Image-Guided Streamline Placement

Greg Turk, University of North Carolina at Chapel Hill
David Banks, Mississippi State University

Abstract

Accurate control of streamline density is key to producing several effective forms of visualization of two-dimensional vector fields. We introduce a technique that uses an energy function to guide the placement of streamlines at a specified density. This energy function uses a low-pass filtered version of the image to measure the difference between the current image and the desired visual density. We reduce the energy (and thereby improve the placement of streamlines) by (1) changing the positions and lengths of streamlines, (2) joining streamlines that nearly abut, and (3) creating new streamlines to fill sufficiently large gaps. The entire process is iterated to produce streamlines that are neither too crowded nor too sparse. The resulting streamlines manifest a more hand-placed appearance than do regularly- or randomly-placed streamlines. Arrows can be added to the streamlines to disambiguate flow direction, and flow magnitude can be represented by the thickness, density, or intensity of the lines.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image generation; I.4.3 [Image Processing]: Enhancement.

Additional Key Words: Vector field visualization, flow visualization, streamline, random optimization, random descent.

1 Introduction

The need to visualize vector fields is common in many scientific and engineering disciplines. Examples of vector fields include velocities of wind and ocean currents (*e.g.*, for weather forecasting), results of fluid dynamics simulation (*e.g.*, for calculating drag over a body), magnetic fields, blood flow, components of stress and strain in materials, and cell migration during embryo development. Existing techniques for vector field visualization differ in how well they represent such attributes of the vector field as magnitude, direction, and critical points.

This work was motivated by two recent innovations for displaying vector fields: spot noise [van Wijk 91] and line-integral convolution (LIC) [Cabral & Leedom 93]. We wondered how to compare the results of the techniques. What is the gauge that measures how well a certain method depicts a vector field? Evidently the placement of the graphical elements is tremendously important. The graphical elements (e.g. coherent streaks) should follow the flow direction, but they should not be spaced too close together or too far apart. Both spot noise and LIC can produce images where stream-aligned streaks are evenly distributed, but that is more an indirect result than a guiding principle in the algorithms. How can the streamlines be positioned to explicitly satisfy a desired distribution?

The elegant hand-designed streamline drawings in physics texts (for example in Figure 1a) provide ample inspiration for vector field illustrations. The streamlines in such illustrations are placed so that no region is devoid of streamlines and no region is overpopulated with them. The eye is drawn to regions where the density of ink in one place differs greatly from that of the surrounding region. When the density of the streamlines is allowed to vary in such illustrations, it is usually to represent field magnitude, where denser line spacing shows greater field strength.

Bertin shows another effective hand-designed representation of flow where the direction of ocean current is represented by chains of arrows that are laid out end-to-end so that the eye connects arrows into streamlines and thus gets a stronger sense of flow orientation [Bertin 83]. The success of this representation depends on having chosen proper endpoints for these chains so that nowhere does the image become cluttered. The techniques presented in our paper will permit designers of vector-field visualizations to control streamline-spacing automatically in order to achieve results that mimic hand-drawn figures.

2 Previous Work

A *streamline* is an integral curve that is everywhere tangent to a given vector field (see, for example, [Kundu 90]). Many researchers have examined how to effectively and accurately *integrate* streamline paths through both regular and irregular meshes. To our

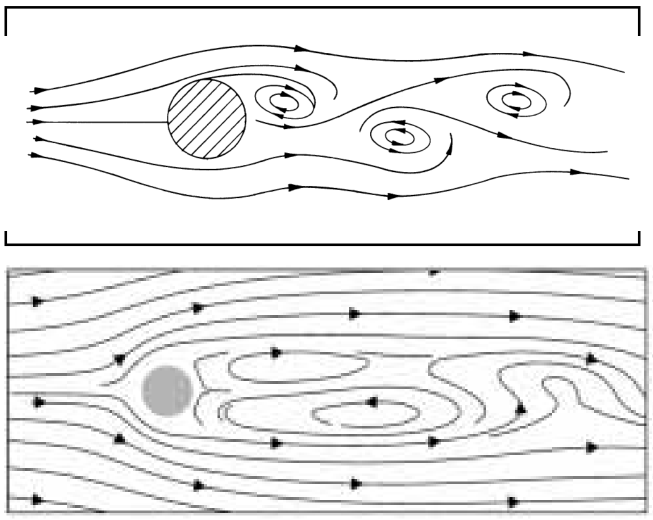

Figure 1: (a) Hand-designed illustration of flow around a cylinder, taken from [Feynman 64] and used with permission from the California Institute of Technology. (b) Automatically generated flow lines using streamline optimization. Data is from fluid flow simulation.

surprise, however, discussions of how best to *place* streamlines are almost nonexistent in the visualization literature. We are aware of three techniques that are used to “seed” streamlines within a vector field: regular grids, random sampling, and user-specified seed points for initiation of streamlines. Our knowledge of random and regular grid seeding of streamlines is almost entirely limited to private communications with visualization researchers. The one published technique that we have found uses particle traces on a 3D surface that are terminated when they come too close to the paths of other particles [Max *et al.* 94]. The virtual wind tunnel (a 3D immersive display system for flow visualization) allows users to initiate streamlines singly or in bundles [Bryson & Levit 91].

Recently there have been several exciting developments in displaying vector fields using texture synthesis. Line integral convolution is a procedure that stretches a given image along paths that are dictated by a vector field [Cabral & Leedom 93] [Forsell 94] [Stalling & Hege 95]. Spot noise is a method of creating noise-like texture by compositing many replicas of a shape [van Wijk 91] [de Leeuw & van Wijk 95]. When the shapes that create spot noise textures are stretched according to a given vector field, the resulting images illustrate the vector field’s direction. Both line integral convolution and spot noise are well-suited to depicting the fine detail of flow *orientation*. They are somewhat less successful (in a single, static image) at showing the flow *magnitude*; moreover, the local flow direction is ambiguous in the sense that it can be interpreted to be either of two directions that are 180 degrees apart.

