Equation 4-24*), the source at the origin will be continuously filled with new random particles, producing a never-ending amount of detail. Similarly, for a different evolution field ***d*** that points inward at edges of the domain ***D***, the same technique will create new particles to flow inward.

For each pixel whose coverage exceeds a certain maximum level, we need to somehow eliminate the excess particles around that pixel. The method I use in this case is also fairly simple, and is illustrated in *Figure 4-9*. First, both the texture and texture coverage are computed for the current set of particles. Then, for each pixel whose coverage level is too great, every particle close to the pixel center is deleted. Next, a new particle is created near the pixel center whose intensity equals the value of the texture at that pixel. The net effect is the collapse of particles packed too tightly together into a single particle with a rough average of their intensities. Thus, the input texture's evolution for that pixel remains nearly continuous.

In addition, for particles that leave the domain ***D***, my implementation immediately deletes them to avoid tracking extra detail. If the evolution field ***d*** is defined on some larger domain then it is very possible that the particle will return to ***D*** at some future time, for example for some oscillatory motion. However, experimentation has shown that this sort of discrepancy is

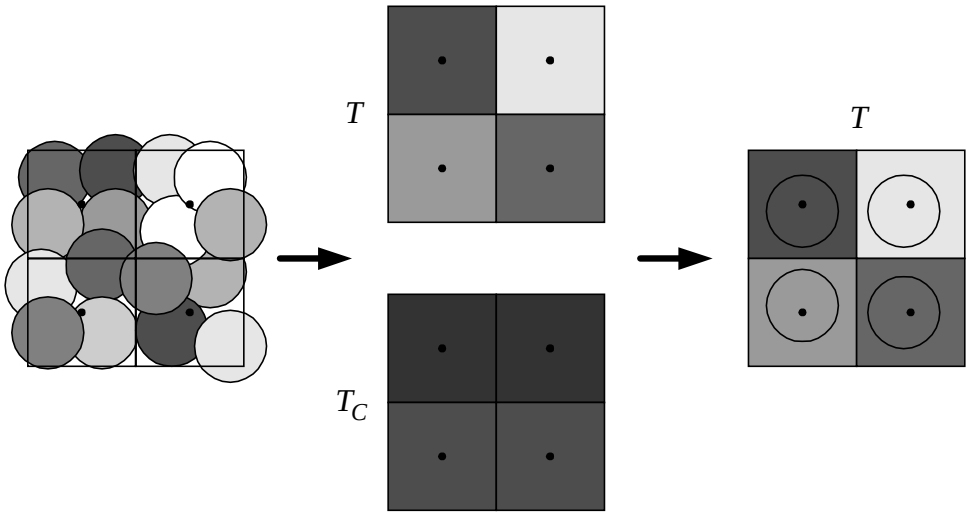

Figure 4-9: Particle merging

not noticeable. Therefore, the extra effort involved in tracking these particles is not worth it.

It should be noted that though these techniques work well from one frame to the next, it does not maintain consistency across a long, looping animation sequence, where $\boldsymbol{f}$ and $\boldsymbol{d}$ are periodic. Because particles are destroyed and created randomly, there will be little correlation between the output pixels of images separated by one period of the loop. This can be corrected by a post-processing step to gradually blend between frames one period apart, as described by Stalling and Hege [18].

In order to advect the particles according to $\boldsymbol{d}$, we can use the same techniques described earlier on integrating first-order ODEs to compute the evolution of the particles over a time step. A fourth-order Runge-Kutta method will of course yield excellent results, but it may be overkill. If the motion of the particles is relatively small over a time step, it usually suffices to use a simple integration scheme like Euler's method. Despite the accumulated errors in the particle positions, the streamlines that are visualized will always accurately reflect $\boldsymbol{f}$. Since the errors are usually small, the average resulting error in the line integral convolution, which takes into account a large number of particles, will be even smaller. Although these small errors in the line integral convolution accumulate over time, the effect is gradual and does not significantly affect the frame-to-frame coherence.

4.2.5 LINE INTEGRAL CONVOLUTION

Though Line Integral Convolution is used as the final stage of DLIC, it could equally well be replaced by another texture-based vector field visualization technique in order to either speed up the computation or increase the quality. For example, in the next section I will describe how the Fast Line Integral Convolution method is used to speed up rendering the animation by an order of magnitude. For fields that contain singularities, a particle-scattering method may improve the quality of the image near the singularities.

It is absolutely critical that the rendering method is deterministic and stable. By deterministic I mean that if a particular time frame in the visualization sequence is rendered twice, the results should be essentially indistinguishable. Also, for two successive frames where the input texture and vector field changes only slightly, the algorithm should be stable in that the resulting output images also change very slightly. For the original LIC algorithm, randomness does not enter anywhere, so it is certainly deterministic. In addition, at each pixel a line integral convolution is computed independently of the results of other pixels, so small changes in the texture and vector field will likewise have a small effect on the output. Care must be taken when using other rendering methods that do not guarantee these properties.

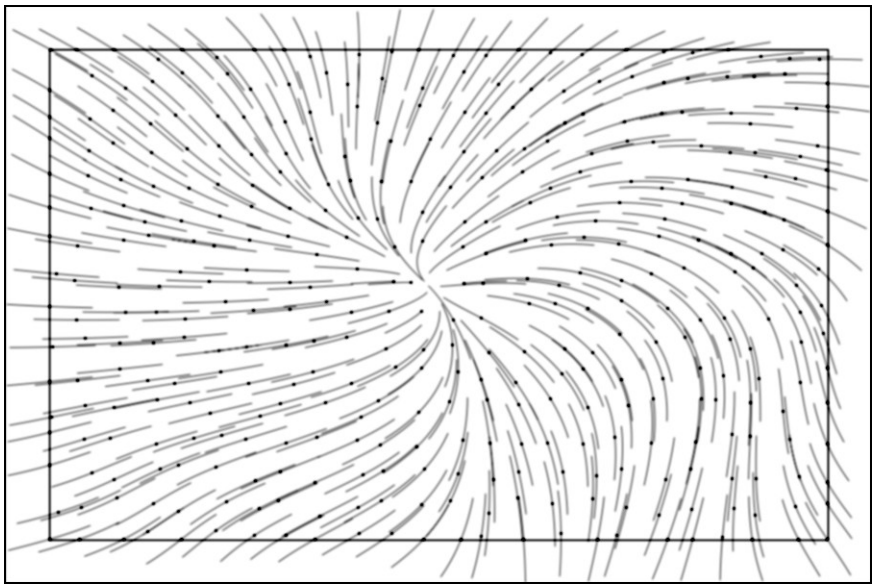

Figure 4-10: Expanding the domain so that the convolutions remain entirely inside

One common problem with these techniques is that the behavior near the edge of the domain $\boldsymbol{D}$ is ill defined. On method for dealing with this is to simply terminate the streamline at the edge and shorten the convolution kernel. Unfortunately, modifying the kernel in this way noticeably affects the output image, especially at the corners where streamlines can be extremely short. Another method is to reverse the direction of the streamline at the edge. This effectively samples the same pixels in the input texture more than once, which again adversely affects the output image.

The solution I use is to increase the domain $\boldsymbol{D}' \supset \boldsymbol{D}$ of the vector fields $\boldsymbol{f}$ and $\boldsymbol{d}$ and the texture image *T*. Assuming the LIC convolution kernel $\kappa(s)$ has a limited extent $s_{\max}$, i.e.

$$\kappa\left(|s| > s_{\max}\right) = 0\,, \tag{4-29}$$

and the domain $\boldsymbol{D}'$ is large enough that

$$\boldsymbol{x} \in \boldsymbol{D} \;\Rightarrow\; \left(\|\boldsymbol{y} - \boldsymbol{x}\| \le s_{\max} \;\Rightarrow\; \boldsymbol{y} \in \boldsymbol{D}'\right), \tag{4-30}$$

then LIC will never need to leave the domain $\boldsymbol{D}'$. In other words, if $\boldsymbol{D}$ is completely contained in $\boldsymbol{D}'$ with its edge at least a distance $s_{\max}$ inside, then a line integral convolution seeded at a point in $\boldsymbol{D}$ will never have to trace a streamline outside $\boldsymbol{D}'$, because the kernel would be zero there anyway. This is illustrated in *Figure 4-10*, where the outer rectangle is $\boldsymbol{D}'$ and the inner is $\boldsymbol{D}$.

