Directional Enhancement in Texture-based Vector Field Visualization

Francesca Taponecco [*]
GRIS Dept.
Darmstadt University

Timothy Urness [†]
Mathematics and Computer Science Dept.
Drake University

Victoria Interrante [‡]
Computer Science and Engineering Dept.
University of Minnesota

Abstract

The use of textures provides a rich and diverse set of possibilities for the visualization of flow data. In this paper, we present methods designed to produce oriented and controlled textures that accurately reflect the complex patterns that occur in vector field visualizations. We offer new insights based on the specification and classification of neighborhood models for synthesizing a texture that accurately depicts a vector field. Secondly, we introduce a computationally efficient method of texture mapping streamlines utilizing outlining textures to depict flow orientation.

Keywords: Texture Synthesis, Flow Visualization

1 Introduction and Motivation

Textures have traditionally been used to visualize vector fields for the purpose of analyzing the form and behavior of flow consistent with theoretical models and to infer the underlying behavior of experimentally-generated flow fields. The use of textures allows for a consistent and highly-detailed representation of a vector field, allowing an observer to both analyze and better understand fluid dynamics.

Line Integral Convolution (LIC) [Cabral and Leedom 1993] is the predominantly used technique in 2D flow visualizations. While quite effective, LIC textures represent only one particular choice from among the vast spectrum of possible texture patterns that could be used to convey flow information. By exploring methods for conveying flow data using features derived from natural texture patterns, we seek to profoundly expand the range of possibilities in texture-based flow representation, and to introduce the possibility of using texture type itself as a visual variable for conveying information about the flow.

In addition to traditional texture synthesis, recent research have provided algorithms in which textures can be deformed, curved, modified and controlled to follow given directions and orientations. In these techniques, a variety of deformation operations must be used to align a given pattern along a newly specified direction. Current texture synthesis techniques, however, do not adequately highlight the anisotropy of a texture when synthesized in this fashion. We propose the use of a new family of specified neighborhood models which can significantly improve the results of texture synthesis along vector fields. Additionally, we introduce a computationally efficient method of texture mapping streamlines utilizing outlining textures to depict flow orientation.

In the following, we review related work which deals with non-homogeneous textures, oriented textures, and textures applied to vector field visualizations. We then introduce new classes of neighborhood models and weighting functions for enhancing vector field direction in a synthesized, texture-based visualization (section 3). In section 4, we present an efficient, texture mapping technique to create an effective visualization of a flow field. In section 5, we compare the two complimentary techniques. Finally, we conclude addressing possible future extensions.

2 Related work

There have been many texture-based techniques developed for flow visualization. Spot Noise [van Wijk 1991] synthesizes a high-resolution image by deforming an input image of random spots according to the vector field. LIC [Cabral and Leedom 1993] convolves a white-noise input texture according to calculated streamlines. There have been several extension to these original methods improving running time and image quality [de Leeuw and van Wijk 1995; Stalling and Hege 1995]. Effective visualization of 2D fluid flow is also possible by techniques that animate textures in accordance with the advection of a given flow field [van Wijk 2002; Wang et al. 2005].

Texture synthesis methods allow for an unlimited quantity of texture patterns to be generated that are perceptually equivalent to a sample input texture. The classical algorithms [Efros and Leung 1999; Wei and Levoy 2000] construct the output image in a pixel-by-pixel scan line order in which a pixel is synthesized by comparing a similarly shaped neighborhood with the original sample. Along these lines, Ashikhmin [Ashikhmin 2001] presents a technique for synthesizing natural textures based on repeating patterns consisting of small objects of familiar but irregular sizes such as flowers and pebbles. Efros and Freeman [Efros and Freeman 2001] introduce image quilting to produce an effective texture synthesis pattern for a wide variety of input textures. This works by essentially piecing together small patches of the input texture which allows the output to maintain both continuity and coherence across the entire pattern. Approaches for synthesizing textures on surfaces have been proposed adapting the direction field for the texturing [Turk 2001], and seamlessly covering the surface with patches [Gorla et al. 2002].