A very different method of illustrating vector field data is to show the important *topological* features of the flow. In general, streamlines that lie in a small neighborhood follow nearly-parallel paths. The exceptions (in a continuously differentiable vector field) occur in neighborhoods of points with zero-valued vectors. Several researchers have developed techniques to identify these critical points (sources, sinks, spirals, centers, and saddles) and the streamlines that issue from them in eigen-directions [Globus *et al.* 91] [Helman & Hesselink 91]. These particular points and curves partition a vector field into simpler regions where a texture-based method suffices to display details of the vector field [Delmarcelle & Hesselink 94].

The remainder of this paper is organized as follows. In Section 3 we present a key concept in our work– a visual quality measure for flow illustrations– and show how this measure can create visually pleasing illustrations containing short arrows. In Section 4 we demonstrate the creation of illustrations that contain well-placed long streamlines. Section 5 discusses how these streamlines can be enhanced to produce a final illustration. We conclude by discussing other applications that might use optimization based on a visual quality measure.

3 Placement of Streamlets

Hedgehog illustrations (sometimes called vector plots) are perhaps the most commonly used method of illustrating a two-dimensional vector field. These are short field-aligned segments or arrows whose base points lie on a regular grid (see Figure 2). The lengths of the segments are often varied according to the field magnitude. The popularity of hedgehog illustrations is almost surely due to their ease of implementation. The resulting images can be slightly enhanced by using short streamlines (streamlets) that curve with the flow instead of using straight lines. We use such streamlets in our figures 2, 3, and 4.

Two artifacts are often present in hedgehog plots. First, the human eye often picks out runs of adjacent arrows and groups them together visually, despite the fact that these groups are an artifact of the underlying grid pattern and not related to the vector field being

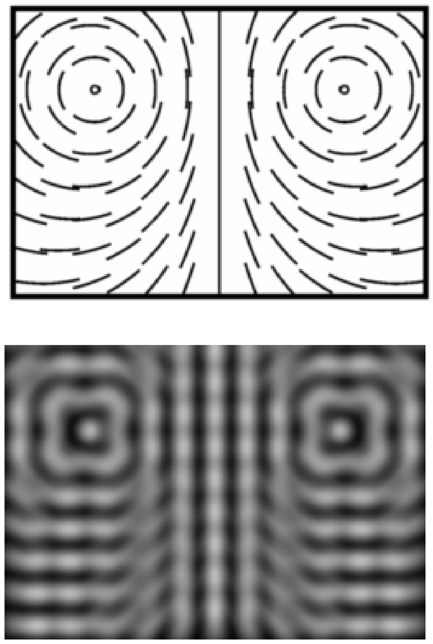
Figure 2: (a) Short streamlines with centers placed on a regular grid (top); (b) filtered version of same (bottom).

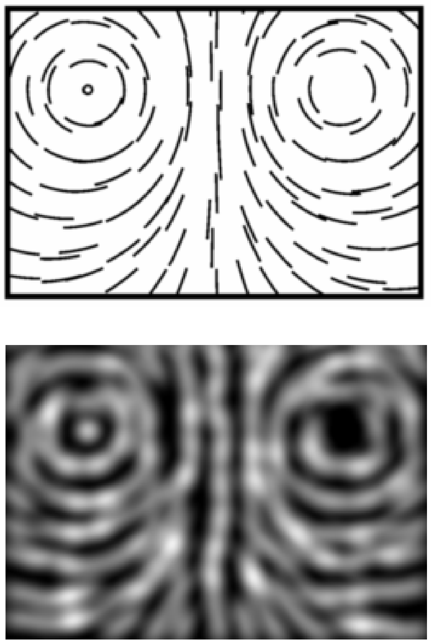
Figure 3: (a) Short streamlines with centers placed on a jittered grid (top); (b) filtered version showing bright and dark regions (bottom).

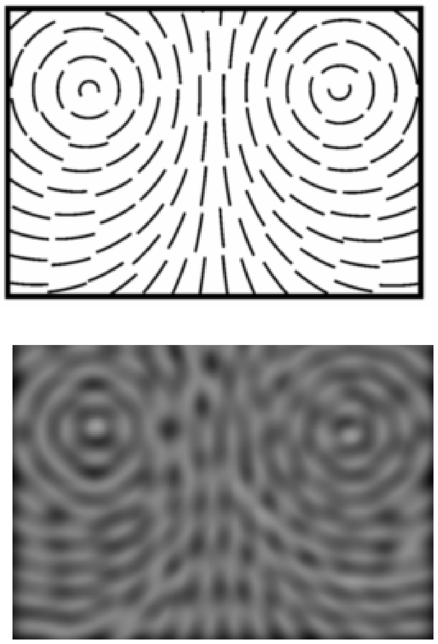
Figure 4: (a) Short streamlines placed by optimization (top); (b) filtered version showing fairly even gray value (bottom).

illustrated. This effect can be seen in Figure 2a where three vertical columns of streamlets erroneously suggest the presence of three parallel field lines. One way to lessen this problem is to oversample the seed points that produce the short segments. The drawback with oversampling is that the resulting image becomes so filled with streamlets that the eye can no longer discern individual elements. A better solution to the sampling problem is to introduce noise, slightly jittering the positions of the arrows to make their regularity less noticeable [Crawfis & Max 92] [Dovey 95]. This strategy is illustrated in Figure 3a.