4.3 IMPLEMENTATION DETAILS

Although the above description of DLIC is essentially complete, a few more details of our implementation are worth discussing.

4.3.1 TRANSFORMING THE FIELDS

One extension that can be added naturally to DLIC is to again separate the output image resolution from the input texture resolution via a transformation $\boldsymbol{\Phi}$. Indeed, this transformation can be made to vary over time, to allow an animation sequence to be zoomed, panned, or rotated:

$$O(\boldsymbol{x}_0, t) = \iint I(\boldsymbol{\Phi}_t(\boldsymbol{x}_0 + \boldsymbol{x}), t)\chi(\boldsymbol{x})d\boldsymbol{x} \qquad (4\text{-}31)$$

Unfortunately, this is not the best method because we want the input texture to match the resolution of the output image. Otherwise, when the transformation zooms in, the level of detail will not increase, and the frequency extent of the streamlines visualized in O will decrease. A more desirable way to allow for time-dependent transformations is to instead transform the fields $\boldsymbol{f}$ and $\boldsymbol{d}$:

$$\begin{aligned} \boldsymbol{f}'(\boldsymbol{x}, t) &= \boldsymbol{f}(\boldsymbol{\Phi}_t(\boldsymbol{x}), t) \\ \boldsymbol{d}'(\boldsymbol{x}, t) &= \boldsymbol{d}(\boldsymbol{\Phi}_t(\boldsymbol{x}), t) \end{aligned} \qquad (4\text{-}32)$$

In addition, the particles that generate the texture T must reside in the coordinate system of $\boldsymbol{f}$ and $\boldsymbol{d}$, and therefore the texture generation procedure must be modified as follows:

$$T(\boldsymbol{x}, t) = \iint \left[\sum_i a_i A(\boldsymbol{x}' + \boldsymbol{x} - \boldsymbol{\Phi}_t(\boldsymbol{p}_i(t))) \right] \chi(\boldsymbol{x}')d\boldsymbol{x}' \qquad (4\text{-}33)$$

This gives us the desired effect, since zooming in causes the particles to separate, and as a result the particle adjustment stage of DLIC will add in new detail automatically. Similarly, zooming out or translating will produce new particles at the edges where they are needed.

4.3.2 FAST DLIC

One very important extension of DLIC is to make use of the Fast Line Integral Convolution algorithm to accelerate rendering, particularly for long animation sequences. As I mentioned earlier, FLIC is sensitive to the ordering of the initial samples, and using a random sampling for each frame will destroy both the necessary conditions of determinism and stability.

Using a pseudo-random generator, we could very well seed samples in exactly the same order for each frame, which would make it deterministic. However, suppose we have a particle

motion field $\boldsymbol{d}$ that moves particles slowly and uniformly in one direction. If this motion does not fall along pixel centers for each frame, then the sampling will be different for each frame relative to the particles. A better approach is to begin sampling at particle centers in a pseudo-random ordering. If the same frame is rendered twice, even if it is translated, the sampling order will be the same relative to the particles. Even for more complicated evolutions of the particles, the sampling points will follow the particles, and hence FLIC will sample the streamlines in a mostly consistent order. Thus, results produced in this way will be fairly stable.

One other issue is how to deal with particles that are removed or created. In my implementation, I keep an ordered list of particles that are used to seed the FLIC samples. When particles are deleted, they are simply removed from that list. Newly created particles are added to the end of the list in a random order. This way, new particles that are created will have a small influence on the results of FLIC via the texture, and will not significantly affect the sampling.

Though this method is not guaranteed to give an absolutely stable ordering, particularly because of the disconnect between the continuous domain of the particles and the discreet representation of the texture and output image, experimentation has shown that this method works very well. The resulting order-of-magnitude speed-up makes this extension well worth the effort.

4.3.3 LOCAL CONTRAST NORMALIZATION

Consider the vector field

$$\boldsymbol{f}(\boldsymbol{r}) = \hat{\boldsymbol{\theta}}, \tag{4-34}$$

whose streamlines go in concentric circles around the origin, and the kernel:

$$\kappa(s) = \begin{cases} \dfrac{1}{2S}, & -S < s < S \\ 0, & \text{otherwise} \end{cases} \tag{4-35}$$

Then, for radii $2\pi r < 2S$ the line integral convolution "folds in" on itself, integrating over the same texture points more than once. Close to the origin, a very small circle of the texture is sampled multiple times. This has the effect that texture values near the origin contribute an inordinate amount to the output, producing exaggerated intensity values. *Figure 4-11* shows the degree of over-sampling in such a vector field for streamlines at different radii.

Another example of this problem is the vector field

$$\boldsymbol{f}(x, y) = \hat{\boldsymbol{x}}, \tag{4-36}$$

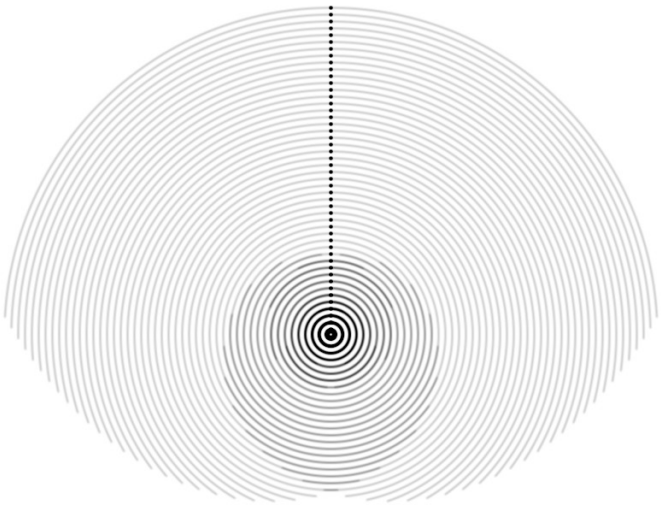

Figure 4-11: Convolution extent in a circular vector field – shaded by over-sampling

with uniformly horizontal streamlines. Suppose line integral convolutions are computed by discrete samples spaced a distance equal to the texture resolution. Then, output intensities $I(\boldsymbol{x},t)$ at pixel centers will sample the texture only at its pixel centers, while any other output intensities will sample interpolated points that lie between pixel centers. Therefore, the output intensity of a point not at a pixel center will effectively be a weighted sum of a much larger set of pixels than for the output intensity at a pixel center. As more texture pixels are averaged together, the variance goes down. Hence, the contrast level in the output intensity is non-uniform, even for this simple vector field.

To solve this problem, we suppose for the sake of simplicity that the pixels in the input texture have the same random distribution of values, and that we know its mean μ and variance s^2 (for example, a uniform or Gaussian distribution is common). We would like to compute the mean μ_I and variance s_I^2 of the output intensity $I(\boldsymbol{x})$ at a constant time

$$I(\boldsymbol{x}) = \int T(\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}}(s))\kappa(s)ds\,. \qquad (4\text{-}37)$$

Of course, the continuous integral is implemented in the algorithm as a discrete sum, and the texture values are interpolated between texture pixels:

$$\tilde{I}(\boldsymbol{x}) = \Delta s \sum_n \tilde{T}(\tilde{\boldsymbol{\sigma}}_{\boldsymbol{x}}(n\Delta s))\tilde{\kappa}(n\Delta s) \qquad (4\text{-}38)$$

Each texture interpolation can be written as a weighted sum over the texture pixels:

$$\tilde{\tilde{T}}\left(\tilde{\sigma}_{\boldsymbol{x}}(n\Delta s)\right)=\sum_m w_m^n\, T(\boldsymbol{x}_m) \tag{4-39}$$

Therefore, the output intensity is itself a weighted sum over the texture pixels:

$$\tilde{I}(\boldsymbol{x})=\sum_m\left[\Delta s\sum_n w_m^n\,\tilde{\kappa}(n\Delta s)\right]T(\boldsymbol{x}_m)=\sum_m W_m T(\boldsymbol{x}_m)$$

For a large number of pixels, by the central limit theorem we know that the resulting mean and variance of the output intensity is:

$$\mu_I\cong\mu\sum_m W_m,\quad s_I^2\cong s^2\sum_m (W_m)^2 \tag{4-40}$$

Note that this result is exact if the texture values have a Gaussian distribution.