Taponecco and Alexa [Taponecco and Alexa 2003] introduced an algorithm that used a Markov model approach to synthesize a texture according to a flow field. The method also has been extended to the temporal domain, permitting the visualization of unsteady vector fields. This technique uses a strongly directional texture that is rotated according to the vector field. The resulting image is created in a pixel-by-pixel manner using Markov Random Field (MRF) texture synthesis. Verma, Kao, and Pang presented a method for generating LIC-like images through texture-mapped streamlines called PLIC (Pseudo-LIC) [Verma et al. 1999]. By experimenting with different input textures, both LIC-like images and streamline-like images can be produced. Shen, Li, and Bordoloi [Shen et al. 2004]

[*]e-mail: francesca.taponecco@gris.informatik.tu-darmstadt.de
[†]e-mail: timothy.urness@drake.edu
[‡]e-mail: interran@cs.umn.edu

utilize texture mapping in addition to analysis of three dimensional geometric properties to volume render three-dimensional unsteady flow fields.

3 Texture Synthesis Approach

Techniques that control texture generation for vector field visualization use deformation operations to shear, curve and align a given pattern along specified directions (figure 1). These operations necessarily modify the texture color, resolution, and appearance in general. This procedure adapts to varying conditions by using different input samples at different output orientations at each step in the algorithm [Taponecco and Alexa 2003].

Using a standard texture synthesis approach, however, does not adequately highlight the isotropy that may be displayed in the sample input texture. For this reason, we build a new class of neighborhoods, based on different pixel weighting functions. In this section, novel schemes to non-uniformly weight the pixels of the neighborhoods are introduced and used during the synthesis process. Elliptical schemes result to be particularly beneficial to preserve and enhance the directionality of the samples; the goal is to provide an accurate synthesis of controlled textures along vector fields, using such novel specifications to maintain good resolution while changing the pattern alignment.

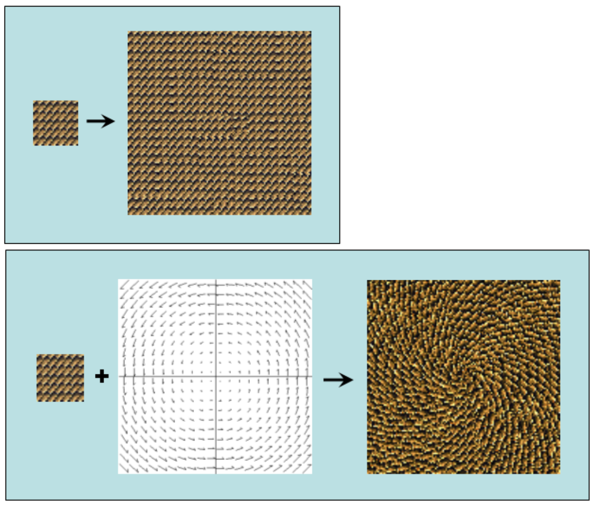


Figure 1: Standard texture synthesis (top). Controlled texture synthesis (bottom). The controlled, field-driven synthesis algorithm requires the input pattern to be synthesized along new directions.

3.1 Synthesis Neighborhoods

3.1.1 Standard squared and L-shaped neighborhoods

In standard pixel-based texture synthesis [Efros and Leung 1999; Wei and Levoy 2000], the output texture is commonly generated in raster scan order using either the *squared* or *L-shaped* neighborhood structures. These structures are illustrated in figure 2.

Since the main problem in standard texture synthesis is to reproduce the characteristics of a small texture sample in a larger domain, the neighborhood kernel must include sufficient surrounding information about the pixels in order to reproduce the given pattern in a perceptually similar way. Consequently, the neighborhood size is crucial and directly depends on the pattern structure complexity. The neighboring pixels around the current point serve to build an array of color values, which have to be compared to similar neighborhoods, leading to the measurement of a probability. Such a measure, usually a distance function based on the $L2-$norm, is used to find the neighborhood that is most similar for the selection of the best matching pixel.