The second problem with hedgehogs is that as streamlets are placed close together, portions of neighboring arrows come very close to one another and may even overlap. Jittering the streamlets may in fact make the overlaps more frequent (compare Figures 2a and 3a). The twin problems of overlapped streamlets and grid regularity both distract the viewer from the data being visualized; we would like to reduce such distractions. We achieve this goal by using an energy measure to guide streamlet placement and thus improve the quality of the final image.

3.1 Optimization of Streamlet Positions

In the discussion that follows, S represents a collection of streamlets s_n for a given vector field V. The elements of S are idealized zero-width curves, distinct from the geometric primitives (*e.g.*, line segments or anti-aliased curves) employed to render them. We denote by $I(x,y)$ the idealized two-dimensional image of the streamlets in S, with $I(x,y) = 0$ except along streamlines in S where it behaves like the Dirac delta function.

Our method creates hedgehog plots by incrementally improving an initial collection of streamlets. The initial collection can be created by placing the streamlets either on a regular grid or in some random fashion, and the final results appear to be independent of which initialization method is chosen. An image may be improved by selecting one streamlet at random and moving it a small amount in a random direction. If the resulting image has a lower energy measure (lower energy means better quality) then that change is accepted. This process is repeated many times, terminating when the energy reaches a threshold or when acceptance of random changes become rare. Such a process is sometimes referred to as random optimization or random descent. Figure 4a shows the result of this algorithm applied to the same vector field as in Figures 2 and 3. Notice how the streamlets of Figure 4 are more evenly spaced than in Figures 2 and 3.

The energy measure that guides the optimization is based on a low-pass-filtered (blurred) version of the image of S which is compared against a uniform gray-level. Let $L * I$ represent a low-pass-filtered version of the image I, where L is a given filter function. If t is the target gray-scale value, then we define the energy measure E as the squared error integrated over the domain:

$$E(I) = \int_X \int_Y [(L * I)(x,y) - t]^2 \, dx \, dy$$

The motivation for this energy measure is that the eye is drawn to regions of an illustration where the density of ink is uneven, and in a hedgehog plot we do not want to draw the eye to any inadvertently bright or dim places. The streamlets should be evenly placed across the image instead of being crowded in any one location. A blurred image contains high values where the streamlets are too close together and low values in regions that are devoid of streamlets. Salisbury and his co-workers made similar used of low-pass filtering to decide whether or not to lay down strokes for pen-and-ink illustrations [Salisbury *et al.* 94]. Figure 2b and 3b show low-pass-filtered versions of Figures 2a and 3a. Locations where two streamlets crowd together in Figures 2a and 3a appear as a high intensity (black) spot in Figures 2b and 3b. Figure 4a and 4b show the corresponding images after the optimization routine has been run. The intensity level in Figure 4b is more uniform than in Figures 2b and 3b.

When the optimization process is animated it looks as though each streamlet is pushing away other nearby streamlets, reminiscent of methods that use repulsion between points to evenly distribute samples on a surface [Turk 91] [Witkin & Heckbert 94]. This similarity should come as no surprise, since both methods are designed to minimize an energy term by making small changes in the position of graphical elements. In fact, we too have implemented streamlet-repulsion as a method for creating hedgehog plots. The visual results of the repulsion method are very similar to the results of random optimization, and the running times are also similar. We pursued the random-descent technique rather than the repulsion method because we expected random descent to be easily extensible to the more complicated task of placing longer streamlines within V (Section 4).

3.2 Implementation of Low-Pass Filter

This section describes the implementation details for efficiently computing the energy term for a given set S of streamlets. There are three components to this computation: the representation of the blurred image, the low-pass filter used to perform the blur, and the manner in which we apply the filter to calculate this blurred image.

It would be computationally prohibitive to calculate the energy term E by actually filtering an entire image each time we consider a random change to some streamlet s_n. Instead, we associate with s_n certain information about how it affects the low-pass-filtered image. The blurred image B contains pixel values for an image of S. A streamlet maintains a list of pixels that it affects in B, together with the values that it contributes to each of those pixels. To test whether moving s_n would improve the value of E, we first remove the contribution of s_n from its list of pixels in B and correct the value of E based on the changes. Next, we add in the pixel contributions for the new position of s_n and recalculate E. We retain the change to s_n if the new value of E is better; otherwise we revert to the old position for s_n. The (un-blurred) image I is purely a conceptual aid, and at no time during optimization do we generate an actual representation of I.

Two criteria influence the choice of a filter to create the blurred image B. First, the filter kernel should have compact support so that filtering operations are fast to compute. Second, the point-spread function should fall off smoothly so that the quality measure changes smoothly with small changes in streamline position. This allows the optimization to detect changes in E even for small changes in the image.

We use the following circularly symmetric filter kernel (from a basis function of cubic Hermite interpolation) to blur the image:

$$K(x, y) = 2r^3 - 3r^2 + 1, \qquad r < 1$$

$$K(x, y) = 0, \qquad r >= 1$$

where $r = \mathrm{sqrt}(x^2 + y^2) / R$.

This function has a similar shape to a two-dimensional Gaussian filter, but it falls off to zero at a distance R away from its center. The ideal density for a set of streamlines may be varied across the image by stretching or shrinking the radius R of the filter.

We sample a streamlet s_n at a finite number of points, resulting in a piecewise-linear curve composed of zero-width line segments. We calculate the filtered image of each segment by considering those pixels in the filtered image that are within a distance R of the segment. The contribution of the line segment to a particular pixel in the filtered image can quickly be computed by a variant of the technique used by Feibush for polygon anti-aliasing [Feibush *et al*. 80]. The line segment is rotated about the pixel center so that it lies horizontally, and then two table-lookups based on the segment's endpoints are used to determine the kernel-weighted contribution to the pixel. We have found that a very coarse low-pass filtered image suffices to guide the placement of streamlines. Typically we use a filter kernel that extends just two or three pixels in radius. The filtered images in Figures 2, 3 and 4 were computed at a much higher resolution than this for expository purposes.