Computing the mean can be simplified tremendously with a few assumptions. Usually, texture interpolation is done in such a way that:

$$\sum_m w_m^n=1 \tag{4-41}$$

In other words, though several pixels contribute to the interpolated result, the total contribution is always unity. For example, this is true with bilinear interpolation. In fact, any convolution kernel with unit volume used to interpolate the texture will have this property. Then,

$$\mu_I=\mu\sum_n\sum_m w_m^n\Delta s\,\tilde{\kappa}(n\Delta s)=\mu\Delta s\sum_n\tilde{\kappa}(n\Delta s), \tag{4-42}$$

which is constant for every pixel. Of course, in the simplest case, if the mean of each pixel is zero, then the mean of the output intensity will also be zero.

Unfortunately, there is no simple way of computing the variance. The brute-force method is to store the entire set of texture pixel weights W_m in memory, and compute it in parallel with the convolution sum. With all the weights set to zero initially, every time we add a weighted texture pixel

$$\left[\Delta s\, w_m^n\,\tilde{\kappa}(n\Delta s)\right]T(\boldsymbol{x}_m) \tag{4-43}$$

to the convolution sum, we accumulate the weight in the corresponding pixel weight:

$$W_m'=W_m+\Delta s\, w_m^n\,\tilde{\kappa}(n\Delta s) \tag{4-44}$$

After the convolution sum is computed, the texture pixel weights can be squared and summed to get the total variance.

Noting that most of the coefficients w_m^n are zero, with some careful programming it is possible to perform the computation with little extra effort. First, in addition to storing the texture pixel weights, we keep a running sum-of-squares W_{tot}^2, which is initialized to zero. Then, for every weighted texture pixel sampled, we apply the rules:

$$\begin{aligned} W_m' &= W_m + \Delta s\, w_m^n\, \tilde{\kappa}(n\Delta s) \\ \left(W_{tot}^2\right)' &= W_{tot}^2 - (W_m)^2 + (W_m')^2 \end{aligned} \qquad (4\text{-}45)$$

This way, the sum-of-squares is always equal to the sum of the squares of all the W_m, without needing to consider the pixels that never take part in the convolution sum in the first place.

Finally, one additional improvement can be applied if the convolution kernel $\tilde{\kappa}(n\Delta s)$ has a finite extent, i.e:

$$\tilde{\kappa}(n\Delta|s| > n_{\max}\Delta s) = 0 \qquad (4\text{-}46)$$

In this case, every two pixels in the convolution sum will be separated by a distance of at most $2n_{\max}\Delta s$. Given a pixel separation of Δx and a texture interpolation kernel of width Δx_T, every texture sample will be separated by at most *M* pixels, where

$$M = \frac{2n_{\max}\Delta s}{\Delta x} + \Delta x_T . \qquad (4\text{-}47)$$

Therefore, there is no need to store the weights for every pixel in the texture. Instead, a square array $\overline{W}$ of *M* by *M* pixel weights is large enough to cover the entire convolution sum. A simple way of mapping the pixels is via:

$$W_m \quad \rightarrow \quad W_{x,y} = \overline{W}(x \bmod M, y \bmod M) \qquad (4\text{-}48)$$

Finally, now that we are able to compute the mean and variance of the output intensity, we can renormalize the distribution. For example, to center the distribution at zero and give it a unit variance, we compute:

$$I'(\boldsymbol{x},t) = \frac{I(\boldsymbol{x},t) - \mu_I}{\sqrt{s_I^2}} \qquad (4\text{-}49)$$

The resulting image has a uniform contrast throughout.

4.4 ALTERNATIVES

In many ways, the core idea behind DLIC is more general than the techniques that I have described above. It is a texture-based method for visualizing a vector field ***f***, where the texture somehow evolves over time according to another vector field ***d***. I have found that my particular implementation works wonderfully when visualizing experiments in electromagnetism. However, it is certainly possible to use alternative methods that fit within the same framework. This section explores some other possibilities and explains why I chose my particular implementation.

4.4.1 DIRECT PARTICLE IMAGING

From the previous discussion of the particle image-formation process, it may seem that the intermediate texture image is unnecessary. Indeed, at each time frame we could very well iterate over each particle, scattering the intensity value in the output image along its streamline. We could use the same particle advection and adjustment techniques described earlier to manage the particles, and skip the texture generation stage altogether. This method reduces the number of translations between the continuous and discrete representations, potentially producing higher quality results. Unfortunately, there does not seem to be any simple way of accelerating this process. There is no way of taking advantage of the fact that many particles lie close to the same streamlines. Thus, it is not a realistic implementation for long animation sequences. In contrast, DLIC is designed to use a texture image as an intermediate stage so that optimized texture-based visualization methods can be used.

4.4.2 TEXTURE WARPING

Since the texture is a representation of all the particles, it is conceivable that we could advect the texture itself according to ***d*** and eliminate the particles altogether. Indeed, this would save a tremendous amount of memory, since we would only need to store one scalar value per pixel. In contrast, there are roughly as many particles as pixels in the texture, and for each particle we need to store its position and intensity value.

But how can we make sure that the level of detail remains roughly the same throughout the texture as it evolves over time? Certainly, we don't have to worry about the texture becoming *too* detailed, since its resolution limits the level of detail that can be represented. However, if the ***d*** field diverges over time, the corresponding features of the texture will be stretched and its frequency extent will be reduced. One possible solution is to maintain a measure of the

"coverage" for each pixel $T_C(\boldsymbol{x},t)$. Initially, when the texture is set to random white noise, the coverage is uniformly set to 1. Then, T_C is warped in exactly the same way as *T*. If the warping is done in a conservative fashion, where

$$\sum_{\boldsymbol{x}} T_C(\boldsymbol{x},t) = C, \qquad \textit{(4-50)}$$

meaning that coverage values are transferred between pixels and not lost or created, then T_C serves as a measure of the frequency extent of each pixel. When the coverage falls below a certain level, this is the result of too much "stretching" of the texture. Therefore, random values can be added to the texture to increase the level of detail in the same way as adding random particles.

Unfortunately, repeatedly warping the texture, and therefore refiltering it ever frame, reduces the frequency extent much more than can be measured by a simple "coverage" indicator. In the paper on Unsteady Flow LIC (*UFLIC*) [16], Shen and Kao suggest applying a sharpening (high-pass) filter to the output and using it as the texture for the next frame. They also suggest mixing in a low-intensity random texture to continuously add small amounts of detail. However, adjusting the level-of-detail globally for every single frame is excessive, and this is essentially a feedback process that takes time to converge. Due to the arbitrary nature of the random detail added, if we tried to visualize a stationary vector field, the output would vary from frame to frame because of the continual sharpening and random detail added.

One extreme solution would be to explicitly compute the frequency content in the texture to find regions where the frequency extent has been limited by stretching and repeated filtering. However, this sort of technique would be computationally expensive. At this point, it makes more sense to simply track the particles that give rise to the texture via the procedure I described earlier. The memory cost is not prohibitive for today's computers.

4.4.3 PATH LINES

In the paper by Forssell and Cohen [7], it was suggested that tracing path lines instead of streamlines was more appropriate for time-varying fields. Unfortunately, this gives the vector visualization technique "predictive" powers, where an image at a certain time includes features from the vector field at a future time. For flow fields, it does take into account the motion of particles over time, but for fields where $\boldsymbol{f}$ and $\boldsymbol{d}$ are not equal, such as electromagnetic fields, this is not very useful.

The UFLIC method works better in that path lines are traced forward in time and the texture value is “deposited” in the output image at that time. Thus, it both accounts for the unsteady motion of particles over time, and does not expose the features in the vector field until the correct time. However, maintaining output images for several future time steps can take up a significant amount of memory for long path lines. And again, this does not work for fields where $\boldsymbol{f} \neq \boldsymbol{d}$.

The goal of DLIC is to visualize the vector field $\boldsymbol{f}$ at each moment in time in such a way that the temporal streamline correspondence can be discerned as specified in $\boldsymbol{d}$. Streamlines are indeed the correct way of visualizing the vector field $\boldsymbol{f}$ at each moment in time. The effect of the past and future is seen only in the correlation of intensities along streamlines over time.

5 EXPERIMENTS IN ELECTROMAGNETISM

The ultimate goal of this research project was to devise a way to visualize dynamic electromagnetic fields with the same amount of detail afforded by the Line Integral Convolution technique. To make use of the DLIC technique, we need to be able to compute $\boldsymbol{f}$ and $\boldsymbol{d}$ fields over time. Electric and magnetic fields can be computed via the principles of classical electromagnetism, for example with the help of a standard textbook such as Griffith's [9]. Describing the motion of the electric or magnetic field lines is typically not discussed, however.