Experimentations with different kernel sizes indicate that the output image quality is largely dependent on finding a suitable neighborhood size for both the sample pattern structure and the vector field. The image quality is often enhanced using longer kernel lengths, as the structure characteristics of the sample pattern can better reproduce the statistics of the texture. However, a large kernel size can be detrimental in the case of vector field that contains strong curvatures (short curvature radius), as the resulting texture will have significant deviations from the vector field. Often times, a compromise is required to obtain the best kernel size to accurately reflect the texture sample and the vector field.

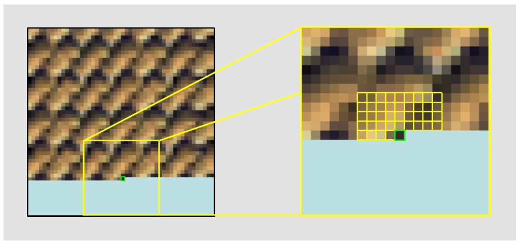


Figure 2: Standard texture synthesis: the enlarged picture shows the L-shaped "causal neighborhood" structure (yellow) around the current pixel to synthesize (green). The light-blue portion of the image still needs to be completed.

3.1.2 Different neighborhood shapes

In the course of our research, we have examined several novel shapes (such as rectangular, half-rectangular, trapezoidal, or rhomboidal) for neighborhoods that could possibly better communicate directional characteristics than a squared (or L-shaped) neighborhood. Insights from these experiments have lead to the use of *asymmetric* instead of symmetric neighborhoods. The asymmetric neighborhoods lead to improvements, stressing orientation in addition to directionality, or highlighting topological features extracted from the vector data. This also can be beneficial for the synthesis of textures whose structure presents a secondary minor direction in addition to its principal anisotropic direction. For these textures, an asymmetric (*e.g.* trapezoidal) weighting scheme is able to account for the two existing pattern directions. Although it is straightforward to model and use irregular structures, we mainly prefer, in order to maintain algorithm consistency, to use regularly-shaped neighborhoods (eventually setting the weights for the edges/corners to be zero), and to use the weighting schemes introduced later to simulate different behaviors and neighborhood shapes.

3.2 Non-uniform neighborhood filtering

3.2.1 Anisotropic neighborhood model specifications

Using a MRF-based approach to vector field visualization, we use samples that are weighted heavily or oriented in a principal direction. The method is easily adaptable to rotations and changes of curvature that often occur in a vector field. In order to maintain and enhance the properties of the vector field during the synthesis, the pixels that build a neighborhood should not be uniformly weighted (as in the standard texture synthesis case). An anisotropic weighting scheme is better suited to preserve the directionality and to enhance continuity along the field direction.

3.2.2 Bilateral filtering

Bilateral filtering has been introduced by Tomasi and Manduchi [Tomasi and Manduchi 1998] and later applied in several applications, see for instance [Durand and Dorsey 2002]. It is a non-iterative scheme for edge-preserving smoothing.

The basic idea of bilateral filtering is to operate in the *range* of an image in the same manner traditional filters do in the *domain*. Two pixels can be close to one another by occupying nearby spatial location, or they can be similar to one another by having nearby values in a perceptually meaningful fashion. Closeness refers to vicinity in the *domain* (geometric closeness), similarity refers to vicinity in the *range* (photometric similarity). A possibility for measuring pixel similarity is to use the CIE-Lab color space [CIE CIE-Lab System 1976], which enables a measure of color similarity that is correlated with human color discrimination performance.