4 Long Streamlines

This section describes how the optimization technique from Section 3 can be extended to create images containing long, evenly-distributed streamlines. One goal of this procedure is to enable fine control over the distance between adjacent streamlines, whether that target spacing be constant-valued or position-dependent. A second goal is to avoid interrupting the streamlines. Since each endpoint of a streamline distracts from the visual flow of the image, our images should favor fewer, longer streamlines over numerous, shorter streamlets. It is not always possible to satisfy the two goals of uniform streamline separation and infrequent streamline breaks. In places where the vector field converges (*e.g.* near a sink) these two goals are at odds with one another. Our solution to the dilemma is to let the energy function be the arbiter between uniform spacing and long streamlines.

The optimization procedure for creating a hedgehog plot consists of repeatedly considering small changes to the positions of the streamlets, accepting only the changes that improve the measure E. The procedure for creating a set of longer streamlines s_n is similar. We improve a set S of streamlines by considering several kinds of changes to the streamlines. In addition to changes in a streamline's position, the algorithm also allows the operations of streamline insertion/deletion, lengthening/shortening of streamlines, and combination of two streamlines, end-to-end, into a single streamline. We use the same quality measure E to determine which changes will be accepted. In pseudo-code, the process for creating the collection S of long streamlines is as follows.

```
S ← null  { S begins as an empty set of streamlines }

{ find an initial group of streamlets }
foreach position (x,y) on a grid
  insert streamlet s at (x,y) into S to produce S'
  if E(S') < E(S) then
    S ← S'

{ improve the collection of streamlines in S }
repeat until accepted changes are rare
  choose an operation
  apply operation to random element(s) of S to produce S'
  if E(S') < E(S) then
    S ← S'
```

Figure 5b shows a collection of streamlines created with the above optimization procedure. The streamlines are evenly spaced and their endpoints are generally located where the vector field diverges or converges. For comparison, Figure 5a shows long streamlines whose seed points lie on a regular grid so that streamline density varies greatly.

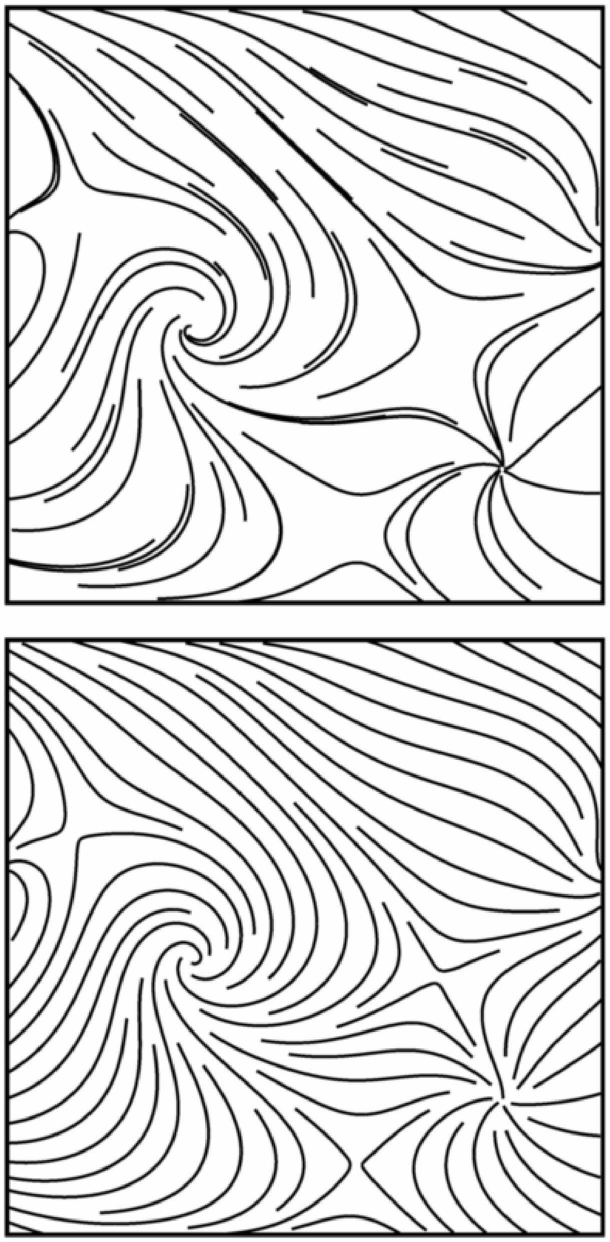
Figure 5: (a) Long streamlines with centers regularly placed on a grid (top); (b) Streamlines placed by density-based optimization (bottom). This data is a randomly generated vector field.

4.1 The Allowable Operations

The primitive streamline operations that we employ to improve the quality of an image are described in more detail below.

Move: Change the position of the seed point of the streamline. Each streamline is defined in terms of this seed point and a length to travel forward and backward through the flow.

Insert:	Create a new streamlet.
Delete:	Remove a streamline entirely from S.
Lengthen:	Add a positive value to the length of the streamline (relative to the seed point) in the forward or backward direction.
Shorten:	Subtract from the length of the streamline (relative to the seed point) in the forward or backward direction.
Combine:	Connect two streamlines whose endpoints are sufficiently close to one another. The location of the join is a weighted average of the two endpoints based on the relative lengths of the streamlines. The length of the new streamline is the sum of the lengths of the two parent streamlines.

