Lecture notes: Visualization I
Visualization of vector fields using Line Integral Convolution and volume rendering

Anders Helgeland

FFI

Chapter 1

Visualization techniques for vector fields

Vector fields play an important role in science and engineering. They allow us to describe a wide variety of phenomena like fluid flow and electromagnetic fields. Large vector fields often exhibit quite complex structures, which can be difficult to reveal. Making an efficient visualization of a vector field is one of the current challenges in scientific visualization.

1.1 Hedgehogs and glyphs

A natural vector visualization technique is to draw an oriented, scaled line for each vector. The line is drawn, starting at a grid point and is oriented in the direction of the vector components associated with that point. The color and length of each line can be set by the vector magnitude. This technique is often referred to as a *hedgehog* or oriented lines. To get a better impression of the direction of the vector field, arrowheads can be added to the lines. Any 2D or 3D geometric representation indicating vector magnitude and direction is called a *glyph* (see figure 1.1).

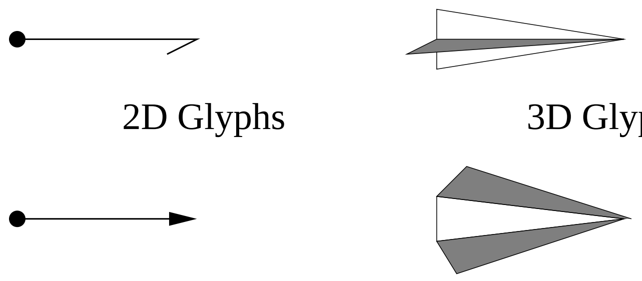

Figure 1.1: Glyphs.

These techniques are best suited for small data sets. If the placements of the glyphs are too dense and the variations in magnitude are too big, the images tends to be “cluttered” and visually confusing.

The results can be improved if some form of *thresholding* is applied. One example which can remove some of the clutter is to neglect the drawing of glyphs where the length of the vector is below a certain value, $\|v\| < c$. The threshold τ is typically a normalized quantity in the range $[0, 1]$. If $\tau = 0$, every vector is displayed. If $\tau = 1$, only the vectors with the largest magnitude are present in the resulting image. Another method is to scale the vectors so that the overlapping of the glyphs are reduced. In figure 1.2, we have used threshold and scale to emphasize regions where the information of the vector field is important. We see from the bottom image that suppressing a larger number of the least significant vectors may show relevant physical information more clearly.

1.2 Curve representation

A better way of representing the vector fields is to draw curves that reveal the orientation and structure of the field. The lines can be colored according to vector magnitude, but also other scalar quantities such as temperature or pressure may be used to color the lines. The computation of path lines and streak lines strongly depends on the capabilities of the underlying hardware. Both these techniques are time dependent, and vector data for multiple time steps have to be stored in the computer during the calculations. The requirements in memory can quickly be of many gigabytes, and not all computers are big enough to handle that.

A possible problem concerning the rendering of field lines is the spatial perception of the objects in the scene. On common graphics workstations, field lines and other curves are displayed using flat shaded line segments, impairing the spatial impression of the image [1]. Phong type shading models [2] are traditionally applied to surface elements, but can be generalized to line primitives in $\mathbb{R}^3$ [1]. Such generalizations have been used to render fur or human hair. However, on current graphics workstations, there is no direct hardware support for the display of illuminated line primitives [1]. Therefore major parts of the illumination calculations have to be performed in software. In 1997 Stalling, Zöckler and Hege [1] presented a method to achieve fast and accurate line illumination, by exploiting texture mapping capabilities of modern graphics hardware. This shading technique allows the visualization of large numbers of field lines in a vector field [1].

Other ways to enhance the three-dimensional impression of the vector field are to represent the field lines by polygonal objects, for example like tubes. One of these techniques is called *streamribbons*. A streamribbon can be constructed by generating two adjacent field lines and then bridging the lines with a polygonal mesh. This technique works well as long as the field lines remain relatively close to another [2]. If the field lines diverge, the resulting ribbons will not accurately depict the vector field, because we expect the surface of a ribbon to be everywhere tangent to the vector field (i.e., definition of field line).

A *streamsurface* is a collection of an infinite number of field lines passing through a curve. The curve defines the starting points for the field lines and if the curve is closed, as in a circle, the surface is closed and we get a *streamtube*. Streamsurfaces can be computed by generating a set of field lines from selected points on the curve. A polygonal mesh is then constructed by connecting adjacent field lines. Like in streamribbons the separation of the field lines can introduce large errors into the surface.

A problem with all these techniques, with the exception of the one proposed by Stalling, Zöckler and Hege [1][1], is the limitation of the number of field lines that can be displayed in the scene, without cluttering the image. This makes the visualization dependent on the choice of seed points. As mentioned before, it is not obvious how to distribute the field lines in space without missing important details of the field. In figure 1.3, the image is a little cluttered because

[1]The *Fast display of illuminated field lines* method, allows the generation of images with thousands of fi eld lines at interactive rate [1]. This means that the positioning of an individual fi eld line becomes less important.

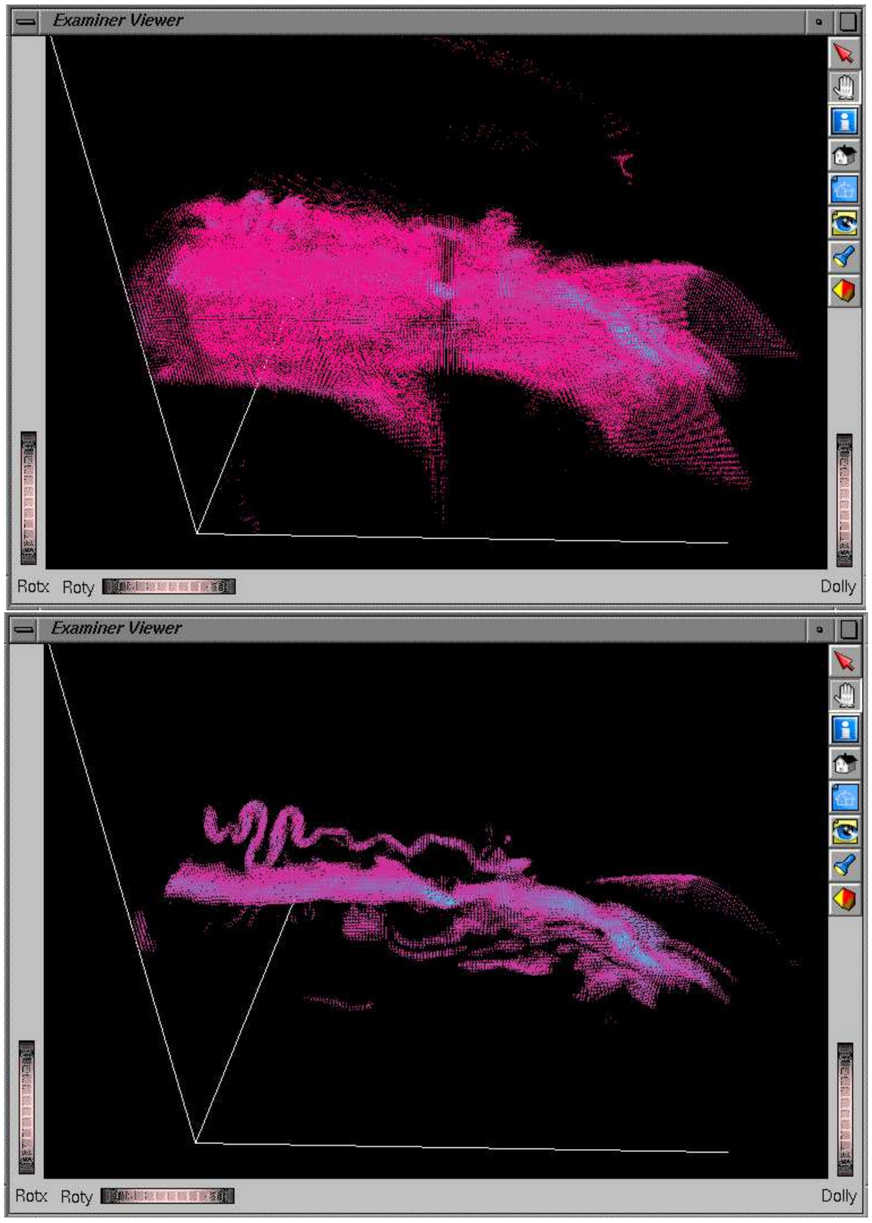

Figure 1.2: Visualization of a vector field using glyphs. In the top image we have set the threshold $\tau = 0.01$ and the scale $s = 4 * \Delta h$, where Δh is the largest of the grid spacings Δx, Δy and Δz. In the bottom image $\tau = 0.17$ and $s = 4 * \Delta h$. The value s determine the length of the largest glyphs.

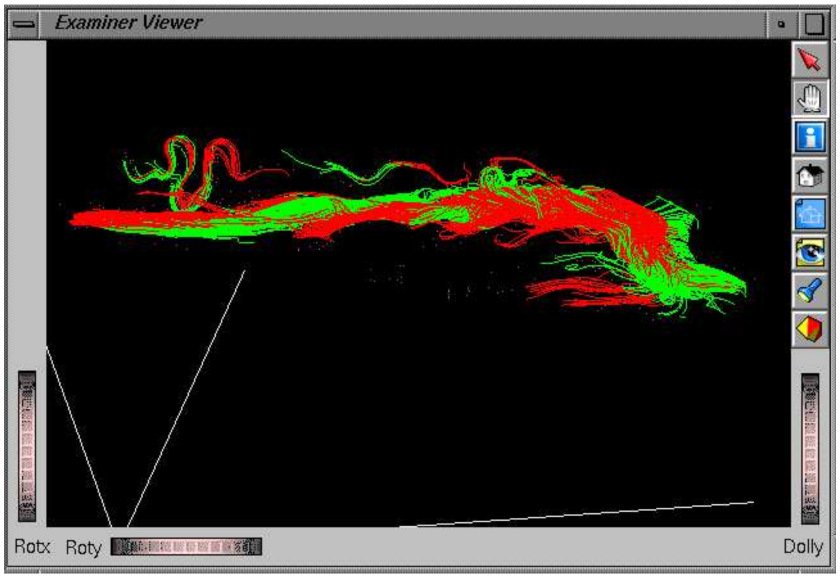

Figure 1.3: Visualization of a vector field using field lines. The red lines are at the “downstream” side of the seed point whereas the green ones are at the “’upstream’ side.

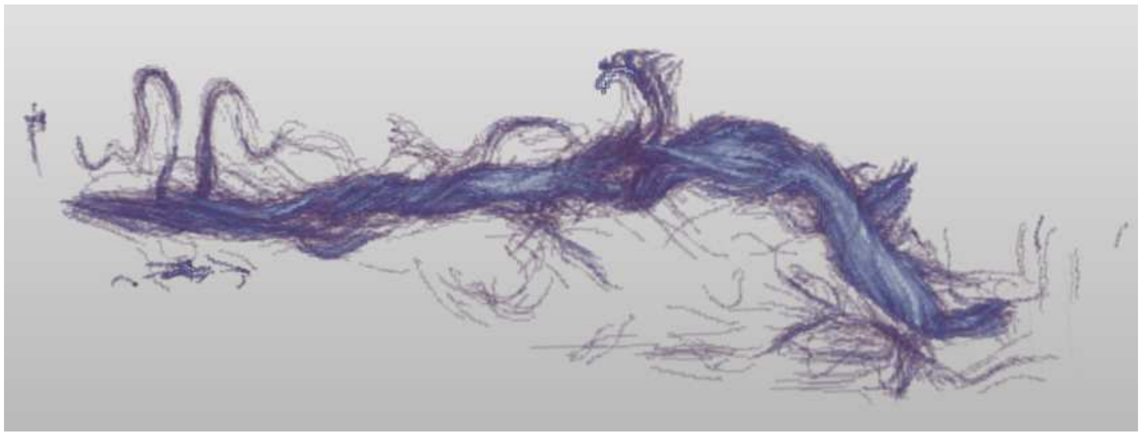

Figure 1.4: Visualization of a vector field using Line Integral Convolution.

of the large number of field lines rendered in the vector field. As in figure 1.2, we have focused on a *region of interest* by thresholding the distribution of seed points.

1.3 Texture based techniques

The use of texture based techniques is an alternative method for visualizing vector fields. Examples of these techniques are spot noise [3], [4] and Line Integral Convolution [5], [6], [7]. These techniques avoid some of the problems with vector visualization discussed in sections 1.1 and 1.2. Figure 1.4 shows the result after applying LIC on the same vector field as visualized with other techniques in the figures 1.2 and 1.3. The vector field is obtained from [8].

Chapter 2

Line Integral Convolution

2.1 Introduction to Line Integral Convolution

Line Integral Convolution (LIC) is a powerful technique used to represent vector fields with high accuracy. It is a texture based technique that can be used to display both two- and three-dimensional fields. LIC is essentially a filtering technique that blurs a texture locally along a given vector field, causes voxel intensities to be highly correlated along the field lines but independent in directions perpendicular to them. It takes a pixel/voxel set and a vector field as inputs and produces a new pixel/voxel set as output, see figure 2.1.