The bilateral filter was designed to maximally suppress image noise with minimal impact on the underlying signal image [Tomasi and Manduchi 1998]. The kernel of a bilateral filter is composed of an inner product of two low-pass filters in real space. The first is a normal low-pass filter, which averages the neighboring pixel values with decreasing weights for pixels at larger distances. The second kernel is also a type of low-pass filter, but the weights for the neighboring pixels are derived from the pixel value differences from the center pixel, instead of being related to geometric distances. The larger pixel difference, the smaller the pixels' contribution during filtering, resulting in a measure of similarity.

We utilize concepts from bilateral filtering in our weighting schemes approach as illustrated below. The feature of edge-preserving smoothing is particularly advantageous to enhance directionality in texture-based vector field visualization. We use the measures of Euclidean distances (*domain*) and intensity differences (*range*) as similarity metrics. The bilateral filter kernels take a form that depend on the weighting function. We use the Gaussian case; hence, the combination of the two filtering operators (product of Gaussians) leads to coefficients that fall off with distance and dissimilarity from the central pixel of the weighted neighborhood.

3.2.3 Gaussian filtering and kernel coefficients

Gaussian low-pass filtering computes a weighted average of pixel values in the neighborhood, in which the weights decrease with distance from the center. To derive the kernel coefficients, digital filters can be designed using a *direct* (FIR) or *recursive* (IIR) form. The direct form is obtained as a finite number of samples of the desired impulse response. The recursive form is designed as a ratio of polynomials in the z domain. Further, *binomial filters* provide a third method for designing Gaussian filters; they are obtained with cascaded convolution of a kernel filter composed of [1,1] (auto-convolution). The set of filter coefficients is known as binomial series, and can be computed using the *Pascal's triangle*:

$$\begin{array}{ccccccccccc} & & & & & 1 & & & & & \\ & & & & 1 & & 1 & & & & \\ & & & 1 & & 2 & & 1 & & & \\ & & 1 & & 3 & & 3 & & 1 & & \\ & 1 & & 4 & & 6 & & 4 & & 1 & \\ 1 & & 5 & & 10 & & 10 & & 5 & & 1 \end{array} \tag{1}$$

where the coefficients $C_{n,k}$ (n representing the row in the triangle, and k the coefficient count in the row) are given by:

$$C_{n,k} = \binom{n}{k} = \frac{n!}{k! \cdot (n-k)!} \tag{2}$$

$$C_{n,k} = \begin{cases} 1 & if \quad k = 0, k = n \\ C_{n-1,k-1} + C_{n-1,k} & \forall \quad k \neq 0, n \end{cases} \tag{3}$$

3.3 Isotropic and anisotropic weighting schemes

As opposed to storing the pixels of the neighborhoods in a uniform fashion, it is possible to adopt non-uniform weighting scheme to give more relevance to pixels near to the center (circular isotropic weighting scheme) or to particularly enhance directionality properties in the synthesis (elliptical anisotropic weighting scheme).

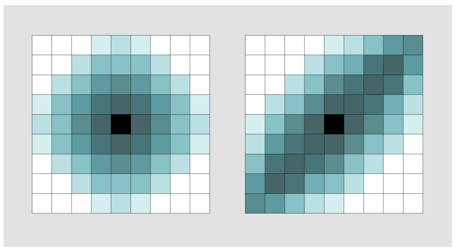


Figure 3: Circular and elliptical weighting schemes.

3.3.1 Circular neighborhood

The *circular neighborhood* model can enhance the information contained in the current pixel in a simple way, letting the value of the surrounding pixels decrease with respect to the distance from the center.

Unlike standard texture synthesis, this approach (seen as field-driven texture synthesis) constantly uses different input samples [Taponecco and Alexa 2004]. These samples are commonly modified versions of an original input pattern, which are derived from different conditions that depend on the control vector field. Thus, the neighbor search for the best match is dynamic; the best fitting pixel selection does not constantly take place inside the same input sample and therefore the pixels chosen to synthesize the output could produce visual artifacts in the resulting texture.