5.1 FIELD LINE MOTION

For visualizing magnetic field lines, we can compute the magnetic field $\boldsymbol{f}$ using the Biot-Savart law, Ampère's law, or Maxwell's equations. Similarly, for electric field lines, $\boldsymbol{f}$ can be computed via Gauss's law, Faraday's law, or Maxwell's equations. Or, both fields can be computed via the scalar and vector potentials. The questions remains, however: how do we track the motion of the field lines over time?

5.1.1 DEFINITION OF A FIELD LINE

Although field lines are essentially imaginary constructs in physics, they can be used to aid our understanding of electromagnetism. A field line is simply a streamline in the electric or magnetic field at a constant time. In quasi-static situations, where the speed of light is much greater than any velocities we would perceive in our animation, these streamlines are easy to construct. For electric fields, we expect to see these streamlines start at positive charges and end at negative charges, while for magnetic fields the streamlines circle around currents. In the relativistic case, we take field lines to be streamlines over the field as perceived by observers located throughout space in the laboratory frame at the same time.

For texture-based vector visualization methods like (D)LIC, it is difficult to map three-dimensional fields to two-dimensional displays. Although it is sometimes possible to effectively use transparency to see the structure of the field, usually the results are much too convoluted. Instead, we take a two-dimensional "slice" of the vector field: points in the two-dimensional domain of $\boldsymbol{f}$ map to a plane in space, and the components of the vector field parallel to the plane map to the range of $\boldsymbol{f}$. We can describe the mapping from the domain of $\boldsymbol{f}$ to the vector field as:

$$\boldsymbol{M}(\boldsymbol{x}) = c\begin{bmatrix}\boldsymbol{M}_x & \boldsymbol{M}_y\end{bmatrix}\boldsymbol{x} + \boldsymbol{O}, \qquad (5\text{-}1)$$

where $\boldsymbol{M}_x$ and $\boldsymbol{M}_y$ are the orthonormal x- and y-axes in three-space, c is a scale factor, and $\boldsymbol{O}$ is to origin. Similarly, given the vector field at a point in space, we can project it back on to the plane with:

$$\boldsymbol{P}(\boldsymbol{E}(\boldsymbol{x}')) = \frac{1}{c}\begin{bmatrix}\boldsymbol{M}_x & \boldsymbol{M}_y\end{bmatrix}^T \boldsymbol{E}(\boldsymbol{x}') \qquad (5\text{-}2)$$

Thus, given the electric (or magnetic) field $\boldsymbol{E}(\boldsymbol{x}')$, we define *f* as:

$$\boldsymbol{f}(\boldsymbol{x}) = [\boldsymbol{P} \circ \boldsymbol{E} \circ \boldsymbol{M}](\boldsymbol{x}) \qquad (5\text{-}3)$$

Now that we have $\boldsymbol{f}$, how do we compute the corresponding $\boldsymbol{d}$ field? In the paper "Field Line Motion in Classical Electromagnetism" [1], Belcher and Olbert define the motion of field lines to be the drift motion of low energy test particles. This definition is physically motivated, and matches our intuition for how field lines should move. For example, if we scatter electric test charges along a magnetic field line, the charges will move with the field line over time. This is a physically plausible but not unique way to define field line motion.

5.1.2 MAGNETIC FIELDS

The following derivation is taken from Belcher and Olbert's article [1]. For visualizing magnetic fields, we start by considering the (non-relativistic) motion of an electric test charge in constant fields:

$$m\ddot{\boldsymbol{x}} = q(\boldsymbol{E} + \dot{\boldsymbol{x}} \times \boldsymbol{B}) \qquad (5\text{-}4)$$

First, we split the electric field into its components parallel and perpendicular to the magnetic field:

$$m\ddot{\boldsymbol{x}} = q\boldsymbol{E}_{\parallel} + q(\boldsymbol{E}_{\perp} + \dot{\boldsymbol{x}} \times \boldsymbol{B}) \qquad (5\text{-}5)$$

Since motion of the particle along the parallel component is motion along the magnetic field line, we make the choice to neglect this portion. Thus for fields where the electric and magnetic fields are not perpendicular, we are not computing the true motion of a particle, but a particle that is constrained to move perpendicular to the magnetic field. With only the perpendicular component, we can then make the substitution

$$\dot{\boldsymbol{x}} = \dot{\boldsymbol{x}}' + \boldsymbol{v}_{drift}, \text{ where } \boldsymbol{v}_{drift} = \frac{\boldsymbol{E}_{\perp} \times \boldsymbol{B}}{\boldsymbol{B}^2}, \qquad (5\text{-}6)$$

which yields the acceleration:

$$\begin{aligned} m\ddot{\boldsymbol{x}}'_{\perp} &= q\left(\boldsymbol{E}_{\perp} + \dot{\boldsymbol{x}}' \times \boldsymbol{B} + \frac{1}{\boldsymbol{B}^2}(\boldsymbol{E}_{\perp} \times \boldsymbol{B}) \times \boldsymbol{B} \right) \\ &= q\left(\dot{\boldsymbol{x}}' \times \boldsymbol{B} + \boldsymbol{E}_{\perp} + \hat{\boldsymbol{B}}\left(\hat{\boldsymbol{B}} \cdot \boldsymbol{E}_{\perp} \right) - \boldsymbol{E}_{\perp} \hat{\boldsymbol{B}}^2 \right) = q\, \dot{\boldsymbol{x}}' \times \boldsymbol{B} \end{aligned} \qquad \textit{(5-7)}$$

The solution to this is a gyrating motion around the magnetic field line, and therefore the solution to the motion of the particle is to drift along the field at $\boldsymbol{v}_{drift}$ while gyrating. Since this is a hypothetical test charge, we can make its gyrations as small as we like. Indeed, even if we allow the $\boldsymbol{E}$ and $\boldsymbol{B}$ fields to vary over space, we could make the radius of gyration small enough that the fields are effectively constant. Finally, if we allow the fields to vary over time as well, the period of gyration can be made much smaller than the time scale over which the fields change.

Thus, we define the motional vector field $\boldsymbol{d}$ for magnetic fields as follows:

$$\boldsymbol{d}(\boldsymbol{x}) = \left[\boldsymbol{P} \circ \left(\frac{\boldsymbol{E} \times \boldsymbol{B}}{\boldsymbol{B}^2} \right) \circ \boldsymbol{M} \right](\boldsymbol{x}),$$

where the domain of $\boldsymbol{d}$ is mapped to three-space, the motion is computed at that point, and then projected back to the plane.

Interestingly, though this motion only has a physical interpretation if the drift velocity is less than the speed of light, it will still give us an intuition for the direction of energy flow since it is parallel to the Poynting vector. For electromagnetic waves, where

$$\boldsymbol{B} = \frac{1}{c} \hat{\boldsymbol{k}} \times \boldsymbol{E}\ ,$$

the drift velocity is the speed of light in the direction of propagation:

$$\boldsymbol{v}_{drift} = \frac{\boldsymbol{E} \times \boldsymbol{B}}{\boldsymbol{B}^2} = \frac{c}{\boldsymbol{E}^2} \boldsymbol{E} \times \left(\hat{\boldsymbol{k}} \times \boldsymbol{E} \right) = c \cdot \hat{\boldsymbol{k}}$$

5.1.3 ELECTRIC FIELDS

Similarly, for visualizing electric fields, we consider the motion of a hypothetical magnetic test charge. Its equation of motion is:

$$m\ddot{\boldsymbol{x}} = q_m \left(\boldsymbol{B} - \dot{\boldsymbol{x}} \times \boldsymbol{E} / c^2 \right) \qquad \textit{(5-8)}$$