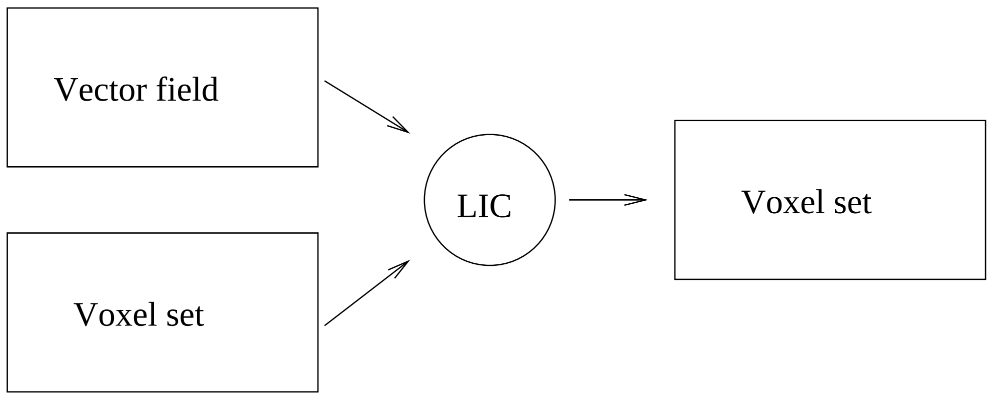

Figure 2.1: A vector field and a voxel set are inputs to the Line Integral Convolution resulting in a new voxel set.

Since introduced in 1993 by Cabral and Leedom [5], Line Integral Convolution has been an active field of research within the computer graphics and visualization community. Several researchers have developed the LIC algorithm further and the method has found many application areas, ranging from computer art to scientific visualization. Two examples of LIC images are shown in figure 2.2.

2.2 Convolution

Convolution is a mathematical definition that can be applied to several areas, such as image processing, optics and signal processing. The convolution of two real functions $f = f(x)$ and

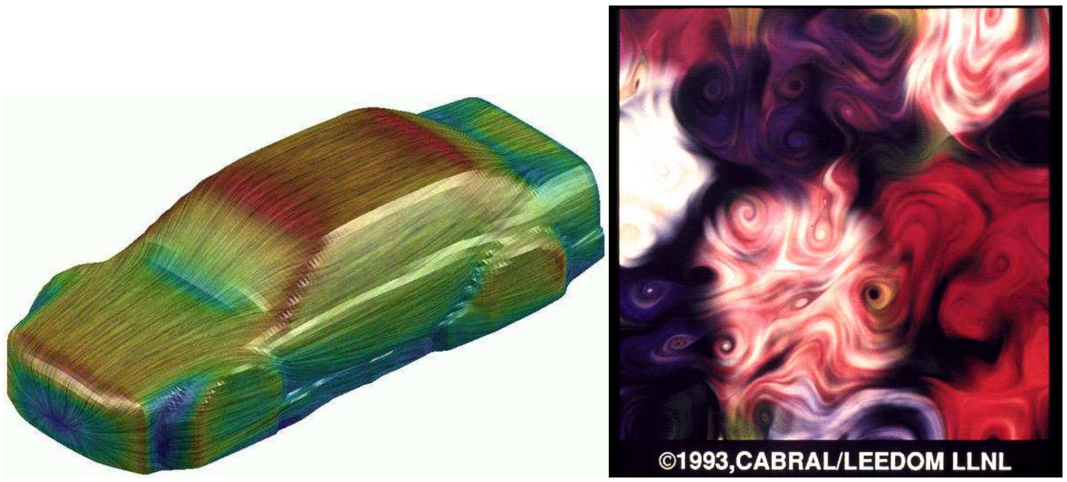

Figure 2.2: Examples of LIC images. The image on the left depicts the computed velocity field close to a racing car computed at the *Italian Aerospace Research Center* (CIRA). The image on the right is a picture of flowers convolved by a given 2D vector field taken from [5].

$g = g(x)$ is defined as

$$I(x) = f(x) * g(x) = g(x) * f(x) = \int_{-\infty}^{\infty} f(y)g(x-y)dy. \tag{2.1}$$

If we convolve $f(x)$ with the *Dirac delta* function

$$\delta(x) = \begin{cases} 0 & (x \neq y), \\ \infty & (x = y), \end{cases} \tag{2.2}$$

we obtain

$$f(x) * \delta(x) = \int_{-\infty}^{\infty} f(y)\delta(x-y)dy = f(x). \tag{2.3}$$

In figure 2.3, we see how a convolution with a “box” function leads to a smearing of the function f.

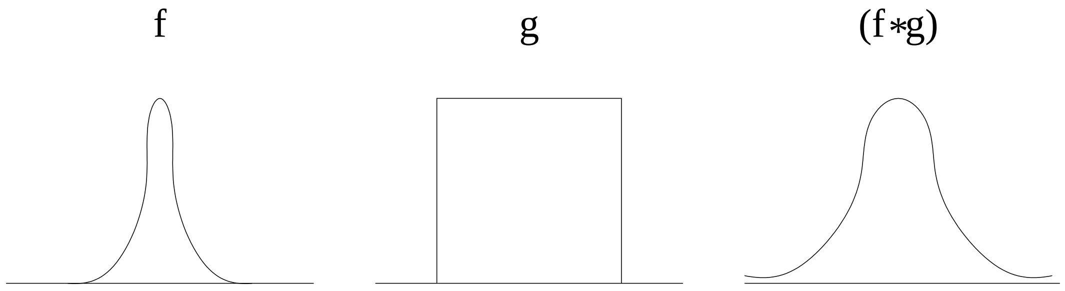

Figure 2.3: Convolution of the function f with a “box” function g.

Convolution is commonly used in image processing. The convolution is then typically represented by a two-dimensional convolution matrix A, where the matrix elements describe the blurring effect applied to the image. The intensity of a pixel $I(p_{ij})$ in the new image is found by adding intensity values from “neighboring” pixels in the original image times the matrix element matching the position to the pixels. If for example a picture is convolved by the 3×3 matrix

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{2.4}$$

the only pixel to contribute in the finding of $I(p_{ij})$, is p_{ij} itself. The result after such a convolution is the original image. The intensity of a pixel after convolution can be found by

$$I_{ij} = n \sum_{k=1}^{K} \sum_{l=1}^{L} A_{kl} I_{i+k-K/2, j+l-L/2}, \tag{2.5}$$

where $I_{ij} = I(p_{ij})$ and n is a normalization constant. Figure 2.4 demonstrates the effect of convolving an image by a 7×7 matrix. Notice the blurring effect in the right image.

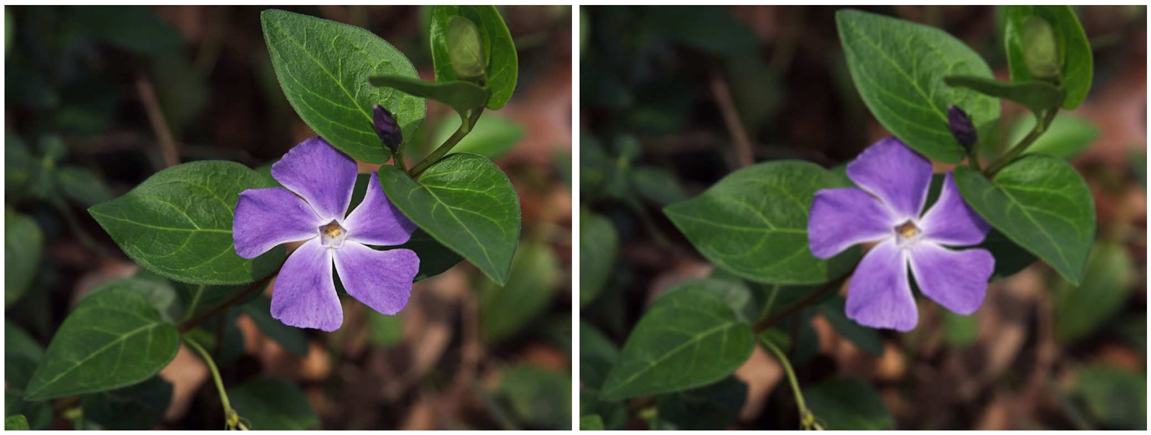
Figure 2.4: Blurring of a picture.

2.3 Convolution along a vector

Line Integral Convolution is a modification of a technique called *DDA convolution* [5]. In this method, each vector in a field is used to compute a DDA line which is oriented along the vector and going in the positive and negative vector direction some distance L. A convolution is then applied to the texture along the DDA line. The input texture pixels under the convolution kernel are summed, normalized by the length of the convolution kernel, $2L$, and placed in an output pixel image for the vector position. Figure 2.5 illustrates this operation for a single vector in the field.

The DDA approach depicts the vector field inaccurately. It assumes that the local vector field can be approximated by a straight line. As a result, DDA convolution gives an uneven rendering, treating linear portions of the field more accurately than areas with high curvature, such as areas with small eddies or vortices. This becomes a problem in visualization of vector fields, since details in the small scale structure are lost. Line Integral Convolution solves some of this problem, as the convolution takes place along curved segments.

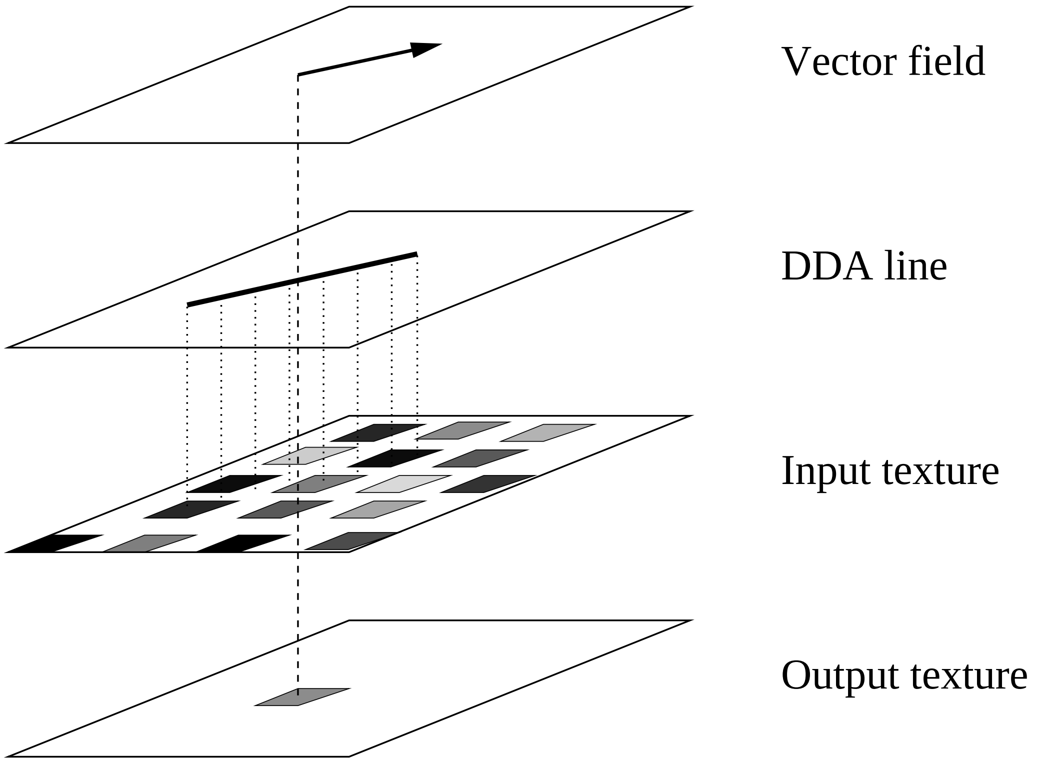

Figure 2.5: Convolution along a vector. The pixel in the output texture is a weighted average of all the input texture pixels covered by the DDA line.

2.4 LIC

For a given vector field $\boldsymbol{F} : \mathbb{R}^3 \to \mathbb{R}^3$, the idea of Line Integral Convolution is to blur an input texture along field lines of $\boldsymbol{F}$. The LIC algorithm carries out the blurring by applying an one-dimensional convolution throughout the input texture. Each voxel in the output texture is determined by the convolution kernel and the texture voxels along the local field line indicated by the vector field. As a result, the intensity values of the output scalar field are strongly correlated along the field lines, whereas perpendicular to them almost no correlations appear. LIC images can therefore provide a clear visual impression of the directional structure of $\boldsymbol{F}$. This is illustrated in figure 2.6.