Why do we allow so many kinds of changes during the optimization process? Presumably we could create any possible collection of streamlines using only **insert** and **delete** operations if we allow newly-inserted streamlines to assume any length and position. However, an actual implementation of the optimization process using such a restricted set of operations would be prohibitively slow to converge. We use the larger complement of operations so that the optimization procedure can move smoothly through the space of all collections of streamlines. For example, suppose that joining two particular streamlines would greatly improve the measure E. The optimization routine could choose at random to remove each of these streamlines and, also at random, create another streamline that fills the void left by the two that were removed. It is very unlikely that these three independent events would happen by chance. Explicitly providing a **combine** operation makes this small change in visual appearance much more likely to occur.

We find candidate pairs of streamlines for the **combine** operation by querying a data structure that maintains the positions of streamline endpoints and can return pairs of endpoints whose distance is less than a given tolerance. There are several ways in which we can favor joining together streamlines. We could add a term to the energy function that gives a higher energy to those images that contain more streamlines. Instead of this approach, however, we choose to accept **combine** operations if they result in a new value of E that is no greater than the old energy value plus a tolerance.

We can animate the optimization process by displaying the collection of streamlines every time a favorable change occurs. An animation of the optimization indicates the role of each operation. First, streamlets are inserted throughout the image. After this initial phase is finished the result looks much like a hedgehog plot using a jittered grid, reminiscent of a Poisson-disk distribution of points. Next, many of the streamlines gradually lengthen. As streamline endpoints approach one another, pairs of streamlines combine to form longer streamlines. This dual process of lengthening and joining creates many longer streamlines that typically follow nearly-parallel trajectories. Gradually the changes in the image become minor, and many of the changes at this stage are streamlines moving a small distance, evening the spacing between neighbors. Changes are accepted with decreasing frequency, and the process is terminated when accepted changes become sufficiently rare.

4.2 Acceleration Using an Oracle

The stochastic optimization produces good results, but it spends considerable time entertaining changes that are unlikely to improve the image. The method can be accelerated by using an *oracle* that suggests changes that are likely to decrease the energy function E. An oracle is only effective if it can be consulted quickly and its answers are generally reliable. The oracle described in this section typically speeds up the convergence of the optimization by a factor of three to five.

There are two systematic ways for an oracle to select changes to propose: an *image-based* approach, and a *streamline-based* approach. Our oracle uses a combination of the two. The image-based approach examines the blurred image B to identify places where the streamlines are too sparse. The oracle makes **insert** suggestions in these places. The streamline-based approach examines the neighborhood of each individual streamline to decide if an operation applied to the streamline is likely to improve the image. The oracle uses information gathered from around a streamline to decide whether to suggest a **lengthen**, **shorten** or **move** operation. More precisely, the oracle keeps a running measure of how “energetic” a given streamline is, and it maintains a priority queue that orders the streamlines based on their individual level of energy. When consulted, the oracle returns one of the most energetic streamlines, along with a suggestion of how to lessen its measure of energy.

The energy of a streamline is the sum on three factors: “desire” to lengthen, “desire” to shorten, and “desire” to move. Each of these factors is calculated by sampling the image B at a small number of positions near the streamline. The desire to lengthen is computed by comparing the target gray level t with the image values a short distance beyond the endpoints of the streamline. The lower these values are with respect to t, the greater the streamline desires to grow into this empty region. The desire to shorten is found by sampling B on either side of the streamline endpoints. If these values are too high, the streamline desires to shrink. The desire to move is computed by comparing the image values on one side of the streamline with the values on the other side. The greater the difference between these two values, the more the streamline desires to change its position. We typically consult 20 samples of the image B to determine each of the three factors that determine a streamline’s energy. This sampling is an inexpensive task in comparison to creating the entire path of a streamline and then low-pass-filtering the resulting curve.

The oracle need not bother suggesting that a streamline be deleted. Every time the optimization routine attempts to modify a streamline it can easily check whether entirely removing the streamline improves the total energy measure E of the image. This is done by evaluating E after the contribution of the streamline to the image B is removed and before the altered streamline’s effect is added to B.

The oracle is important for improving efficiency, but it is the energy measure E that drives the optimization. The oracle is used purely as a source of suggestions for how to reduce E, not as a source of directives that are applied blindly. The oracle’s suggestions are only accepted if the change improves the image quality. We have found it effective for the oracle to propose 50% of the changes, and for the other changes to be chosen completely at random. Thus *any* change to the collection of streamlines is possible, which makes it unlikely that the optimization will overlook a worthwhile improvement arising from any systematic bias of the oracle.

4.3 Intensity Tapering at Streamline Ends

Some streamlines must terminate within a region of converging flow or else the target density of the image cannot be preserved there.

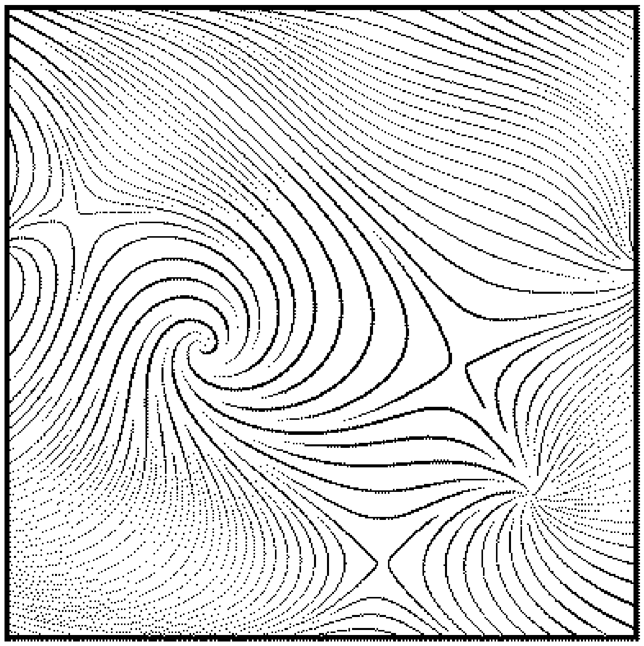

Figure 6: Field magnitude has been redundantly mapped onto streamline density and width. Large magnitude is indicated by dense, thin curves.