Again, ignoring the component of the magnetic field that is parallel to the electric field, since it would move the particle parallel to the field lines, we can make the drift velocity substitution

$$\dot{\boldsymbol{x}} = \dot{\boldsymbol{x}}' + \boldsymbol{v}_{drift}\ ,\ \text{where}\ \boldsymbol{v}_{drift} = c^2 \frac{\boldsymbol{E} \times \boldsymbol{B}_{\perp}}{\boldsymbol{E}^2}\ , \qquad \textit{(5-9)}$$

which yields the equation:

$$\begin{aligned} m\ddot{\boldsymbol{x}}'_{\perp} &= q\left(\boldsymbol{B}_{\perp} - \dot{\boldsymbol{x}}' \times \boldsymbol{E}/c^2 - \frac{1}{\boldsymbol{E}^2}(\boldsymbol{E} \times \boldsymbol{B}_{\perp}) \times \boldsymbol{E} \right) \\ &= q\left(-\dot{\boldsymbol{x}}' \times \boldsymbol{E}/c^2 + \boldsymbol{B}_{\perp} + \hat{\boldsymbol{E}}\left(\hat{\boldsymbol{E}} \cdot \boldsymbol{B}_{\perp}\right) - \boldsymbol{B}_{\perp}\hat{\boldsymbol{E}}^2 \right) = -\frac{q}{c^2}\dot{\boldsymbol{x}}' \times \boldsymbol{E} \end{aligned} \qquad \textit{(5-10)}$$

The solution to this is again a gyrating motion, and thus the motion of the magnetic monopole is also the drift velocity plus the gyration. Electric and magnetic fields that vary over space and time can be made effectively constant for small enough gyrations so that this motion is still a good approximation. Thus, the motional field for electric field lines is defined as:

$$\boldsymbol{d}(\boldsymbol{x}) = \left[\boldsymbol{P} \circ \left(c^2 \frac{\boldsymbol{E} \times \boldsymbol{B}}{\boldsymbol{E}^2} \right) \circ \boldsymbol{M} \right](\boldsymbol{x}) \qquad \textit{(5-11)}$$

Though this physical interpretation also only holds for drift velocities less than the speed of light, the motion still depicts the flow of energy. In this case, the drift velocity for electromagnetic waves again yields the speed of light in the direction of propagation.

5.2 THE FALLING MAGNET

One experiment that we wanted to visualize was a permanent magnet falling through the axis of a conducting ring. This experiment is a good demonstration of induction and the resulting Eddy currents in the ring.

5.2.1 EXPERIMENTAL SETUP

The experiment consists of two objects, a permanent magnet and a conducting ring,

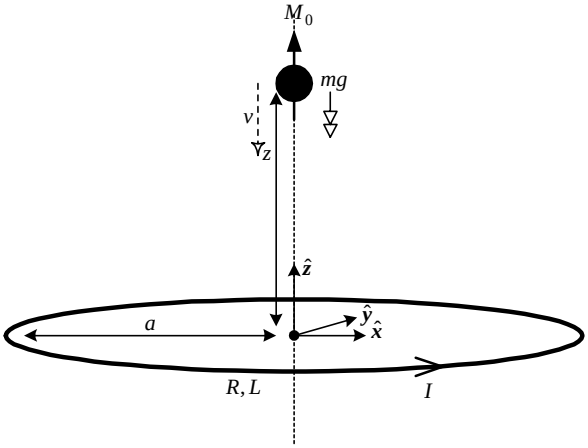

Figure 5-1: Falling magnet experimental setup

illustrated in *Figure 5-1*. For the sake of computation, the magnet is modeled as an infinitesimal magnetic dipole, and the conducting ring has an infinitesimal cross-sectional area. The ring is fixed, centered at the origin with its axis pointing in the $\hat{\boldsymbol{z}}$ direction, and has radius *a*, resistance *R*, and inductance *L*. The dipole is constrained to slide along the axis of the ring with its dipole moment $M_0\hat{\boldsymbol{z}}$ pointing along the axis. Initially, the dipole is located at a height *z* above the ring and has a zero velocity *v*, and the current *I* in the ring also starts at zero. Gravity takes it toll, and the dipole begins to fall down into the ring. The change in flux through the ring gives rise to a current in the ring, giving rise to a magnetic field, which in turn affects the dipole.

5.2.2 COMPUTING THE FIELDS

The electric and magnetic fields can be computed separately for the dipole and ring and then superimposed. For a stationary dipole, the magnetic field can be computed by:

$$\boldsymbol{B}_{dip}(\boldsymbol{r}) = \frac{\mu_0}{4\pi} r^{-3}\left[3(\boldsymbol{m}\cdot\hat{\boldsymbol{r}})\hat{\boldsymbol{r}} - \boldsymbol{m}\right] = \frac{\mu_0 M_0}{4\pi r^3}\left(3r_z\hat{\boldsymbol{r}} - \hat{\boldsymbol{z}}\right) \qquad (5\text{-}12)$$

Note that $\boldsymbol{r}$ is measured in the coordinate system whose origin is at the dipole's center. If the dipole is moving with a constant velocity in the laboratory frame, the magnetic field gives rise to an electric field:

$$\boldsymbol{E}_{dip}(\boldsymbol{r}) = \boldsymbol{v}\times\boldsymbol{B}_{dip} = v\left(\hat{\boldsymbol{z}}\times\boldsymbol{B}_{dip}\right) \qquad (5\text{-}13)$$

Additionally, if the dipole is accelerating, it will produce other radiating terms. For this experiment, the time scale is much larger than the time it takes for light to traverse the domain of the field we are concerned with, so we can ignore these extra terms.

For the ring of current, the computation is much more difficult. We begin by computing the vector potential $\boldsymbol{A}$ in the coordinate system centered at the ring's center:

$$\boldsymbol{A}_{ring}(\boldsymbol{r}) = \frac{\mu_0}{4\pi}\int\frac{\boldsymbol{I}}{\rho}dl = \frac{\mu_0 I}{4\pi}\int_0^{2\pi}\frac{-\hat{\boldsymbol{x}}\sin\theta + \hat{\boldsymbol{y}}\cos\theta}{\sqrt{(r_x - a\cos\theta)^2 + (r_y - a\sin\theta)^2 + r_z^2}}\,a\,d\theta \qquad (5\text{-}14)$$

The scalar potential is zero since there are no electric charges. We can simplify this somewhat since the problem has azimuthal symmetry by constraining it to the *x*-*z* plane. In addition, we can remove the dependence on radius by doing a change of coordinates:

$$\vec{x}' = a\frac{r_x\hat{x}+r_y\hat{y}}{\sqrt{r_x^2+r_y^2}}, \quad \vec{y}' = a\frac{-r_y\hat{x}+r_x\hat{y}}{\sqrt{r_x^2+r_y^2}}, \quad \vec{z}' = a\hat{z}$$
$$r_x' = \frac{1}{a}\sqrt{r_x^2+r_y^2}, \qquad r_y' = 0, \qquad r_z' = \frac{r_z}{a} \tag{5-15}$$

The integral then becomes:

$$\begin{aligned}\boldsymbol{A}_{ring}(\boldsymbol{r}') &= \frac{\mu_0 I}{4\pi}\int_0^{2\pi}\frac{-\vec{\boldsymbol{x}}'\sin\theta+\vec{\boldsymbol{y}}'\cos\theta}{\sqrt{(r_x'-\cos\theta)^2+(r_y'-\sin\theta)^2+r_z'^2}}d\theta \\ &= \frac{\mu_0 I}{4\pi}\int_0^{2\pi}\frac{-\vec{\boldsymbol{x}}'\sin\theta+\vec{\boldsymbol{y}}'\cos\theta}{\sqrt{r_x'^2+r_z'^2+1-2r_x'\cos\theta}}d\theta\end{aligned} \tag{5-16}$$

The electric field can therefore be computed by

$$\begin{aligned}\boldsymbol{E}_{ring}(\boldsymbol{r}') &= -\frac{\partial}{\partial t}\boldsymbol{A} = \frac{\mu_0}{4\pi}\frac{dI}{dt}\int_0^{2\pi}\frac{-\vec{\boldsymbol{x}}'\sin\theta+\vec{\boldsymbol{y}}'\cos\theta}{\sqrt{r_x'^2+r_z'^2+1-2r_x'\cos\theta}}d\theta \\ &= \frac{\mu_0}{2\pi}\frac{dI}{dt}\vec{\boldsymbol{y}}'\int_0^{\pi}\frac{\cos\theta}{\sqrt{r_x'^2+r_z'^2+1-2r_x'\cos\theta}}d\theta\end{aligned} \tag{5-17}$$

where the integral has been simplified by taking advantage of symmetry. This is an elliptic integral and must be evaluated numerically.