Given a field line $\boldsymbol{\sigma}$, Line Integral Convolution can mathematically be described by

$$I(\mathbf{x}_0) = \int_{s_o - L}^{s_0 + L} k(s - s_0)\, T(\boldsymbol{\sigma}(s))\, ds, \tag{2.6}$$

where $I(\boldsymbol{x}_0)$ is the intensity for a voxel located at $\mathbf{x}_0 = \boldsymbol{\sigma}(s_0)$. In this equation k denotes the filter kernel of length $2L$ and T denotes the input texture. The curve $\boldsymbol{\sigma}(s)$ is parameterized by the arc-length s. The filter length or the *convolution length* determine how much the texture is smeared in the direction of the vector field. With L equal to zero, the input texture is passed through unchanged. As the value of L increases, the output texture is blurred to a greater extent. Stalling and Hege [6] found good results by choosing the convolution length $2L$ to be $1/10$th of the image width.

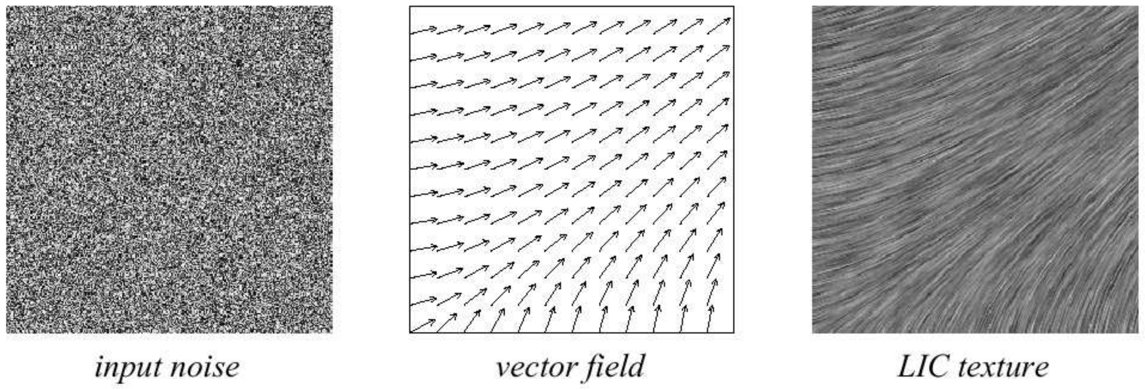

Figure 2.6: A 2D example where line integral convolution is applied to a white noise input texture. We see how the input texture is blurred along the field lines of the vector field. The images are taken from [9].

In the algorithm (for 2D) proposed by Cabral and Leedom [5], referred to as *CL-LIC* hereafter, computation of field lines were done by a variable step Euler's method. The local behavior of the vector field is approximated by computing a local field line that starts at the center of a pixel (x, y) and moves out in the "downstream" and "upstream" directions,

$$\begin{aligned} P_0 &= (x + 0.5, y + 0.5), \\ P_i &= P_{i-1} + \frac{V(P_{i-1})}{\|V(P_{i-1})\|}\Delta s_{i-1}, \\ \\ P'_0 &= P_0, \\ P'_i &= P'_{i-1} - \frac{V(P'_{i-1})}{\|V(P'_{i-1})\|}\Delta s'_{i-1}, \end{aligned} \tag{2.7}$$

The convolution is expressed as follows,

$$\begin{aligned} F_{out}(x, y) &= \frac{\sum_{i=0}^{l} F_{in}(P_i)h_i + \sum_{i=0}^{l'} F_{in}(P'_i)h'_i}{\sum_{i=0}^{l} h_i + \sum_{i=0}^{l'} h'_i}, \\ h_i &= \int_{s_i}^{s_i+\Delta s_i} k(w)\, dw, \end{aligned}$$

where

- $V(P_i)$ is the vector from the input vector field at the point P_i.
- $F_{out}(x, y)$ is the output pixel value at point (x, y).
- $F_{in}(P_i)$ is the input pixel value at point P_i.
- l and l' are the convolution distances along the positive and negative directions, respectively.

- P_i represent the i*th* cell the field line steps in the positive direction, and P_i' represent the i*th* cell in the negative direction.
- $k(w)$ is the convolution filter function.
- Δs_i is the arc length between the point s_i and s_{i+1} along the field line.
- $s_0 = 0$

This is done for each pixel, eventually making an output LIC image.

2.5 Fast LIC

The algorithm suggested by Cabral and Leedom [5] is very compute intensive. Even in 2D, the algorithm involves a large number of arithmetic operations and can be rather slow. In 1995 Stalling and Hege [6] proposed a fast and more accurate LIC algorithm. In the LIC algorithm proposed by Cabral and Leedom, for each pixel in the output image, a separate field line segment and a separate convolution integral are computed. Stalling and Hege points out two types of redundancies in this approach. First, a single field line usually covers lots of image pixels. Therefore in *CL-LIC* large parts of a field line are recomputed very frequently. Second, for a constant filter kernel k very similar convolution integrals occur for pixels covered by the same field line. This is not utilized by Cabral and Leedom's algorithm. Consider two points located on the same field line, $\mathbf{x}_1 = \boldsymbol{\sigma}(s_1)$ and $\mathbf{x}_2 = \boldsymbol{\sigma}(s_2)$. Assume, that the points are separated by a small distance $\Delta s = s_2 - s_1$. Then for a constant filter kernel k the convolution integral (2.6) for $\mathbf{x}_2$ can be written as

$$I(\mathbf{x}_2) = I(\mathbf{x}_1) - k\int_{s_1-L}^{s_1-L+\Delta s} T(\boldsymbol{\sigma}(s))\,ds + k\int_{s_1+L}^{s_1+L+\Delta s} T(\boldsymbol{\sigma}(s))\,ds. \tag{2.8}$$

The intensities differ by only two small correction terms that are rapidly computed by a numerical integrator. By calculating long field line segments that cover many pixels and by restricting to a constant filter kernel, we avoid both types of redundancies being present in CL-LIC. The length of the field line or the *field line length* is typically larger than the convolution length. In designing the fast-LIC algorithm, Stalling and Hege suggest an approach which relies on computing the convolution integral by sampling the input texture T at evenly spaced locations $\mathbf{x}_i$ along a pre-computed field line $\boldsymbol{\sigma}(s)$. First a field line is computed for some location $\mathbf{x}_0 = \boldsymbol{\sigma}(s_0)$ (see figure 2.7). The convolution integral (2.6) for this location is approximated as

$$I(\mathbf{x}_0) = k\sum_{i=-n}^{n} T(\mathbf{x}_i), \tag{2.9}$$

with $\mathbf{x}_i = \boldsymbol{\sigma}(s_0 + ih_t)$, where h_t is the distance between different sample points. To ensure normalization we set $k = 1/(2n+1)$. After having computed $I(\mathbf{x}_0)$, we step in both directions along the current field line, updating the convolution as follows

$$\begin{aligned} I(\mathbf{x}_{m+1}) &= I(\mathbf{x}_m) + k\big[T(\mathbf{x}_{m+1+n}) - T(\mathbf{x}_{m-n})\big], \quad m = 0, 1, \ldots, M \\ I(\mathbf{x}_{m-1}) &= I(\mathbf{x}_m) + k\big[T(\mathbf{x}_{m-1-n}) - T(\mathbf{x}_{m+n})\big]. \quad m = 0, -1, \ldots, -M. \end{aligned} \tag{2.10}$$

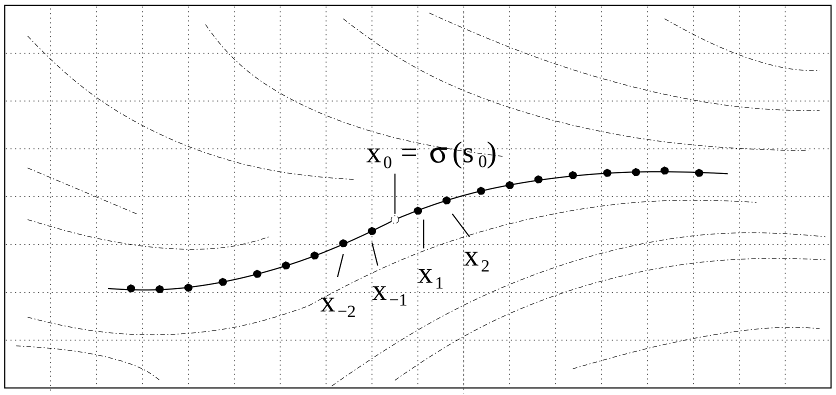

Figure 2.7: The input texture is sampled at evenly spaced locations $\mathbf{x}_i$ along a field line $\boldsymbol{\sigma}$. For each location the convolution integral $I(\mathbf{x}_i)$ is added to the pixel or voxel in 3D containing $\mathbf{x}_i$. A new field line is computed only for those pixels or voxels where the number of samples does not already exceed a user-defined limit.

For each sample point the corresponding output image pixel is determined and the current intensity is added to that pixel. In this way, we efficiently obtain intensities for many pixels covered by the same field line. Running through all output image pixels, the algorithm requires the total number of hits already occurred in each pixel to be larger than some minimum. If the number of hits in a pixel is smaller then the minimum, a new field line is computed. Otherwise that pixel is skipped. At the end, accumulated intensities for all pixels has to be normalized against the number of hits. The algorithm referred to as *fast-LIC* can be described by the pseudocode presented in figure 2.8.

```
for each pixel p
    if (numHits(p) < minNumHits) then
        initiate fi eld line computation with $\mathbf{x}_0$ = center of p
        Compute convolution $I(\mathbf{x}_0)$
        add result to pixel
        set m=1
        while m < some limit M
            Update convolution $I(\mathbf{x}_m)$ and $I(\mathbf{x}_{-m})$
            add result to pixels containing $\mathbf{x}_m$ and $\mathbf{x}_{-m}$
            set $m = m + 1$
for each pixel p
    normalize intensity according to numHits(p)
```

Figure 2.8: Pseudocode of fast-LIC.

Accuracy is especially important in fast-LIC because multiple field lines determine the intensity of a single pixel. If these lines are incorrectly computed, the LIC pattern gets disturbed. This is most evident near the center of a vortex in the vector field. The LIC-algorithm proposed by Cabral and Leedom, used a variable step Euler's method in the computation of field lines. Stalling and Hege [6] employ a fourth-order Runge-Kutta method, thus making the algorithm more accurate.

If the step size between the sample points is too big, we may miss some of the pixels (voxels in 3D) along the computed field line. This can lead to images with aliasing [2] problems. Stalling and Hege have found a step size of $h_t = 0.5$ times the width of a texture cell to be sufficient.

2.6 Some improvements

After the first LIC-algorithm was introduced in 1993, a number of suggested improvements have been made. In 1994, Forsell [10] describes an extension that makes it possible to map flat LIC images onto curvelinear surfaces. So far the algorithm only worked for vector fields over regular two-dimensional Cartesian grids. In 1995, Stalling and Hege [6] proposed the fast-LIC algorithm discussed in section 2.5. Shen, Johnson and Ma [11] introduced in 1996 a technique for injecting dye into the LIC field to highlight the flow field's local feature. The dye insertion method utilizes the LIC's natural "smearing" to simulate advection of dye within the flow field. The simulation of dye injection is done by assigning colors to isolated local regions in the input white noise texture. Cells whose streamline pass through such regions receive color contributions from the dye.

In 1997, Wegenkittl, Gröller and Purgathofer [12] presented *Oriented Line Integral Convolution* (OLIC), where also the information about the orientation of the vector field is present in the resulting image. And in 1998, Interrante and Grosch [7] looked at some techniques for visualizing 3D flow through a volume.

Chapter 3

Volume LIC

Although Line Integral Convolution is most commonly used to depict 2D flows, or flows over a surface in 3D, LIC methods can equivalently be used to depict 3D flows through a volume [7]. When LIC is applied to a solid noise texture, the output is a solid LIC texture that is blurred along the directions of the vector field. For 2D vector fields and surfaces in 3D this works well, because the resulting LIC texture are two-dimensional. But when working with volumetric data, it can be difficult to get a good impression of the vector field from a series of solid or partially opaque 2D slices rendered via direct volume rendering (see figure 3.1). The image of the vector field will be incomplete and the inner details are completely lost.

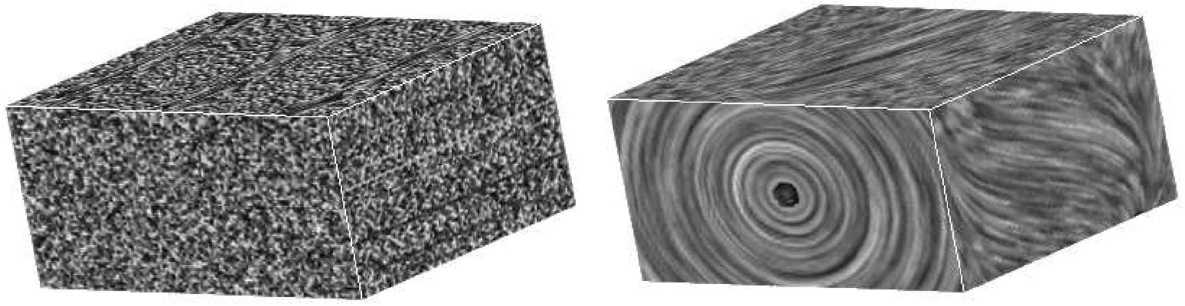

Figure 3.1: Left: A solid white noise input 3D texture. Right: The output texture after LIC.