The resulting break of the streamline is visually jarring if it is rendered as a rectangular end cap. We make the termination less abrupt by gradually decreasing the width or intensity of the streamline near its endpoint. We can gently fade a streamline by allowing yet another operation, namely streamline *tapering*. Each streamline carries with it (in addition to its center and length) two positions along its length that indicate where to begin linearly fading to the background color at either endpoint. This intensity tapering is used to weight the contribution of the streamline to the filtered image *B*. Streamline tapering allows the optimization to find an even closer match to the ideal gray-scale value in regions near the streamline ends. In practice we have found it most effective to let intensity tapering be a separate optimization phase, after the streamlines have settled into their final position. In this final phase each streamline is allowed to perform only two operations: 1) changes in length, and 2) changes in the locations at which to begin intensity tapering. Performing the intensity tapering after long streamlines have been formed avoids the possibility that the optimization will produce many short, intensity-tapered streamlines to satisfy the target density. Figures 6 and 8 are rendered using the tapering information to modulate streamline width and intensity, respectively.

Saito and Takahashi have demonstrated a similar tapering effect for drawing contour lines of a scalar field [Saito & Takahashi 90]. They use information about the gradient of the scalar field to guide the fading out of the contour lines. Their technique can also be used for drawing streamlines of vector fields where the divergence is zero everywhere, but it has no obvious generalization when the divergence is non-zero (*e.g.* fields with sources and sinks).

4.4 Optimization Issues

Two recent techniques in computer graphics provided inspiration for the optimization approach described here. The first of these is the work by Andrew Witkin and Paul Heckbert for distributing particles over an implicit surface [Witkin & Heckbert 94]. In their constrained optimization method, they let a small number of seed particles repel one another in order to distribute themselves evenly over a surface. They found that it is helpful to allow the initial particles to grow, split, shrink or die to accommodate any change in surface area when the surface geometry is being edited. Their operations on particles are analogous to our operations on streamlines.

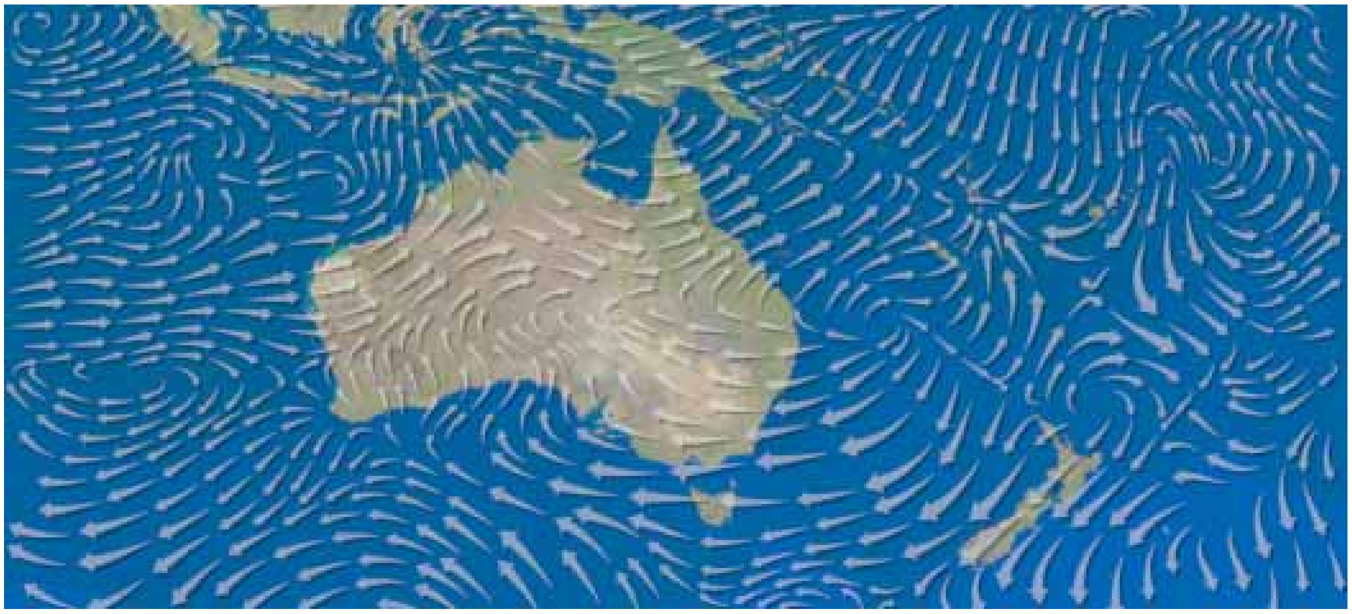

Figure 7: Chains of arrows indicate wind direction and magnitude over Australia. The arrows were deposited along streamlines created by streamline optimization. Higher velocity is indicated by larger arrows. The vector field data was calculated using a numerical weather model.

A second source of inspiration was the mesh optimization work by Hugues Hoppe and co-workers [Hoppe *et al.* 93]. Their technique uses three fundamental operations to automatically simplify a polygonal mesh: edge split, edge collapse, and edge swap. They used an energy measure to guide the optimization by random descent. The high quality of the results produced by this method encouraged us to try random descent in streamline optimization.