The magnetic field is

$$\begin{aligned}\boldsymbol{B}_{ring}(\boldsymbol{r}') &= \nabla\times\boldsymbol{A} = -\frac{\partial A_y}{\partial r_z'}\vec{\boldsymbol{x}}' + \frac{\partial A_x}{\partial r_z'}\vec{\boldsymbol{y}}' + \left(\frac{\partial A_y}{\partial r_x'}-\frac{\partial A_x}{\partial r_y'}\right)\vec{\boldsymbol{z}}' \\ &= \frac{\mu_0 I}{4\pi}\int_0^{2\pi}\left(-\frac{1}{2}\right)\frac{-2\vec{\boldsymbol{x}}'r_z'\cos\theta-2\vec{\boldsymbol{y}}'r_z'\sin\theta+2\vec{\boldsymbol{z}}'\left[(r_x'-\cos\theta)\cos\theta-(r_y'-\sin\theta)\sin\theta\right]}{\left[(r_x'-\cos\theta)^2+(r_y'-\sin\theta)^2+r_z'^2\right]^{3/2}}d\theta \\ &= \frac{\mu_0 I}{2\pi}\int_0^{\pi}\frac{\vec{\boldsymbol{x}}'r_z'\cos\theta+\vec{\boldsymbol{z}}'(1-r_x'\cos\theta)}{\left(r_x'^2+r_z'^2+1-2r_x'\cos\theta\right)^{3/2}}d\theta\end{aligned} \tag{5-18}$$

where again symmetry has exploited to simplify the integral. Although the equations for computing the fields at a point seem formidable, they can both be evaluated by computing two fairly coarse numeric integrals.

5.2.3 EQUATIONS OF MOTION

Since the dipole is constrained to lie along the axis of the ring with its dipole moment pointing along the axis, it is easy to compute the force it feels:

$$\boldsymbol{F} = -mg\hat{\boldsymbol{z}} + \nabla(\boldsymbol{m}\cdot\boldsymbol{B}) = \left(-g + M_0\frac{\partial B_z}{\partial z}\right)\hat{z} \qquad (5\text{-}19)$$

Fortunately, the magnetic field along the axis can be evaluated in closed form:

$$B_z = \frac{\mu_0 I}{2}\cdot\frac{a^2}{\left(a^2 + r_z^2\right)^{3/2}} \qquad (5\text{-}20)$$

Therefore, if we use the variable z to denote the height of the dipole over the ring, the equation of motion for the dipole is:

$$m\frac{d^2 z}{dt^2} = -mg - \frac{3\mu_0 M_0 a^2}{2}\cdot\frac{I\cdot z}{\left(a^2 + z^2\right)^{5/2}} \qquad (5\text{-}21)$$

Next, although the ring is constrained not to move, we will get a current in it due to the changing magnetic field of the moving dipole. From Faraday's Law,

$$\oint \boldsymbol{E}\cdot d\boldsymbol{l} = IR = -\frac{d}{dt}\int \boldsymbol{B}\cdot d\boldsymbol{A} = -\frac{d}{dt}\int \boldsymbol{B}_{dip}\cdot d\boldsymbol{A} - L\frac{dI}{dt}, \qquad (5\text{-}22)$$

where R is the ring's resistance and L is its inductance. The flux through the ring is equal to the flux through a spherical cap centered at the ring's center, which is:

$$\int B_{dip}\cdot dA = \frac{\mu_0 M_0}{2}\frac{a^2}{\left(a^2 + z^2\right)^{3/2}} \qquad (5\text{-}23)$$

Therefore, the equation for the current in the ring is:

$$IR = -L\frac{dI}{dt} + \frac{3\mu_0 M_0 a^2}{2}\cdot\frac{z}{\left(a^2 + z^2\right)^{5/2}}\cdot\frac{dz}{dt} \qquad (5\text{-}24)$$

By introducing $v = dz/dt$, we can summarize the equations of motion with three first-order differential equations:

$$\left.\begin{aligned} \frac{d}{dt}z &= v \\ \frac{d}{dt}v &= -g - \frac{1}{m}F(z)I \\ \frac{d}{dt}I &= -\frac{R}{L}I + \frac{1}{L}F(z)v \end{aligned}\right\} \quad F(z) = \frac{3\mu_0 M_0 a^2}{2} \cdot \frac{z}{\left(a^2 + z^2\right)^{5/2}} \qquad (5\text{-}25)$$

Given the initial conditions, this set of equations can be integrated very accurately using a scheme such as the 4th order Runge-Kutta method.

5.2.4 RESULTS

In *Figure 5-2* below are two sequences of images from an animation of the dipole falling through the ring. Both the input textures and output images have been filtered substantially in order to enlarge the streamline features. It should also be noted that the time step is larger than normal in order to show enough of a change between successive images. The images represent a cross-section of the vector field through the origin with the $\hat{\boldsymbol{z}}$ axis pointing up and the $\hat{\boldsymbol{x}}$ axis pointing to the right. Thus, the dipole falls vertically down the center of the images, and the ring intersects the plane of the image at two points.

The left-hand column shows the dipole just before it goes through the ring. The flux through the ring due to the dipole varies more the closer it gets, and so a large current is induced in the ring. As a result, the magnetic field lines appear to be compressed as they "squeeze" through the ring, exerting an upward force on the dipole and a downward force on the ring. Although the correspondence between field lines is not always obvious when the images in time are presented vertically, the thick bright and dark bands (over-saturated intensities) can serve as references. Note especially the interesting evolution of the bright stripes near the current ring.

In the right-hand column we see the dipole below the ring, falling further down. Although it is accelerating due to gravity, again the flux change through the ring induces a current that slows down the falling dipole. We see this as the field lines stretching around and through ring, and as a consequence there is a downward force on the ring and the dipole is "pulled" up. Again, the field evolution is the most interesting in the bright bands around the ring.

It should be noted that animating the fields of this particular experiment was especially difficult and error prone. One reason is that the dipole represents a singularity that is difficult for the streamline integrator to detect automatically. In addition, notice that the magnetic field emanates vertically from the dipole, and that the dipole itself is moving vertically. This results in a horizontal motion for the particles below and above the dipole, instead of vertically with the

velocity of the dipole as we would expect. Indeed, this is a direct consequence of the fact that our definition of particle motion ignores any components along the direction of the magnetic field. Therefore, instead of moving with the dipole, the particles appear to flow around it. Worse, since streamlines near the center of the dipole depend on a small number of particles, movements in those particles produce large variations in the output intensity. One solution may be to introduce motion in $\boldsymbol{d}$ along the streamlines in $\boldsymbol{f}$ according to what we would expect, for example $\hat{\boldsymbol{B}}\left(\boldsymbol{v}\cdot\hat{\boldsymbol{B}}\right)$, but we have not done that for these images.