3.1 Choice of input texture

3.1.1 Region Of Interest

When working with scientific data, we can make use of scalar values like temperature and absolute value of velocity to specify a critical region in the volume where the information of the vector field is especially important [7]. Hence, the presentation of the data can be clarified by isolating and emphasizing information in these critical regions. Interrante and Grosch [7] found that when LIC is used together with a Region Of Interest (ROI), better results can be achieved if the ROI mask is applied as a preprocess to the input texture, before the Line Integral Convolution, rather than as a postprocess to the output afterwards. In the first case, in which the

ROI mask is applied before LIC, the Region Of Interest mask is guided by the flow itself, with the result that the boundaries of the ROI will be everywhere aligned with the direction of the vector field. In the second case, the visible portion of the vector field in the LIC texture will be completely determined by ROI mask, making boundaries which will not in general follow the direction of the flow. Figure 3.2 shows the result after applying LIC to an input texture that has been masked by a Region Of Interest. The visualized vector field is a vorticity field obtained from a simulation done at FFI [8]. The vorticity magnitude was used to specify the ROI mask. The textures were defined to be twice as large as the vector field $(594 \times 394 \times 194)$, so that the details could be seen more easily.

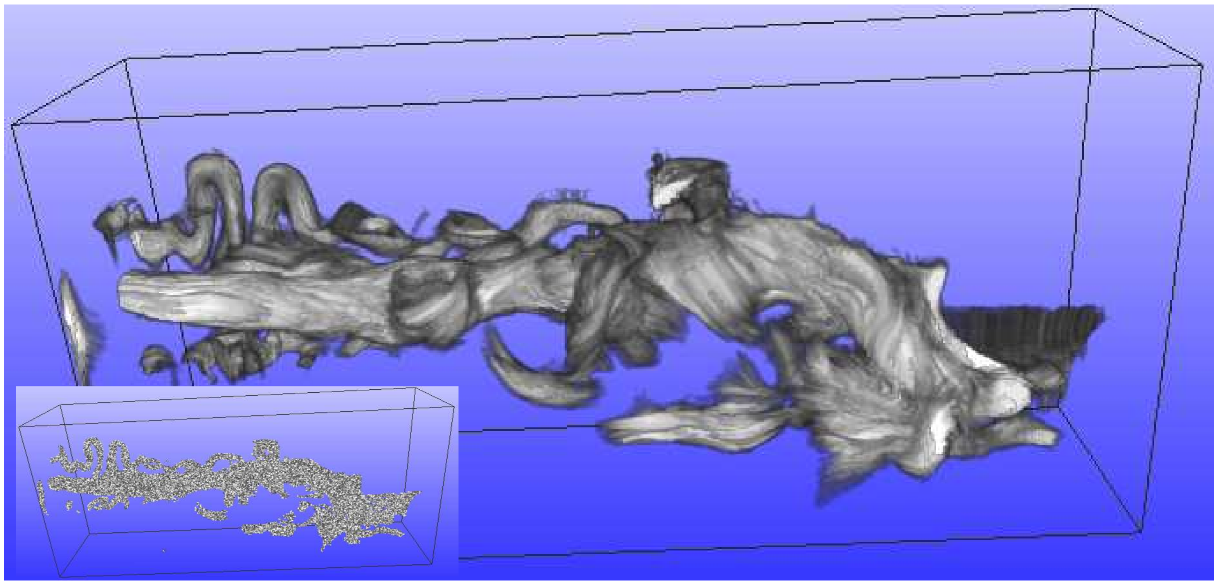

Figure 3.2: Left: The masked input texture. Right: The resulting LIC texture.

3.1.2 Sparse input texture

When Line Integral Convolution is applied to a solid noise texture, even one that has been masked by a Region Of Interest function, the output image looks more or less like a solid "object". The details of the vector field can still be difficult to depict. Applying LIC over an input texture consisting of a sparse set of points([7] , [13]) produces an output image which gives a much better impression of the vector field. Instead of a solid object it now produce a collection of densely placed field lines.

One of the strengths with Line Integral Convolution applied to dense (white noise) input textures, is that it is not dependent on the choice of seed points. When LIC is applied to a sparse input texture though, this is not the case. The LIC texture is then computed by generating strokes through the volume by advecting the distributed points in the input texture with the empty space between them. As a result, the output texture is dependent on the placements of the distributed points. However, since texture based techniques allow the display of a much larger number of lines simultaneously in an image, making the position of each stroke less important, statistical methods for distributing the points in the volume can be applied.

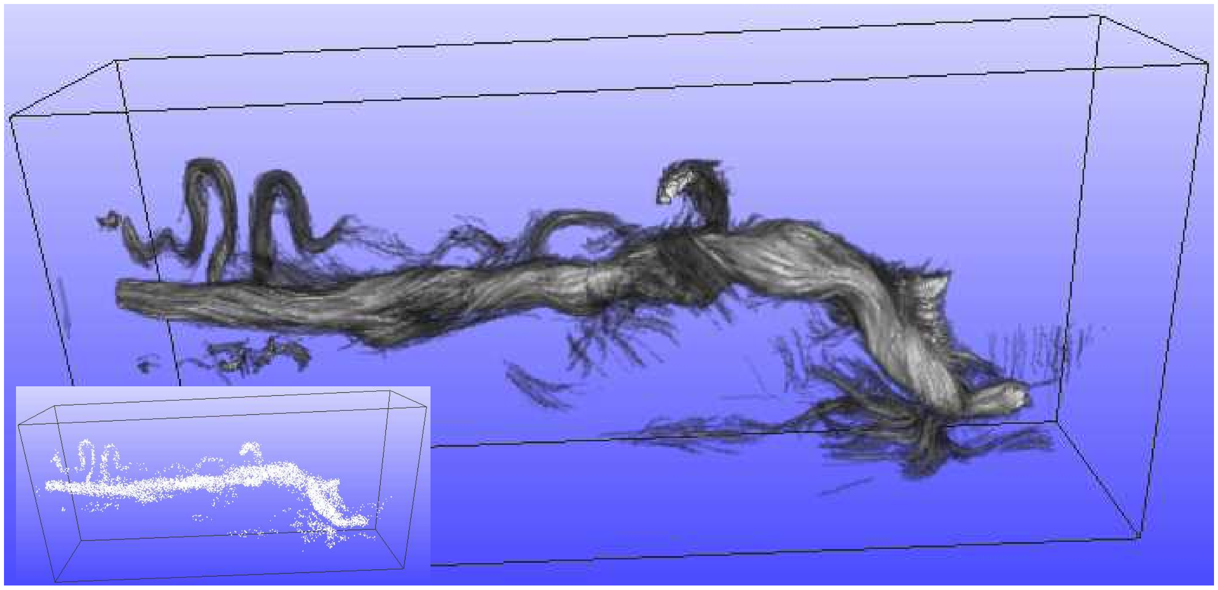

Figure 3.3: The input texture where 9514 points are distributed after the vorticity magnitude and the resulting LIC texture.

We have tried different approaches in distributing the points or voxels in the 3D texture. In the first approach the idea was to make a texture where the points was distributed after the scalar value that was used in making the ROI mask. Hence, we get output images where regions with high scalar values are more emphasized than other regions with lower values. In this approach, the regions with the highest scalar values becomes more cluttered than the regions with lower scalar values. Another option is a more random approach. This method leads to a LIC texture where the field lines are more evenly distributed and with some datasets it can give a better impression of the vector field. Figures (3.3, 3.4, 3.5) show some examples of Line Integral Convolution applied to input textures with different distribution functions. In figure 3.3, the points in the input texture are distributed according to the vorticity magnitude. While in the figures 3.4 and 3.5, a random approach is used. The number of points or spots in the input texture in figures 3.3 and 3.4, are about 9500. In figure 3.5, about 20000 spots are used.

The algorithm for computing a random input texture can be described by the pseudocode in figure 3.6.

The density of the distributed points in the input texture is determined by the *density factor*. The final set of points chosen are set to 255. The rest of the voxels are set to zero. To differentiate the strokes in the output texture, the use of white noise data has been common when applying LIC to a dense input texture. When applying LIC to a sparse input texture though, the use of various level of grey is not necessary. Instead, we differentiate the individual field lines by employing a shading technique called limb darkening. This will be discussed in 4.3.

Best results were achieved when requiring a minimum distance between the selected points in the input texture. This prevents the spots in the input texture and thus the field lines in the output image from getting too close. In this approach, the details of the vector field are displayed more clearly. To prevent the lines from getting too close, ideally, the distribution of the field lines itself should be controlled, rather than the distribution of points [1]. However, by limiting the total length of the field line and if the field lines are integrated an equal distance in upstream

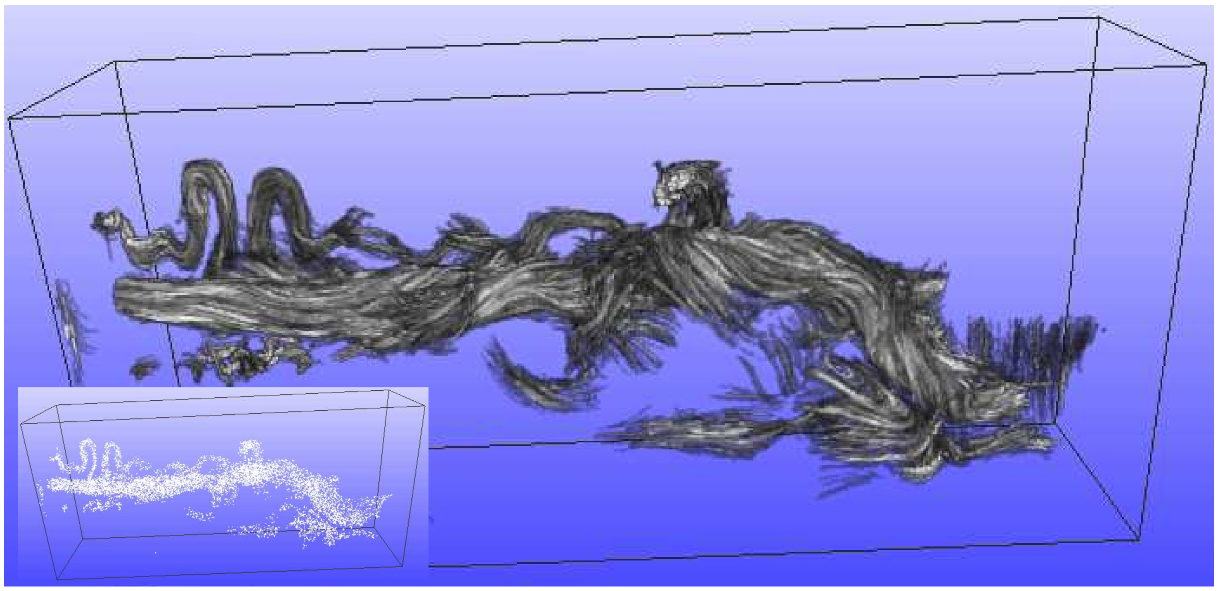

Figure 3.4: The input texture where 9528 points are distributed randomly and the resulting LIC texture.

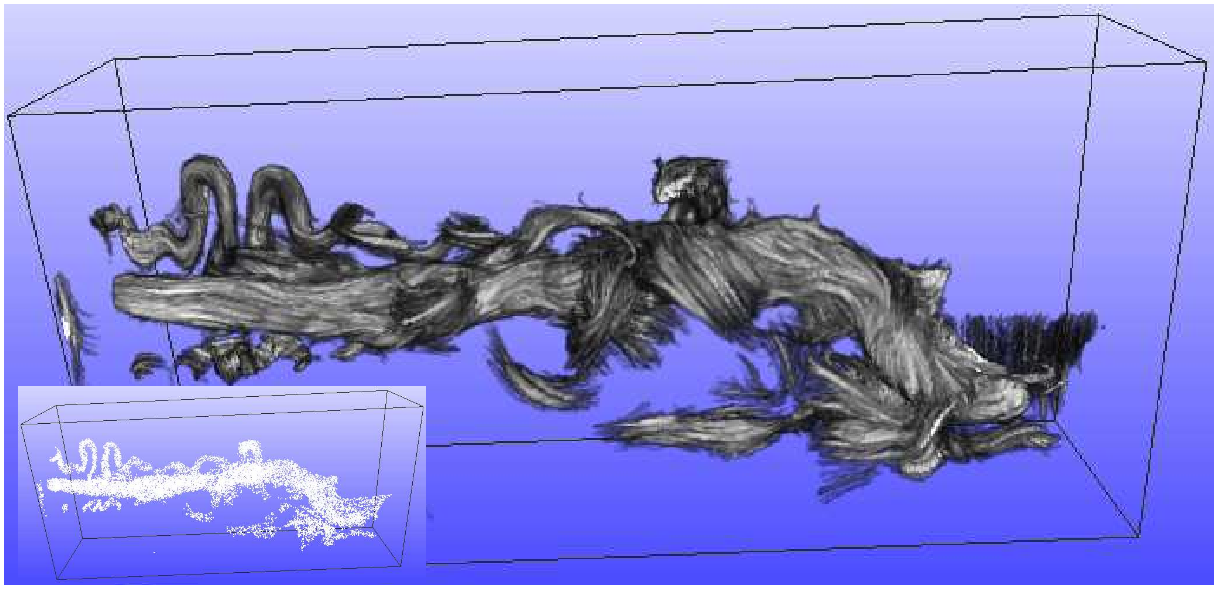

Figure 3.5: The input texture where 20724 points are distributed randomly and the resulting LIC texture.

```
for each voxel v
    set input texture value to zero
for each voxel v
    if (scalar value(v) > threshold value) then
        compute random number ([0,1])
        if (random number > density factor ([0,1]))
            input texture value(v) = 255
```

Figure 3.6: Pseudocode for a random input texture.

and downstream direction (as in LIC), reasonable results are obtained by just controlling the position of the points.