One frequently-voiced concern about optimization techniques is that the behavior of the system is highly sensitive to the values of many parameters. An example of such a parameter for streamline optimization is the maximum distance a streamline can move. The fear is that the system may require a large amount of "parameter tweaking." Happily, we have found it unnecessary to change our parameter settings between datasets. The single parameter that we specify for an illustration is the desired distance between neighboring streamlines (which can even be position-dependent). Other parameters are derived from this target-distance. We believe that researchers who implement the techniques described here will not have difficulty replicating our results. To relieve the burden of re-implementing our technique, we are making our source code publicly available at http://www.cs.msstate.edu/~banks/IGSP.

5 Binding Visual Attributes to Streamlines

There is an important distinction between a streamline (a zero-width integral curve) and the geometric elements associated with its display. A simple approach for displaying a streamline is to draw an anti-aliased curve that connects vertices sampled along the streamline, but such a constant-width, constant-intensity curve is not necessarily the best way to visualize the flow. For example, the curves are unchanged if all the vectors reverse direction in the underlying vector field; that is, the sense of flow direction is ambiguous in a simple streamline display. Arrows can be inserted into the image to disambiguate the flow direction. We apply two different techniques to bind arrow-shaped glyphs to streamlines. The first technique is to traverse the streamlines and deposit an arrow whenever the integrated arc-length along a streamline is sufficient to accommodate the arrow's length. Such an object-order traversal is appropriate for binding a long chain of glyphs onto a streamline. The second approach is to distribute arrow-glyphs uniformly throughout the image and then snap them to the nearest point on a streamline. Such an image-order traversal is appropriate for images with only a few scattered arrows serving as reminders of the flow direction.

Often some important scalar quantity is associated with a vector field. The scalar value might be the temperature or density in a fluid flow, or it might be the magnitude of the vector field at each point. We would like to bind visual attributes to display such a scalar quantity along with the streamlines. The thickness and the grayscale-intensity of a streamline offer two convenient visual attributes to convey a scalar quantity. Figure 6 shows a vector field whose magnitude is bound to the width of the streamlines and where the streamlines themselves have been placed so that the scalar field determines the distance between neighboring streamlines.

6 Results

In this section we show some examples of images constructed using the optimization method for positioning long streamlines. The first example is Figure 1b, which illustrates a numerical simulation of flow around a cylinder. The arrowheads in this figure disambiguate flow orientation in the eddies. Figure 7 shows computed wind velocity in the vicinity of Australia. First, the long streamline optimization method placed streamlines through the image. Then arrows were bound to these streamlines. The size of the arrow indicates the wind magnitude. The arrows line up head-to-tail so that the eye can easily follow from one to the next, as is favored by illustrators [Bertin 83]. Human-subject studies have shown that if a graphical stroke varies in width from large to small, people have a strong sense that the direction is towards the larger end [Fowler & Ware 89]. This guided our choice of tapered arrows in Figure 7.

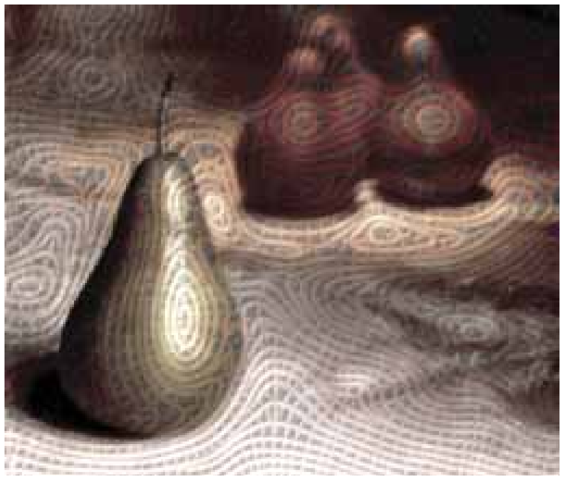

Figure 8: "Pears." The texture in this image was created by combining streamlines in two directions: along the gradient of the blurred intensity and at 90 degrees to the gradient. Original photograph courtesy of Herb Stokes.

Another application of the streamline-placement technique is to create iso-intensity contours that are evenly spaced. Consider the effect of highlighting several discrete intensity levels in a gray-scale image: even if the intensity-values are chosen in equal increments, the resulting contours are likely to clump together in some regions and spread apart in others. Our optimization technique provides a convenient way to adaptively sample the intensity values so that the curves are uniformly distributed in the image. Figure 8 shows how the technique can be applied to a color photograph. We converted the image to monochrome, blurred it, and then calculated its gradient vector field. We ran the optimization on the gradient vector field and on a vector field orthogonal to it (and thus aligned with the iso-value lines). The two sets of streamlines that resulted were combined and used as a mask to apply the original color values to the grayscale image. The effect is akin to weaving, with constant-intensity thread being used along the contours.

Our streamline optimization program was written in C++, and the calculations for the figures herein were performed on a Silicon Graphics Indigo2 with an R4400 processor operating at 250 MHz. Figures 4 (a) and 5 (b) were created in under one minute, and the streamlines for Figures 6, 7 and 8 required roughly 15 minutes each. We expect that fine-tuning the code would improve the speed by a factor of two to four.

7 Conclusion and Future Work

There are several logical extensions to the streamline optimization method presented in this paper. This same process can be used to create streamlines on curved surfaces by running the optimization in the parametric space of the surface and correcting for mapping distortions. The technique could also be used to create streamlines in three dimensions, although computational efficiency will probably become an issue. The density of 3D streamlines could be made dependent on additional properties of the vector field, such as proximity to vortex cores. Another research area is in creating illustrations that reveal different levels of detail when the viewer is at various distances.