Magnet just above the ring *Magnet falling below ring*

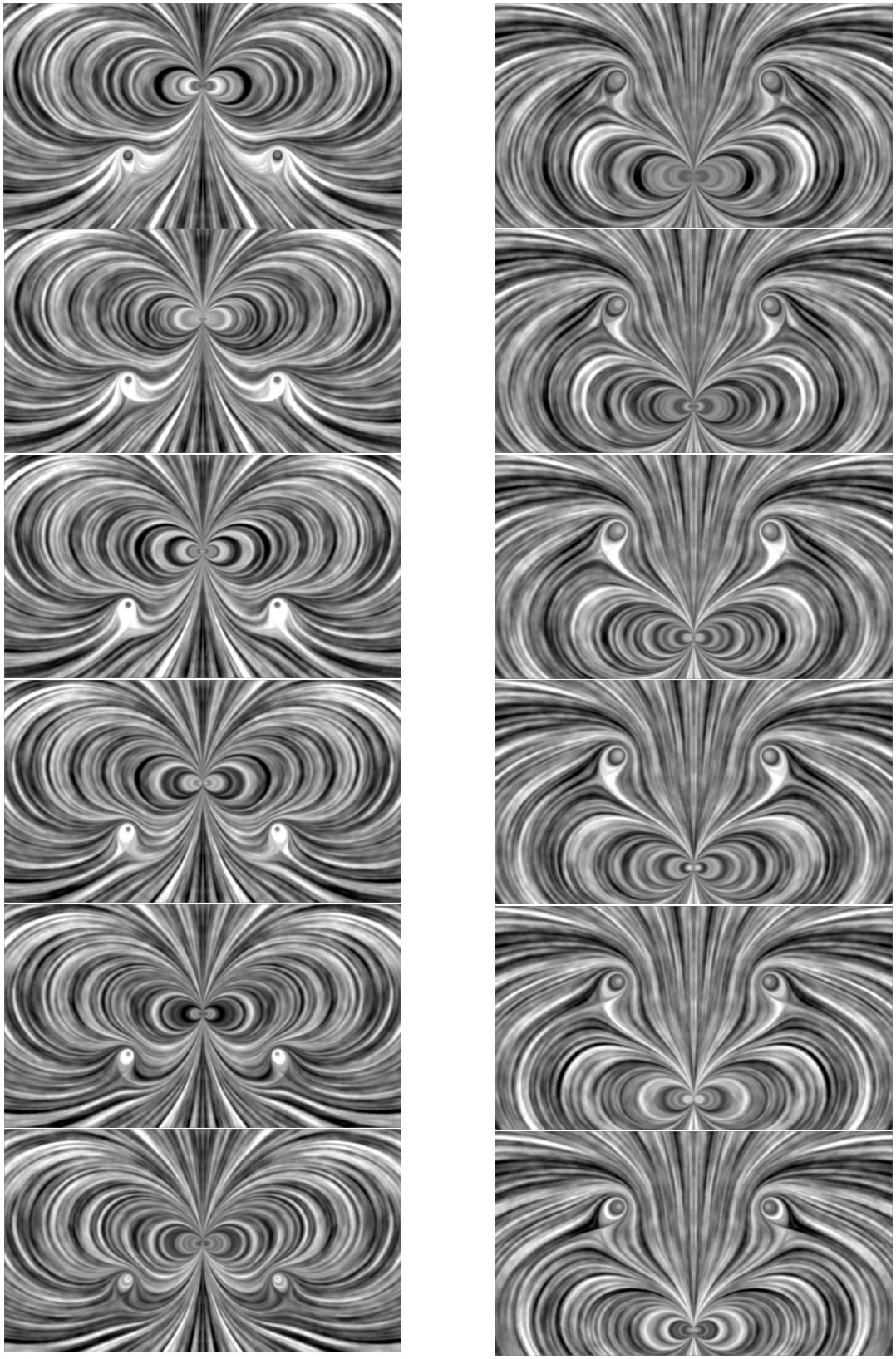

Figure 5-2: Animation sequences of the magnet falling through the conducting ring

5.3 RADIATING DIPOLE

Another experiment we were interested in visualizing is the radiation pattern of a time-varying electric dipole.

5.3.1 EXPERIMENTAL SETUP

In this experiment, we have a single electric dipole located at the origin with its dipole moment oriented along the $\hat{\boldsymbol{z}}$ axis, as in *Figure 5-3*. We allow the dipole to vary as any function in time:

$$\boldsymbol{p}(t) = p(t)\hat{\boldsymbol{z}}, \tag{5-26}$$

and we would like to see the resulting radiation pattern. Therefore, we have to adjust the time and distance scales so that the speed of light is visible.

5.3.2 COMPUTING THE FIELDS

To compute the fields, we need to be able to evaluate the dipole moment function at the retarded time, or the time that a signal traveling at the speed of light left the dipole in order to reach the point:

$$t_r = t - \frac{r}{c} \tag{5-27}$$

The electric field can be computed as:

$$\boldsymbol{E}(\boldsymbol{r}) = \frac{1}{4\pi\varepsilon_0}\left[\frac{3\hat{\boldsymbol{r}}(\boldsymbol{p}(t_r)\cdot\hat{\boldsymbol{r}}) - \boldsymbol{p}(t_r)}{r^3} + \frac{3\hat{\boldsymbol{r}}(\dot{\boldsymbol{p}}(t_r)\cdot\hat{\boldsymbol{r}}) - \dot{\boldsymbol{p}}(t_r)}{cr^2} + \frac{(\ddot{\boldsymbol{p}}(t_r)\times\hat{\boldsymbol{r}})\times\hat{\boldsymbol{r}}}{c^2 r}\right] \tag{5-28}$$

Similarly, assuming the dipole is stationary, the magnetic field is:

$$\boldsymbol{B}(\boldsymbol{r}) = \frac{1}{4\pi\varepsilon_0}\left[\frac{\hat{\boldsymbol{r}}\times\dot{\boldsymbol{p}}(t_r)}{c^2 r^2} + \frac{\hat{\boldsymbol{r}}\times\ddot{\boldsymbol{p}}(t_r)}{c^3 r}\right] \tag{5-29}$$

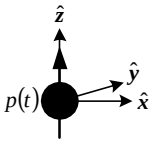

Figure 5-3: Electric dipole setup

In order to evaluate these, we need to be able to compute the first and second derivatives of $p(t)$.

For our experiment, we wanted to visualize an oscillating dipole:

$$p(t) = p_0 \sin \omega t, \quad \dot{p}(t) = p_0 \omega \cos \omega t, \quad \ddot{p}(t) = p_0 \omega^2 \sin \omega t \tag{5-30}$$

In addition, we wanted to observe the effects of switching a dipole on and off. The dipole switches on via the smooth differentiable function:

$$p(0 < t' < 1) = p_0 \left(6t'^5 - 15t'^4 + 10t'^3\right), \quad t' = \frac{t - t_{on}}{T_{on}} \tag{5-31}$$

where the dipole begins switching at t_{on} and has finished in time T_{on}. Similarly, the dipole switches off via the complementary function:

$$p(0 < t' < 1) = p_0 \left(1 - 6t'^5 - 15t'^4 + 10t'^3\right), \quad t' = \frac{t - t_{off}}{T_{off}} \tag{5-32}$$

5.3.3 RESULTS

Although the physical interpretation behind the particle "drift" model of field line motion breaks down when they are moving at or beyond the speed of light, the results of DLIC are still useful. The "drift velocity" still gives us the direction of the electromagnetic Poynting flux. For a radiating dipole, the result is a circle of electromagnetic waves expanding at the speed of light, as we expect.

In the left column of *Figure 5-4*, we see a cross-section of the field of an oscillating dipole located at the center with its moment pointing up in the plane of the image. The correspondence between the field lines of successive images is very clear in these images. The frequency of oscillation is high enough that the periodicity is clear in the image, and we can see waves being "pushed out" as a new domain emanates from the dipole. At the left and right edges of the image, we are far enough away from the dipole that the electromagnetic waves begin to look planar locally.

On the right in *Figure 5-4*, we see the electric field of the dipole switching on and then off. Areas of random white noise indicate a zero field. In the first three images, we see the expanding field of the dipole switching on. Except for the outer-most portion of the field, the pattern is the same as for a constant dipole. In the next three images, the dipole has switched off and we see an expanding circle of zero field. Note the interesting band just outside the zero field for which the dipole is actually switching.