3.1.3 Detail enlargement

It is possible to adjust the size of the input texture so that a single texture cell is covered by lots of output texture cells (voxels). This leads to smoother and more accurate strokes in the LIC texture. Hence, we get images where the details can be more easily seen.

To avoid aliasing, a smaller step size between the sample points have to be used when computing the convolution integrals (2.9) and (2.10). This is to ensure that most of the voxels still are covered by the field line. If we for example use textures twice as large as the vector field, we should reduce the step size by a factor of two. Some examples of detail enlargement are shown in figure 3.7.

3.2 Seed LIC

For large 3D data sets (512^3 or greater), even the Fast LIC algorithm proposed by Stalling and Hege is very computationally intensive. We present a new method for computing 3D LIC textures *Seed LIC* [14]. This method exploits the sparsity of the input texture by calculating field lines and computing the convolution starting from a set of distributed points (the seed points) only. The seed points can either be chosen utilizing certain properties of the field to be visualized, for example the vector magnitude, or they can be distributed randomly.

The algorithm, which we have called Seed LIC, is based upon fast-LIC and can be described by the pseudocode in figure 3.8.

The algorithm behaves similar to fast-LIC. The main difference is that in Seed LIC, we initiate the field lines and compute the convolution starting from the seed points only. In addition, we normalize the intensity of the voxels so that the voxel values vary from 0-255. Hence, the range of the data is increased.

The Seed LIC does not give quite as smooth result as the fast-LIC. In Stalling and Hege's [6] algorithm, the values in every voxel of the output image are computed, while in our algorithm we only compute some of them. As a result, some of the information in the final image is lost. Nevertheless, we see from figure 3.9 that Seed LIC results in images where the directional

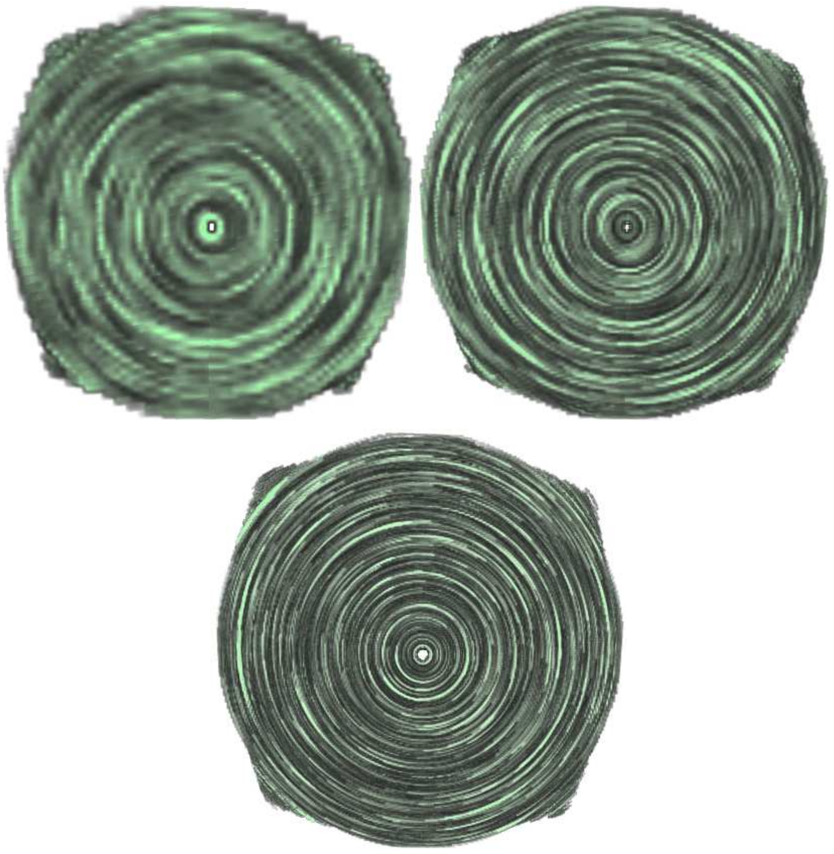

Figure 3.7: Details of a synthetic vector field displayed at different "resolution" factors (1,2 and 4).

structure of the vector field is clearly visible, though the details are not as clear as for the fast-LIC.

Figure 3.10 shows the result of Seed LIC applied to a little more dense input texture of resolution $256 \times 256 \times 256$. It took 70.9 seconds to compute the LIC texture. By increasing the resolution and using a proper sparse input texture, we can reveal the details of the vector field. In comparison, it took 752 seconds to create the fast-LIC image in figure 3.9.

The Seed LIC algorithm can be faster than the Fast LIC algorithm by more than an order of magnitude, and can thus be used in an interactive setting. Instead of a computational time of several hours, the CPU time used for generating the LIC textures can be reduced to a few minutes or even seconds.

3.3 Aliasing

Aliasing [2] can be a problem when using voxel graphics. Representing a field line with voxels, as in LIC, results typically in a "stair-stepped" appearance. Some of the aliasing present in the output texture is removed by the trilinear interpolation performed during the rendering[1]. The reason is when Line Integral Convolution is applied to a sparse input texture, the resulting field

[1]Both Viz and Volumizer 2 have support for trilinear interpolation.

```
for each voxel v
    set voxel value to zero
for each voxel v
    if (Input texture value(v) > 0) then
        initiate field line computation with $\mathbf{x}_0$ = center of v
        Compute convolution $I(\mathbf{x}_0)$
        add result to voxel
        set m=1
        while m < some limit M
            Update convolution $I(\mathbf{x}_m)$ and $I(\mathbf{x}_{-m})$
            add result to voxels containing $\mathbf{x}_m$ and $\mathbf{x}_{-m}$
            set $m = m + 1$
for each voxel v
    normalize intensity according to numHits(v)
for each voxel v
    normalize intensity so that highest value is 255
```

Figure 3.8: Pseudocode of Seed LIC.

lines in the LIC texture are mostly covered by very “low” values, and interpolations between these values and the core values, results in smoother field lines. Figure 3.11 demonstrates the effect of the interpolation. Trilinear interpolation not only reduces aliasing but also, with proper color and opacity tables, creates a halo effect which makes it easier to separate the individual lines from one another. The halo effect will be discussed in section 4.3.

The Seed LIC is more influenced by aliasing than the fast-LIC. This is due to the fewer number of voxels calculated in the Seed LIC algorithm. While this algorithm only assigns values to the voxels along the field lines from the seed points, the fast-LIC assigns values to “nearby” voxels which leads to a smearing of the field line.

Some aliasing present in the Seed LIC texture can be reduced by convolving the output texture with a $3 \times 3 \times 3$ filter. Convolving the texture with a convolution filter A, where a_{222} = 1 and the rest of the entries are set to for example 0.25, leads to a smearing of the field lines, resulting in smoother strokes. Figure 3.12 shows both the results obtained from Seed LIC and the combined Seed LIC and convolution technique.

Convolving the output texture leads to thicker strokes. If preserving the thickness of the individual strokes is wanted, this can be achieved by increasing the resolution of the input and output texture by a factor of three prior to the convolution.

It should be mentioned that the goal in scientific visualization is not to render the scene in a photo-realistic way, but to generate images which provide maximum insight into the data and the underlying processes. Nevertheless, the convolution technique leads to a smearing of the data and can with the appropriate color table provide a better spatial perception of the rendered LIC volume.

The voxel sets used to compare the visualization techniques depicted in the figures 3.11 and 3.12 are subsets of the computed textures. While the fast-LIC texture took about 25 hours to compute, the Seed LIC only took 26.86 seconds. The convolution of the LIC texture took 206.19 seconds to calculate. Although convolution increases the computation time, it still computes

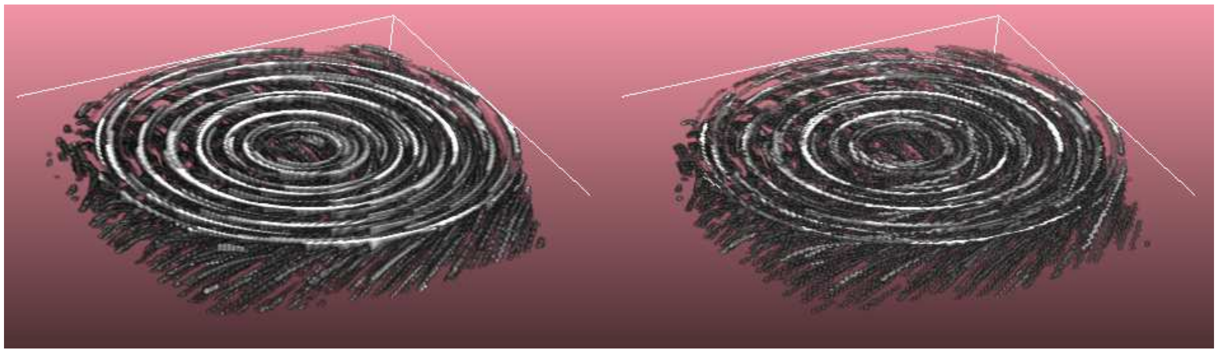

Figure 3.9: Left: The output image after applying fast-LIC to an input texture of resolution $128 \times 128 \times 128$. Right: The output image after applying the Seed LIC to the same input texture. The texture computed with fast-LIC took 752 seconds while the texture computed with Seed LIC took 4.7 seconds. While the Seed LIC algorithm only assigns values to the voxels along the field lines from the seed points, the fast-LIC assigns values to “nearby” voxels which leads to a smearing of the field lines.

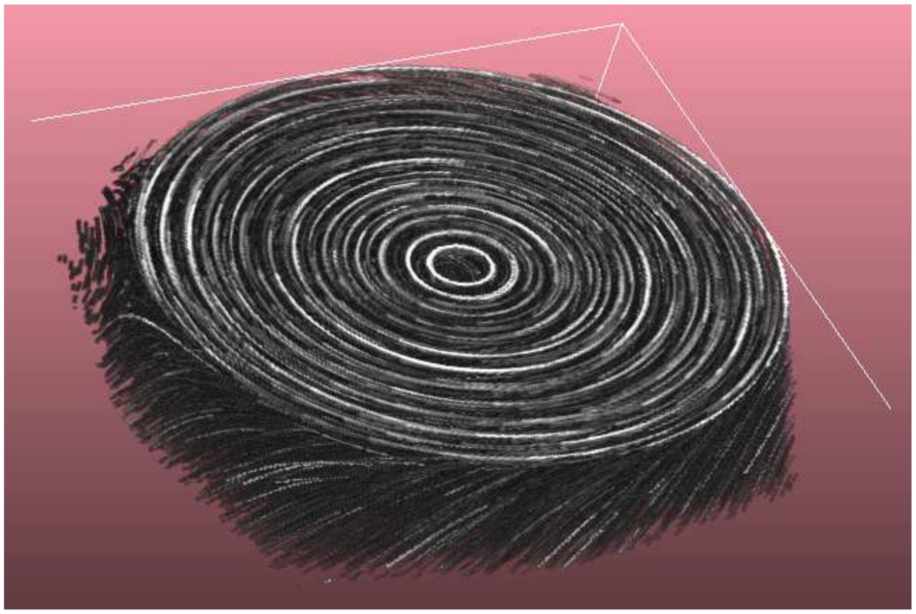

Figure 3.10: Visualization of a texture after Seed LIC was applied to an input texture of resolution $256 \times 256 \times 256$, where the number of seed points were 17341.