We expect that the notion of guiding the placement of graphical elements by a visual measure of quality will have applications beyond vector field visualization. For instance, a similar optimization method might prove useful in placing graphical elements in a texture. Another potential use for such techniques is for computer generation of illustrations that have a hand-drawn appearance [Saito & Takahashi 90] [Winkenbach & Salesin 94].

8 Acknowledgments

We thank Glenn Wightwick of IBM Australia and Lloyd Treinish of the IBM T. J. Watson Research Center for the Australia wind data. Earth image is courtesy of Geosphere, Inc. The fluid flow data of Figure 1 was provided courtesy of David Rudy, NASA Langley Research Center. We thank Peggy Wetzel and Mary Whitton for help in making video of this work. Funding for this work was provided in part by the NSF Science and Technology Center for Computer Graphics and Scientific Visualization. Travel support was provided by ICASE and the NSF Engineering Research Center at MSU.

9 Bibliography

[Bertin 83] Bertin, Jacques, *Semiology of Graphics*, translated from French, The University of Wisconsin Press 1983.

[Bryson &Levit 91] Bryson, Steve and Creon Levit, "The Virtual Wind Tunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows," *Proceedings Visualization '91*, San Diego, California, October 22–25, pp. 17–24.

[Cabral & Leedom 93] Cabral, Brian and Leith (Casey) Leedom, "Imaging Vector Fields Using Line Integral Convolution," *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH '93), pp. 263–270.

[Crawfis & Max 92] Crawfis, Roger and Nelson Max, "Direct Volume Visualization of Three Dimensional Vector Fields," *Proceedings of the 1992 Workshop on Volume Visualization*, pp. 55–60.

[de Leeuw & van Wijk 95] de Leeuw, Willem C., and Jarke van Wijk, "Enhanced Spot Noise for Vector Field Visualization," *Proceedings Visualization '95*, Atlanta, Georgia, Oct. 29 – Nov. 3, pp. 233–239.

[Delmarcelle & Hesselink 94] Delmarcelle, Thierry and Lambertus Hesselink, "The Topology of Symmetric, Second-Order Tensor Fields," *Proceedings Visualization '94*, Washington, D.C., October 17–21, pp. 140–147.

[Dovey 95] Dovey, Don, "Vector Plots for Irregular Grids," *Proceedings Visualization '95*, Atlanta, Georgia, Oct. 29 – Nov. 3, pp. 248–253.

[Feibush *et al*. 80] Feibush, Eliot, Marc Levoy and Robert Cook, "Synthetic Texturing Using Digital Filters," *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH '80), pp. 294–301.

[Feynman 64] Feynman, Richard P., Robert B. Leighton and Matthew Sands, *The Feynman Lectures on Physics*, Addison-Wesley, Reading, Massachusetts, 1964.

[Forssell 94] Forsell, Lisa K., "Visualizing Flow over Curvalinear Grid Surfaces Using Line Integral Convolution," *Proceedings Visualization '94*, Washington, D.C., October 17–21, pp. 240–247.

[Fowler & Ware 89] Fowler, David and Colin Ware, "Strokes for Representing Univariate Vector Field Maps," *Graphics Interface '89*, London, Ontario, June 19–23, 1989, pp. 249–253.

[Globus *et al*. 91] Globus, A., C. Levit and T. Lasinski, "A Tool for Visualizing the Topology of Three-Dimensional Vector Fields," *Proceedings Visualization '91*, San Diego, California, October 22–25, pp. 33–40.

[Helman & Hesselink 91] Helman, J. L. and L. Hesselink, "Visualization of Vector Field Topology in Fluid Flows," *IEEE Computer Graphics and Applications*, Vol. 11, No. 3, pp. 36–46.

[Hoppe *et al*. 93] Hoppe, Hugues, Tony DeRose, Tom Duchamp, John McDonald and Werner Stuetzel, "Mesh Optimization," *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH '93), pp. 19–26.

[Kundu 90] Kundu, Pijush K., *Fluid Mechanics*, Academic Press, Inc., San Diego, 1990.

[Max *et al.* 94] Max, Nelson, Roger Crawfis and Charles Grant, "Visualizing 3D Velocity Fields Near Contour Surfaces," *Proceedings Visualization '94*, Washington, D.C., October 17–21, pp. 248–255.

[Saito & Takahashi 90] Saito, Takafumi and Tokiichiro Takahashi, "Comprehensible Rendering of 3-D Shapes," Computer Graphics, Vol. 24, No. 4 (SIGGRAPH '90), pp. 197–206.

[Salisbury *et al.* 94] Salisbury, Michael P., Sean E. Anderson, Ronen Barzel and David H. Salesin, "Interactive Pen-and-Ink Illustration', *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH '94), pp. 101–108.

[Stalling & Hege 95] Stalling, Detlev and Hans-Christian Hege, "Fast and Resolution Independent Line Integral Convolution," *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH '95), pp. 249–256.

[Turk 91] Turk, Greg, "Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion," *Computer Graphics*, Vol. 25, No. 4 (SIGGRAPH '91), pp. 289–298.

[van Wijk 91] van Wijk, Jarke J., "Spot Noise: Texture Synthesis for Data Visualization," *Computer Graphics*, Vol. 25, No. 4 (SIGGRAPH '91), pp. 309–318.

[Winkenbach & Salesin 94] Winkenbach, Georges and David H. Salesin, "Computer-Generated Pen-and-Ink Illustrations," *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH '94), pp. 91–98.

[Witkin & Heckbert 94] Witkin, Andrew and Paul Heckbert, "Using Particles to Sample and Control Implicit Surfaces," *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 94), pp. 269–277.