Oscillating dipole *Dipole switching on and off*

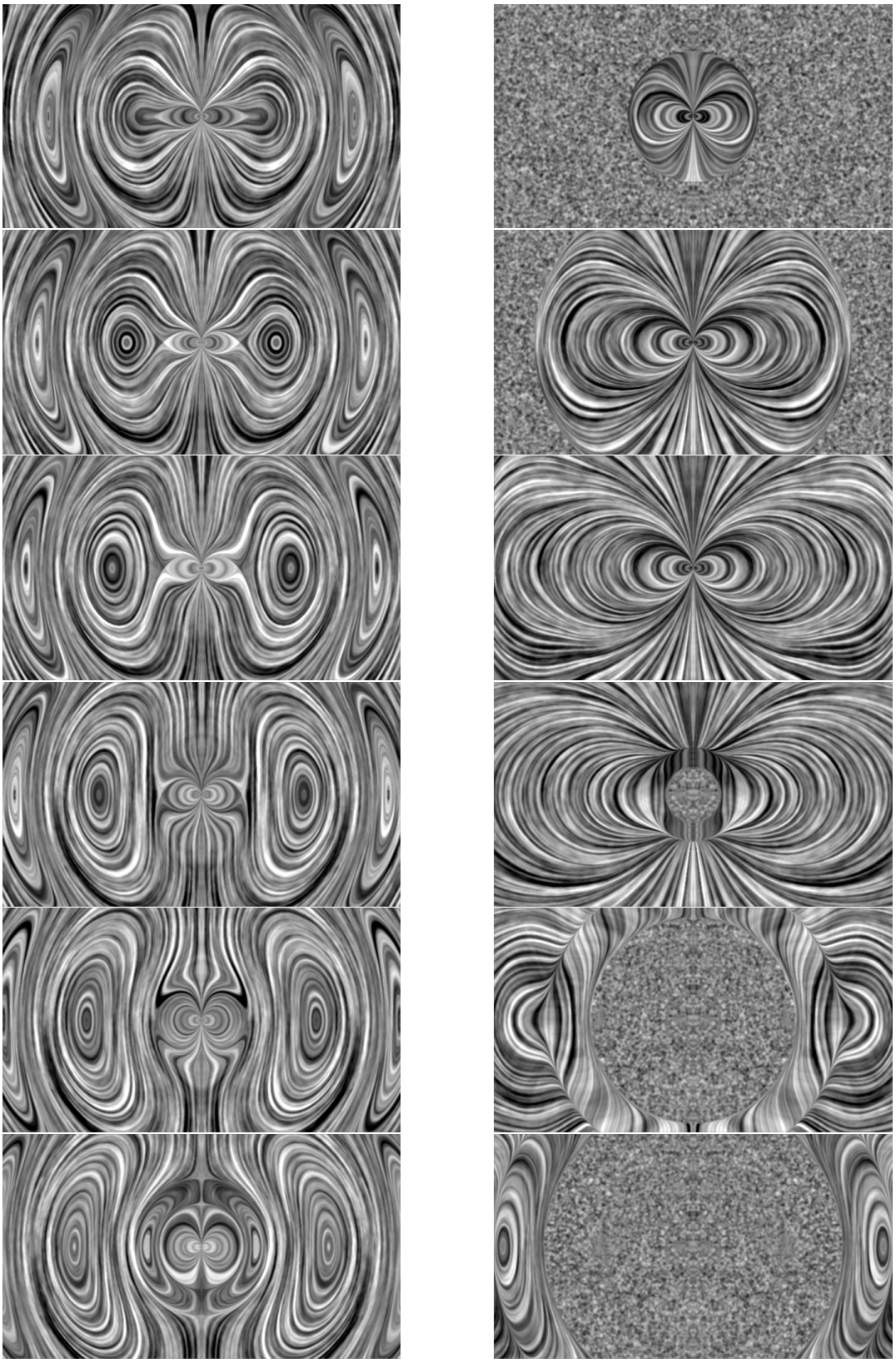

Figure 5-4: Expanding field waves of electric dipoles

6 CONCLUSIONS

6.1 SUMMARY

The animation sequences we generated for the two experiments demonstrate the power of DLIC in rendering dynamic vector fields. Provided that we can describe the dynamics of a time-varying vector field ***f*** via another motional vector field ***d***, DLIC produces a sequence of coherent images that communicate the evolution of the field. It does so by using ***d*** to advect a collection of random particles, generating an input texture to represent those particles, and finally passing it to LIC along with ***f*** to render the vector field images. In this thesis, I have presented a complete description of the details on how I implement DLIC, for example the texture generation and particle advection schemes.

6.2 FUTURE WORK

During the development of DLIC, several implementation choices needed to be made. Though this thesis attempts to discuss the algorithm in as much generality as possible, the full range of possible implementation variations has not been explored. For example, rather than using LIC and FLIC, perhaps some other texture-based visualization method would produce better results, particularly near singularities in the vector field. Or, there may be a more elegant and efficient way of rendering LIC-like images directly from the particles rather than sampling them to a texture first. Other methods for managing particles as they advect over time may produce higher-quality, more stable renderings. It may also be worthwhile to explore ways of representing the evolution of streamlines other than via the ***d*** field. In the example of the magnet falling through the ring, it does not seem to be the best way of describing motion near the dipole singularity. Finally, though it has not been our primary focus, speeding up the computation is highly desirable for lengthy animations. Being able to render these images in real-time would be an incredibly powerful tool for exploring vector fields.

7 ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor Professor John Belcher for his support. He brought the problem of visualizing dynamic electromagnetic fields to my attention, and his insights into the evolution of electromagnetic fields helped me devise the DLIC solution. The experiments and many of the concepts used in this thesis are due to him.

In addition, I would like to thank the Center for Educational Computing Initiatives for the use of some of its computing resources, and in particular I would like to thank Andrew McKinney and Mark Bessette for their continual support, advice, and patience.

Finally, I would like to thank Jennifer Hsieh for her patience and generosity during my UROP and thesis work.

8 REFERENCES

[1] J. Belcher and S. Olbert. “Field Line Motion in Classical Electromagnetism.” Submitted to the American Journal of Physics in 2001.

[2] J. Bryan. “Adaptive LIC Using a Curl Field and Stochastic LIC Approximation Using OpenGL.” Retrieved March 8, 2001 from the World Wide Web: *http://www.cis.ohio-statue.edu/~jbryan/School/cis888/*

[3] B. Cabral and C. Leedom. “Imaging Vector Fields Using Line Integral Convolution.” Proc. SIGGRAPH ’93, pp. 263-270, 1993.

[4] R. Crawfis and N. Max. “Texture Splats for 3D Scalar and Vector Field Visualization.” Proc. Visualization ’93, pp. 261-265, 1993.

[5] U. Diewald, T. Preußer, and M. Rumpf. “Anisotropic Diffusion in Vector Field Visualization on Euclidean Domains and Surfaces.” IEEE Transactions on Visualization and Computer Graphics, vol. 6, no.2, pp. 136-149, 2000.

[6] L. Forssell. “Visualizing Flow Over Curvilinear Grid Surfaces Using Line Integral Convolution.” Proc. Visualization ’94, pp. 240-247, 1994.

[7] L. Forssell and S.D. Cohen. "Using Line Integral Convolution for Flow Visualization: Curvilinear Grids, Variable-Speed Animation, and Unsteady Flows.” IEEE Trans. Visualization and Computer Graphics, vol. 1, no. 2, pp. 133-141, June 1995.

[8] W. Freeman, E. Adelson, D. Heeger. “Motion Without Movement.” ACM Computer Graphics, vol. 25, no. 4, pp. 27-30, 1991.

[9] D. Griffiths. Introduction to Electrodynamics, 3rd edition. Prentice Hall, 1999.

[10] P. Heckbert. “Filtering by Repeated Integration.” Proc. SIGGRAPH ’86, pp. 315-321, vol. 20, no. 4, 1986.

[11] B. Jobard and W. Lefer. “The Motion Map: Efficient Computation of Steady Flow Animations.” Proc. Visualization ’97, pp. 323-328, 1997.

[12] M.-H. Kiu, and D. Banks. “Multi-Frequency Noise for LIC.” Proc. Visualization ’96, pp. 121-126, 1996.

[13] W. de Leeuw and R. van Liere. "Comparing LIC and Spot Noise.” Center for Mathematics and Computer Science, CWI.

[14] W. de Leeuw and J. van Wijk. “Enhanced Spot noise for Vector Field Visualization.” Proc. Visualization ’95, pp. 233-239, 1995.

[15] N. Max and B. Becker. “Flow Visualization Using Moving Textures.” Proc. ICASE/LaRC Symposium on Visualizing Time Varying Data, Nov. 1995.

[16] H.-W. Shen and D. Kao. “UFLIC: A Line Integral Convolution for Visualizing Unsteady Flows.” Proc. Visualization ’97, pp. 317-322, 1997.

[17] H.-W. Shen and D. 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, 1998.

[18] D. Stalling and H.-C. Hege. “Fast and Resolution Independent Line Integral Convolution.” Proc. SIGGRAPH ’95, pp. 249-256, 1995.

[19] D. Stalling. “Fast Texture-Based Algorithms for Vector Field Visualization.” Doctoral dissertation at the Department of Scientific Visualization at the Konrad-Zuse-Zentrum Berlin (ZIB), 1998.

[20] V. Verma, D. Kao, and A. Pang. “PLIC: Bridging the Gap Between Streamlines and LIC.” Proc. Visualizaton ‘99, pp.341-348, 1999.

[21] R. Wegenkittl, E. Gröller, and W. Purgathofer. “Animating Flowfields: Rendering of Oriented Line Integral Convolution.” Proc. Computer Animation ’97, pp. 15-21, IEEE Computer Society, June 1997.

[22] R. Wegenkittl and E. Gröller. “Fast Oriented Line Integral Convolution for Vector Field Visualization via the Internet.” Proc. Visualization ’97, pp.309-316, 1997.