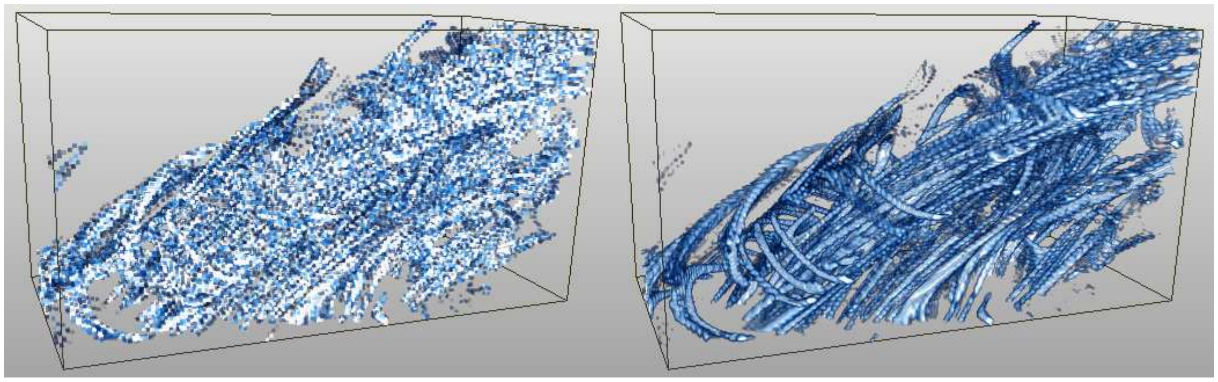

Figure 3.11: The effect of trilinear interpolation. Left: A rendered LIC texture without the use of trilinear interpolation. Right: A rendered LIC texture with the use of trilinear interpolation. Both the LIC textures are computed with fast-LIC. Trilinear interpolation leads to smoother field lines.

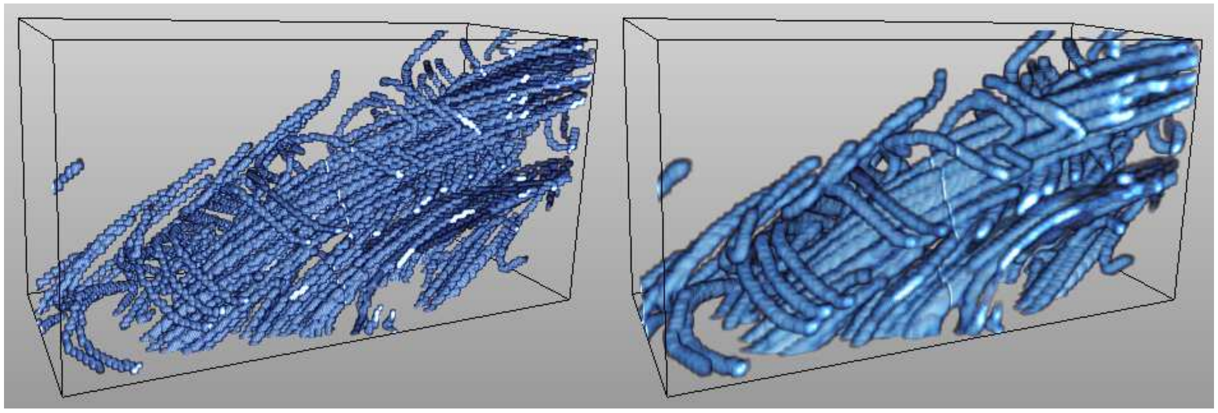

Figure 3.12: Left: The result after applying seedLIC. Right: The result after convolving the output texture obtained from Seed LIC. Convolving the LIC texture leads to smoother field lines.

much faster than fast-LIC. The complete fast-LIC texture and the convolved Seed LIC texture are displayed in the figures 3.13 and 3.14. The resolution of the textures are $398 \times 298 \times 798$.

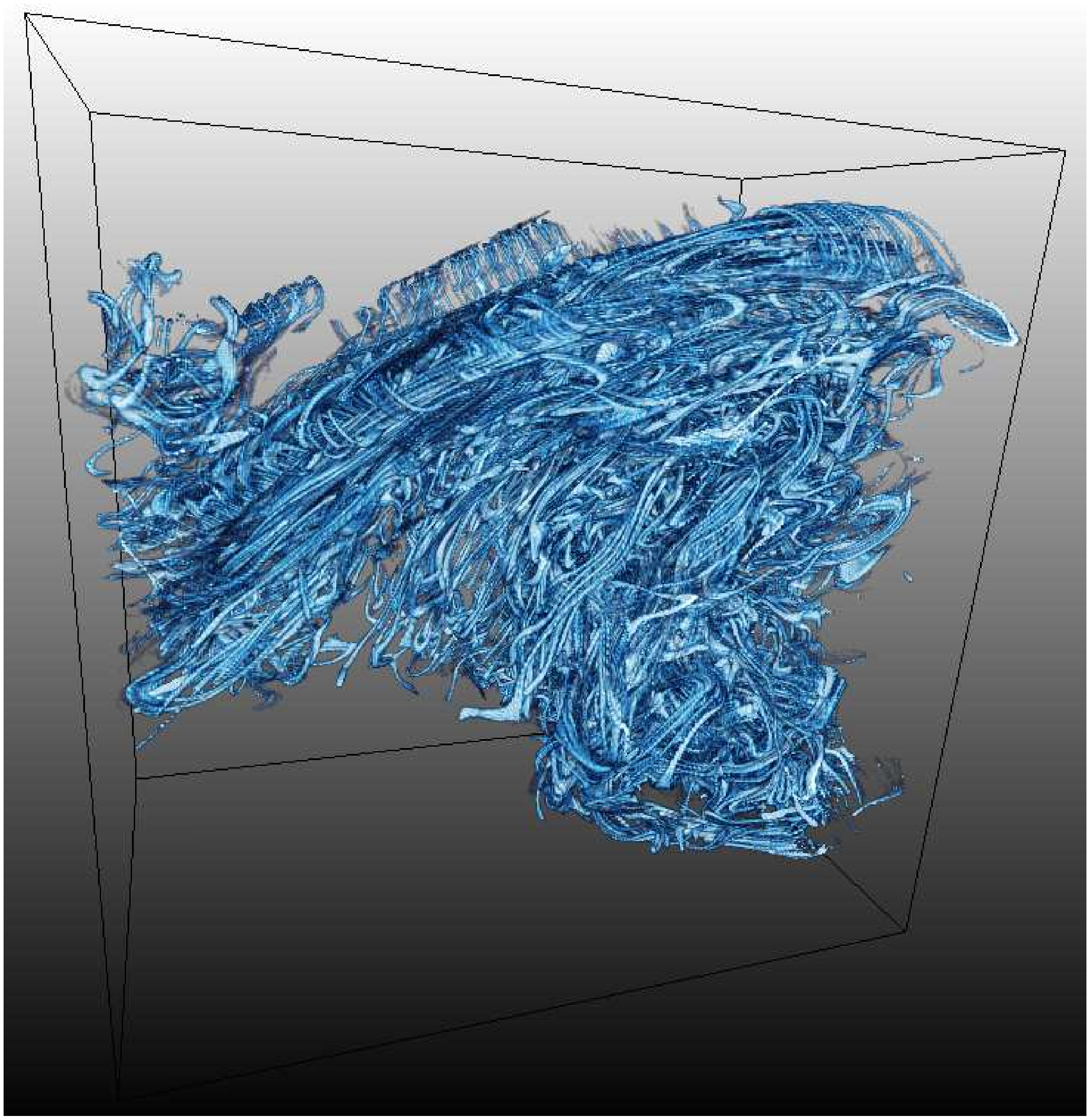

Figure 3.13: Visualization of a vorticity field, obtained from [15], using fast-LIC.

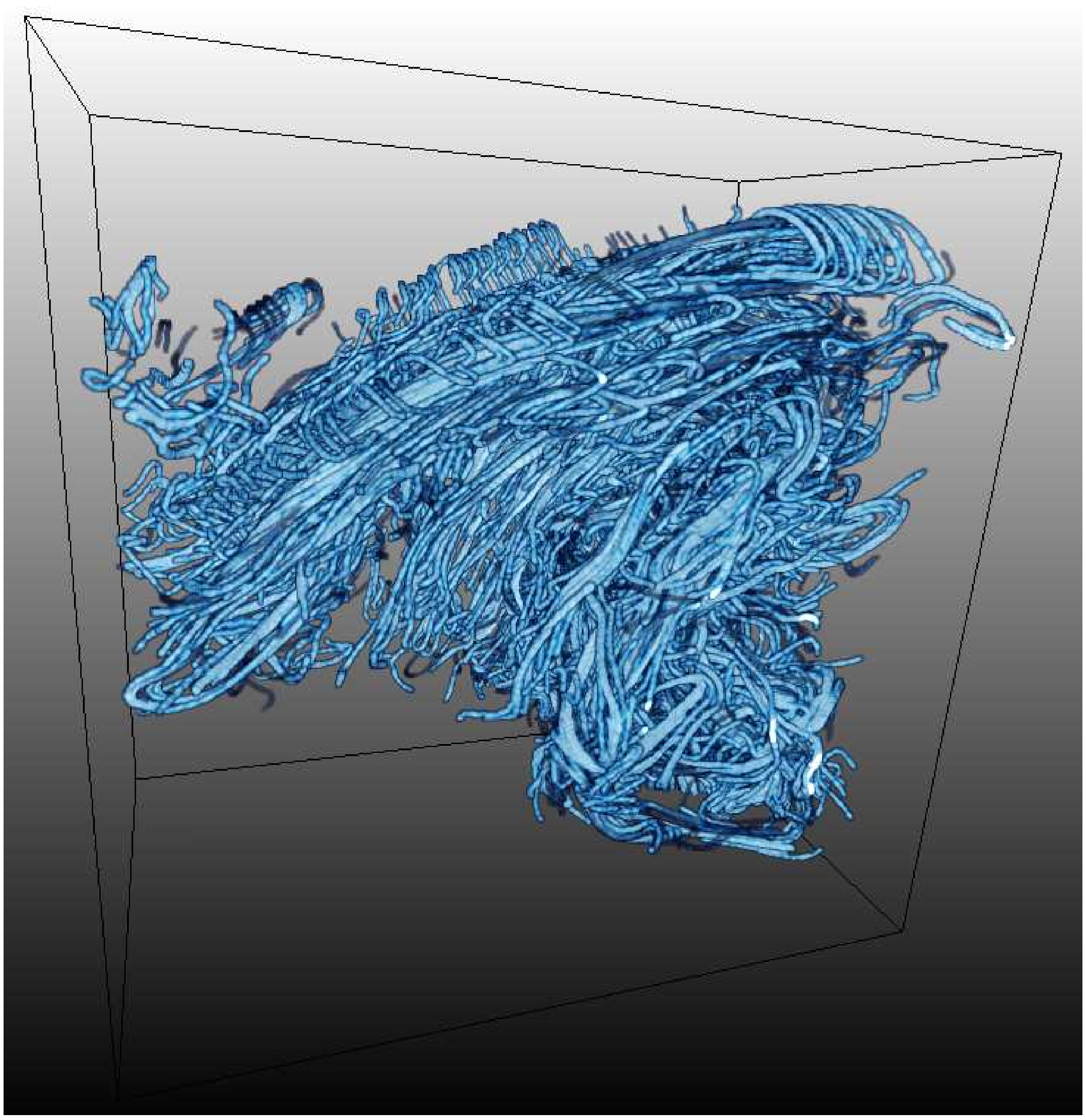

Figure 3.14: Visualization of a vorticity field, obtained from [15], using Seed LIC and convolution.

Chapter 4

Volume visualization with LIC

4.1 Assignment of color and opacity values

Some graphics systems allow interactive modification of the texture lookup tables used for the assignment of color and opacity values. This is an indispensable feature that simplifies the process of finding an appropriate color table and to produce a meaningful visualization of the LIC texture. A possible problem in representing volume LIC textures, is how to clearly and effectively convey the inner details of the texture and the 3D shape and relative depth relations among the similarly directed, densely clustered field lines traced by LIC. From subsection 3.1.2, we have seen that the use of a sparse input texture is one way to reveal the inner details of a vector field. The manipulation of the opacity value is another possibility. Suppressing or assigning low opacity to the lowest scalar values results in a semi-transparent representation of the LIC texture and can be used as an alternative to the use of sparse input textures [16]. Figure 4.1 shows how the transfer function for the opacity or the alpha values affect the visualization of a dense LIC texture.

Color and opacity can also be used to enhance the contrast between the various strokes present in the output texture. This is especially needed when applying Line Integral Convolution to sparse input textures, since most of the voxels values in the resulting LIC texture have a tendency to be rather low. Good results are achieved with a ramp-like function for *alpha* and *value* (in the HSVA model) that increases from low to high data values (see figure 4.1).

4.2 Clipping functionality

The use of clip planes is another approach that allows the user to explore the interior structures of the LIC texture. By interactively moving a clip plane inside the volume, we are able to follow the direction of the field lines more clearly and significantly improve the spatial understanding of the vector field. Since the surfaces of the clip planes do not generally follow the direction of the field lines, best results are achieved when the field lines are separated from one another. This can be done, either by using sparse input texture or manipulation with the opacity values. If a dense texture is used, it can be difficult to get an impression of the directional structures of the vector field because of the apparently missing correlation (see figure 4.2). Figure 4.3 demonstrate the effect of interactive clip planes in a volume renderer application.

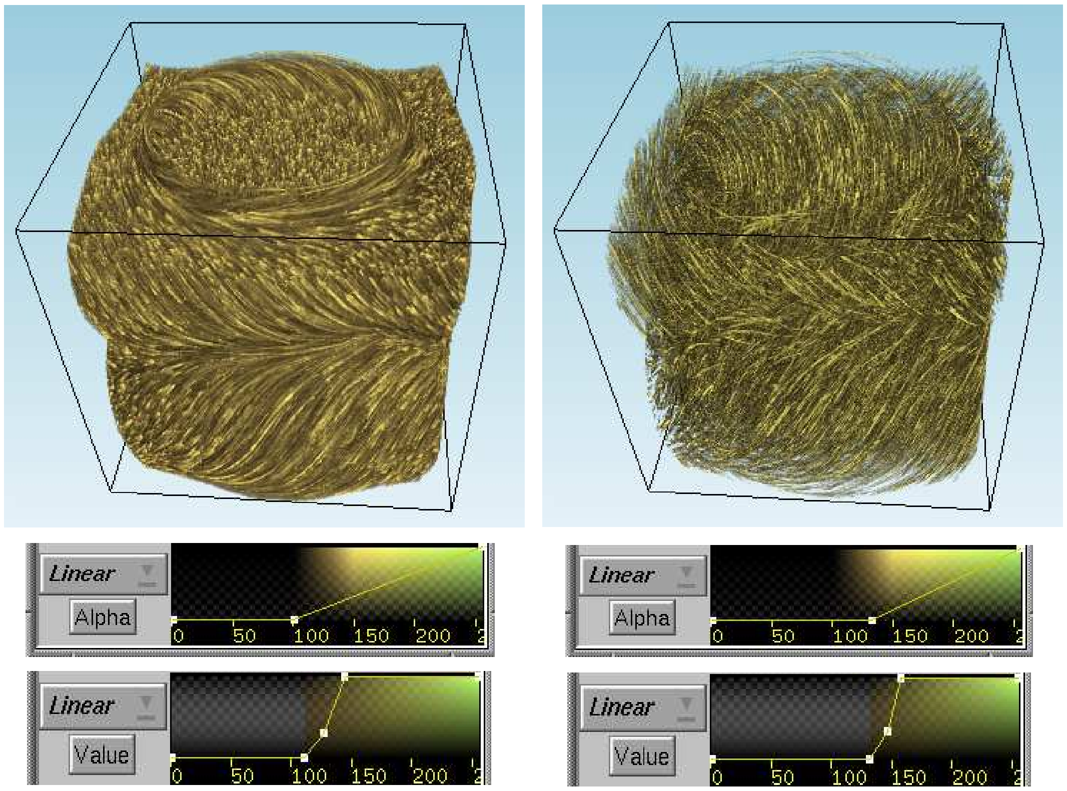

Figure 4.1: Visualization of a dense LIC texture with different alpha and value functions.

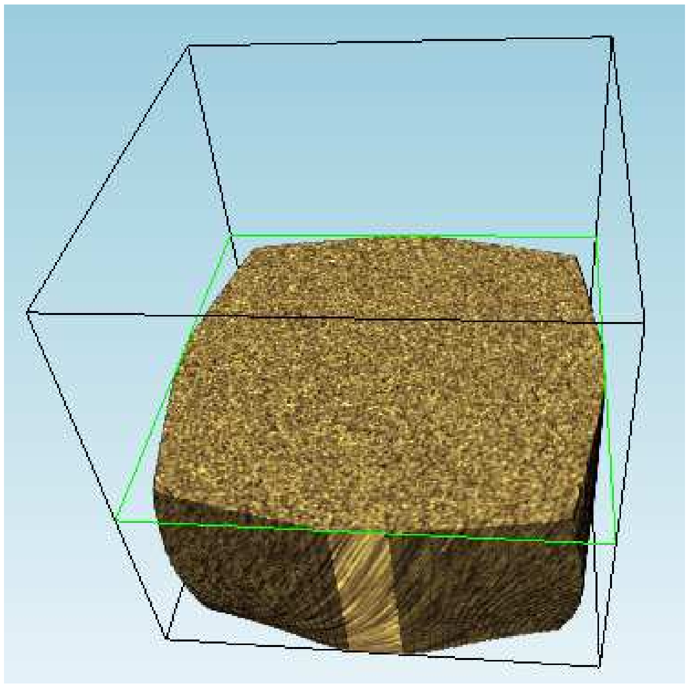

Figure 4.2: Visualization of a dense LIC texture using a clip plane.

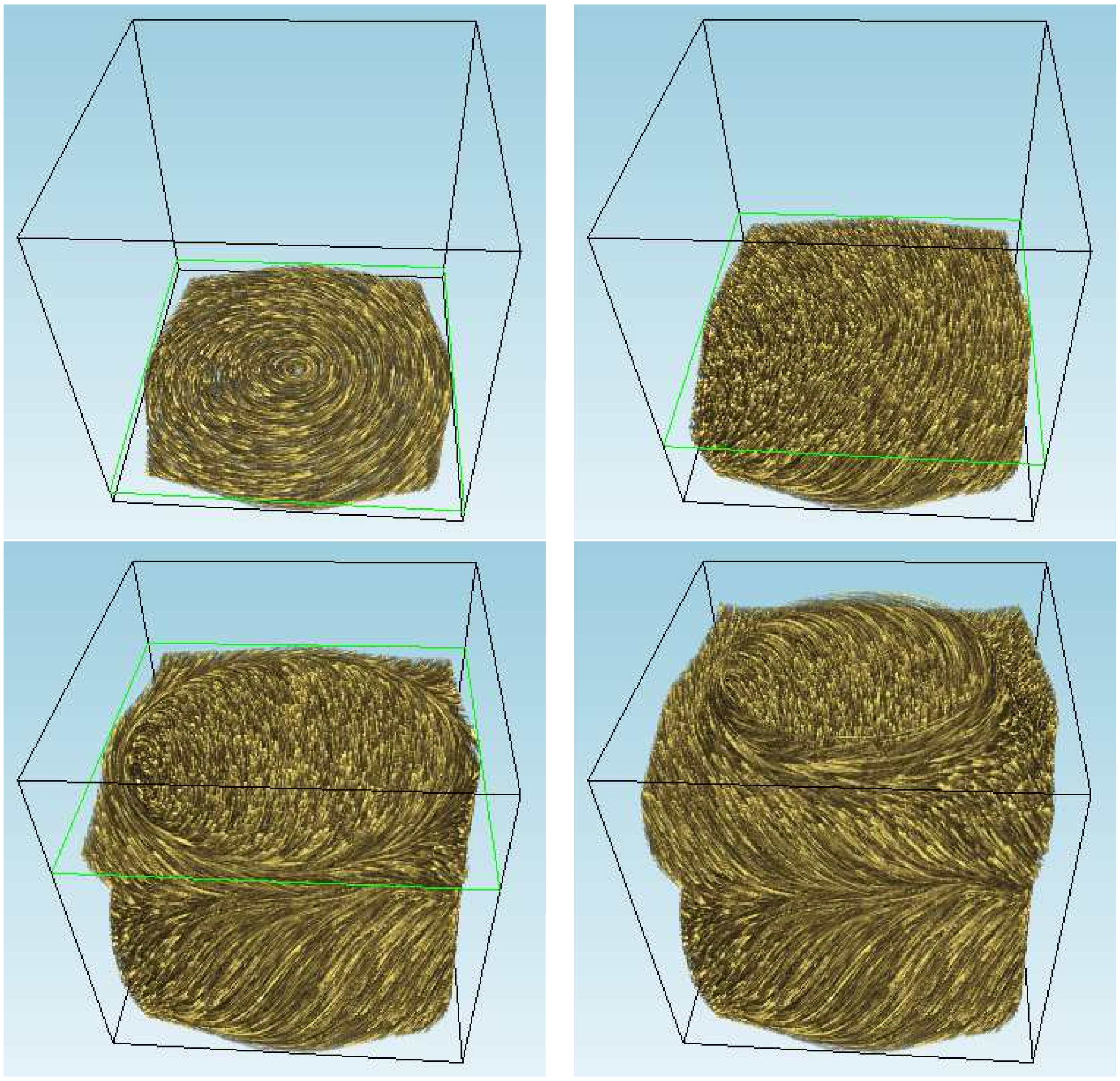

Figure 4.3: Visualization of a LIC texture using a clip plane to reveal the inner structures of the vector field.

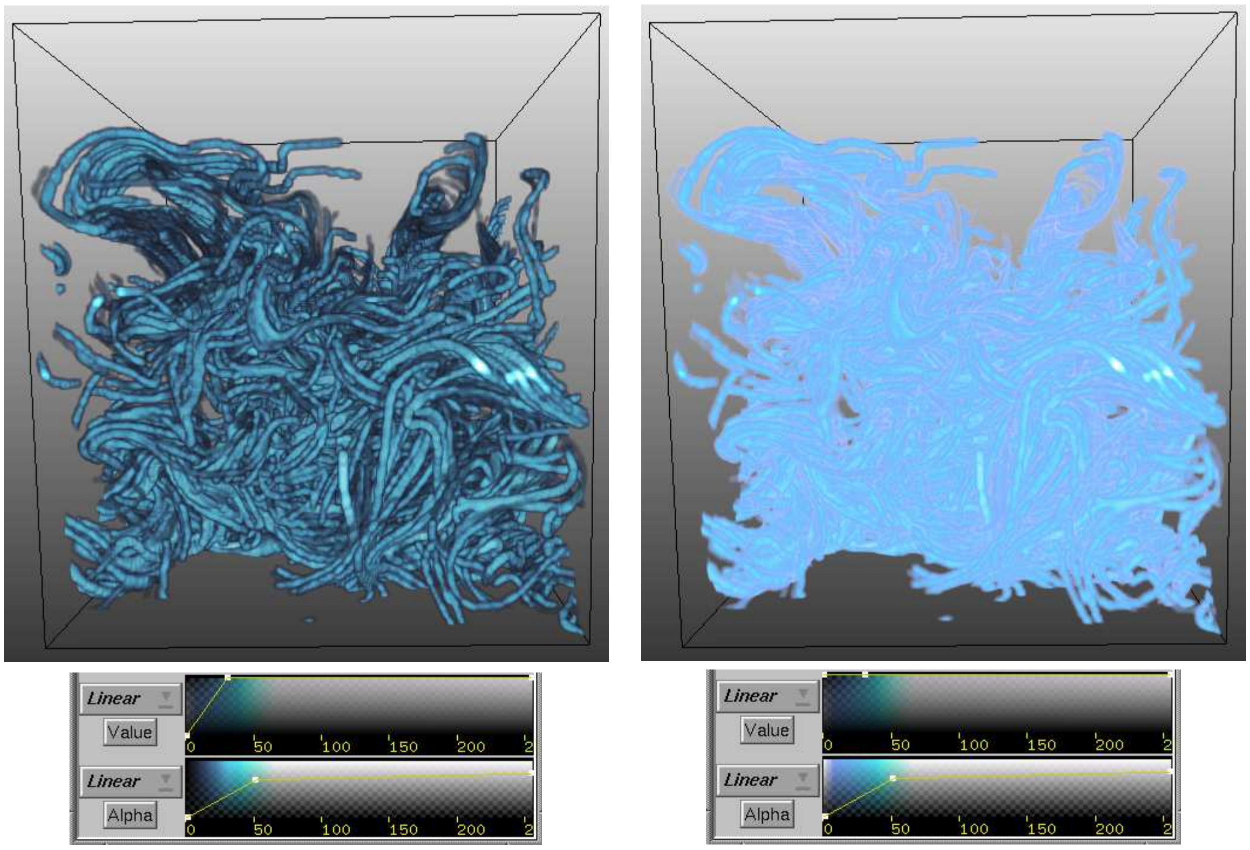

Figure 4.4: Limb darkening used to visualize field lines in a LIC texture. Left: By letting *alpha* change from zero to higher opacity values and the *value* V go from dark to bright, we obtain a deep three-dimensional rendering of the field lines. Right: In contrast, if a constant *value* ($V = 1$) is applied to all the voxels in the LIC volume, a flatter appearance results.

4.3 Shading

To reveal the depth relation among the field lines, a shading, or halo technique called *limb darkening* is used. The technique has its name because the effect obtained is similar to what is well known in astrophysics. Looking at the sun with a small telescope, it has been evident that the center is brighter compared to the limb. This is due to the fact that when watching the center of the sun we see deeper into its atmosphere where the temperature is higher than in the layers we see closer to the edge. Since higher temperatures are visible as brighter, we observe a darkening effect at the limb. This can be utilized in volume visualization as well. By assigning darker values and decreasing the opacity near the edges of an "object" we obtain a more three-dimensional look. This is illustrated in the figure 4.4.

To emphasize the halo effect and to reduce aliasing effects due to the use of voxels to represent the field lines, the textures are oversampled by a factor of 2 to 4 in each dimension and then convolved with an isotropic $3 \times 3 \times 3$ filter. This approach smears the field lines outwards, making the strokes in the output texture thicker and smoother, improving the 3D perception of the LIC texture.

4.4 Two fi elds visualization

Direct volume rendering is very useful when studying multiple fields at the same time, and there exists several options for doing this. The rendering of two fields, can for example be achieved by using two independent sets of color and opacity tables. Another approach is to let one of the fields define the structure or the "body" through opacity and the other field to determine the color.

Figure 4.5 shows an example of the first approach by visualizing two fields simultaneously: *vorticity* ($\boldsymbol{\omega} = \nabla \times \mathbf{u}$), represented by the LIC texture and *enstrophy* ($\eta = \|\boldsymbol{\omega}\|^2$). Figure 4.6 shows the second approach to depict the directional structures of the vorticity field within vortices obtained from [15]. In this method, which we have called “polkagris” or "candy cane" visualization, the enstrophy is used to define the opacity while the LIC texture displaying the vorticity field is used to define the color. Since enstrophy expresses the vortical structures of the flow, we get images of vortex tubes colored by the LIC texture conveying the directional structure of the vorticity field. The field lines of the vorticity field twist around the vortices, resulting in "objects" similar to "candy canes".

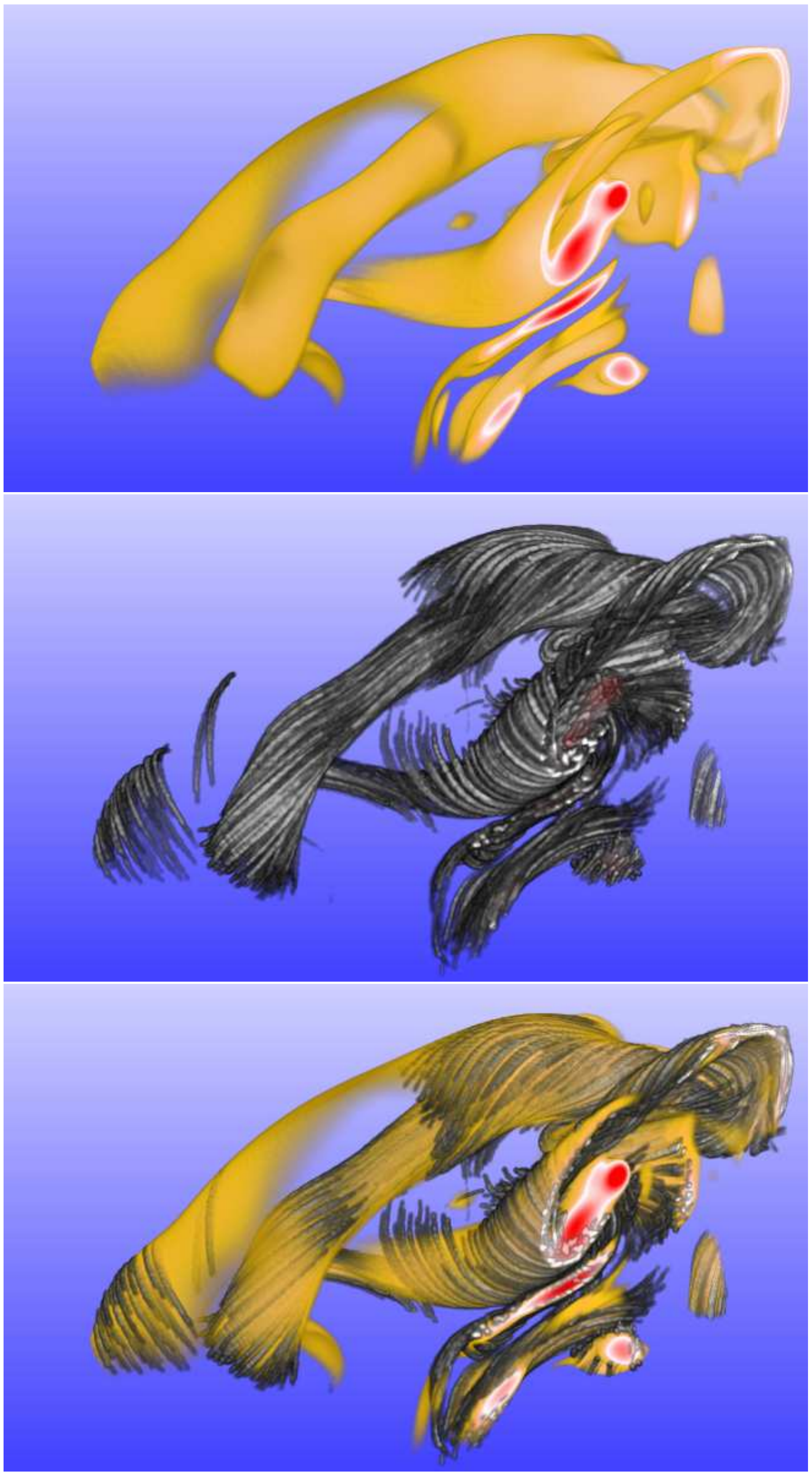

Figure 4.5: Top: Visualization of the enstrophy field. The color varies from yellow to red with increasing enstrophy value. Middle: Visualization of the vorticity field using LIC. Bottom: Visualization of the vorticity field using LIC. Bottom: Two field visualization, using two independent sets of color and opacity tables. Both fields are displayed simultaneously.

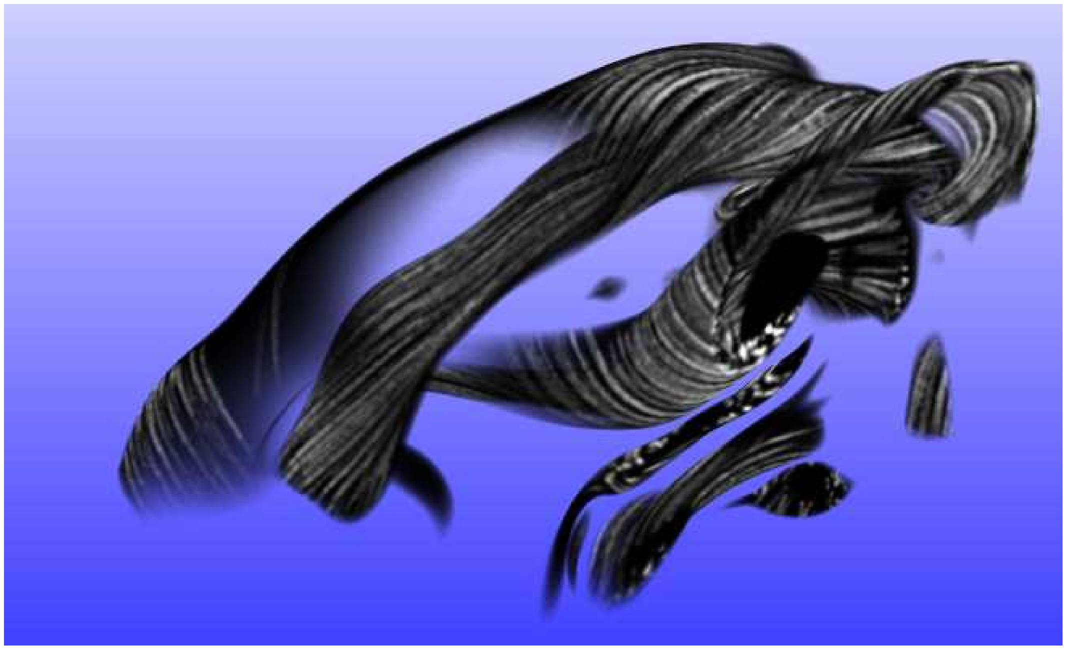

Figure 4.6: Two-field visualization, using enstrophy to define the opacity abd the vorticity field represented by the LIC texture to define the color. The field lines of the vorticity field twist around the vortices, resulting in features reminiscent of "candy canes".

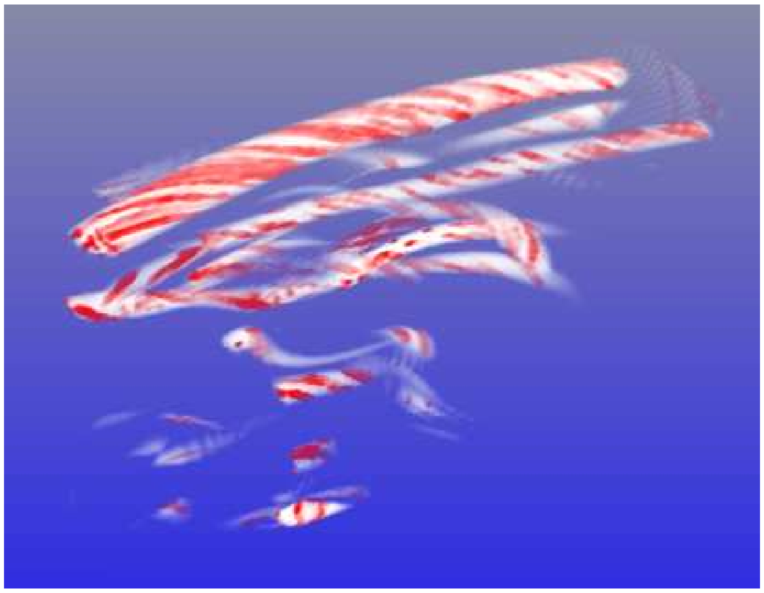

Figure 4.7: Polkagris visualization.

Bibliography

[1] D. STALLING, M. ZÖCKLER, and H.-C. HEGE. Fast display of illuminated field lines. In *IEEE transactions on visualization and computer graphics*, volume 3, pages 118–128, 1997.

[2] W. SCHROEDER, K. MARTIN, and B. LORENSEN. *The Visualization Toolkit*. Prentice Hall, 2nd edition, 1998.

[3] J. J. VAN WIJK. Spot noise-texture synthesis for data visualization. In *Computer Graphics Proceedings*, volume 25 of *Annual Conference Series*, pages 309–318, Juli-August 1991.

[4] W. C. DE LEEUW and J. J. VAN WIJK. Enhanced Spot Noise for Vector Field Visualizations. In *Proceedings of Visualization '95*, pages 233–239, 1995.

[5] B. CABRAL and C. LEEDOM. Imaging Vector Fields Using Line Integral Convolution. In *Computer Graphics Proceedings*, volume 27 of *Annual Conference Series*, pages 263–270, July 1993.

[6] D. STALLING and H.-C. HEGE. Fast and Resolution Independent Line Integral Convolution. In *Computer Graphics Proceedings*, Annual Conference Series, pages 249–256, August 1995.

[7] V. INTERRANTE and C. GROSCH. Recent Advances in Visualizing 3D Flow with LIC. ICASE Report No. 98-26, NASA Langley Research Center, July 1998. http://citeseer.nj.nec.com/interrante98recent.html.

[8] J. O. LANGSETH. 3D visualization of shock waves using volume rendering. In *Proceeding of the International Conference on Godunov Methods*, pages 549–556, 1999.

[9] M. ZÖCKLER, D. STALLING, and H.-C. HEGE. Parallel Line Integral Convolution. In *Proceedings First Eurographics Workshop on Parallel Graphics and Visualization*, Annual Conference Series, pages 249–256, September 1996.

[10] L. FORSELL. Visualizing Flow Over Curvelinear Grid Surfaces Using Line Integral Convolution. In *Proceedings of Visualization 94*, pages 240–247, Oktober 1994.

[11] H.-W. SHEN, C. R. JOHNSON, and K.-L. MA. Visualizing Vector Fields Using Line Integral Convolution and Dye Advection. In *Proceedings of the 1996 Volume Visualization Symposium*, pages 63–70, Oktober 1996.

[12] R. WEGENKITTL, E. GRÖLLER, and W. PURGATHOFER. Animating Flow Fields: Rendering of Oriented Line Integralconvolution. In *Proceedings of Computer Animation 97*, pages 15–21, June 1997.

[13] V. INTERRANTE. Illustrating Surface Shape in Volume Data via Principal Direction-Driven 3D Line Integral Convolution. In *Computer Graphics Proceedings*, Annual Conference Series, pages 109–116, August 1997. http://citeseer.nj.nec.com/interrante97illustrating.html.

[14] ANDERS HELGELAND and OYVIND ANDREASSEN. Visualization of Vector Fields Using Seed LIC and Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):673–682, November-December 2004.

[15] J. WERNE and D. C. FRITTS. Stratified shear turbulence: Evolution and statistics. *Geophysical research letters*, 26(4):439–442, February 1999.

[16] C. REZK-SALAMA, P. HASTREITER, C. TEITZEL, and T. ERTL. Interactive Exploration of Volume Line Integral Convolution Based on 3D-Texture Mapping. In M. GROSS D. EBERT and B. HAMANN, editors, *IEEE Visualization '99'*, pages 233–240, San Francisco, 1999. http://citeseer.nj.nec.com/398075.html.