To avoid these artifacts, we use a pixel-centered approach where the weighting function considers the pixels in a circular manner. The values of the pixels near the pixel being considered are given a stronger weight, while the edges and corners are assigned a less significant weight. In case of strong curvatures in vector field, the pixels in the neighborhood of the pixel under consideration still contribute to describe the field direction, while the pixels at the corners or edges of the neighborhood are derived from a previous synthesis using rotated versions of the neighborhood. The influence of the pixels on the outside of the filter needs to be constrained as they could lead to discontinuities in the resulting image. This motivates the synthesis algorithm to not operate in scanline order, as is traditionally done, but instead in the direction given by the vector field [Gorla et al. 2002]. This ensures that the the algorithm maintains the pattern continuity in the principal direction.

The functions that are most suited for such weighting are monotonically descending functions, such as Gaussian functions or decreasing exponential functions, where the decay factor can be specified in dependence of the field velocity and curvature radius. Again, there exists a tradeoff between computation efficiency and image accuracy. A simple radially symmetric weighting function using a 2d Gaussian leads to the weighting scheme illustrated in figure 3 (left).

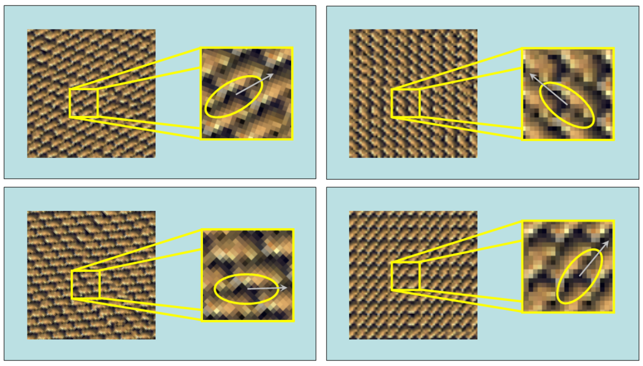


Figure 4: Directional enhancement using the elliptical weighting scheme.

3.3.2 Elliptical neighborhood

Next, we adapt the circular neighborhood structure to a more directionality sensitive one: the elliptical structure. We specify this family of neighborhood models for use in controlled oriented texture synthesis. The relative structure is elliptically shaped (figure 3 right). In other words, the weights of the neighboring pixels are weighted using an *elliptical scheme*. Similar to the circular neighborhood, the intensity of the weights decreases from the central pixel to the external borders of the neighborhood. In contrast to the circular neighborhood, the pixel weights do not isotropically decrease from the center. Instead, the weights more rapidly decrease in the direction of the minor axis and more slowly in the major axis direction of the ellipse. The major axis of the elliptical neighborhood is oriented in the direction of the vector field that defines the new pattern orientation and controls the deformation of the resulting textured image (figure 4). Orientating the major direction of the elliptical neighborhood along the deformation or the vector field allows for a better control the output texture generation. This direction is given by the phase of the control vector field.

The range of values, angles, and orientations depend on the size of the neighborhood structure. Smaller neighborhood models can define just a few independent directions, as the two principal axis cannot greatly differentiate in orientation if the size is limited. On the contrary, neighborhoods characterized by having large size can better distribute anisotropic weights, specifying several possible directions and orientations for the major axis. The amount of such degrees of rotation is directly related to the radius of the neighborhood model.

In figure 5, we show for simplicity the case of a 3-sized neighborhood, where possible orientation spanning in the range $[0°, 360°]$ and where the 4 principal orientations for elliptical weighting are circled. Note that in case the original pattern only presents characteristics of directionality, the orientations that differ by 180 degrees are identical. In case the original pattern additionally presents directionality (*e.g.* directional glyphs), we can also distinguish between opposite aligned cases.

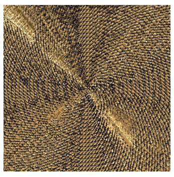
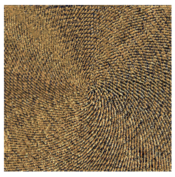
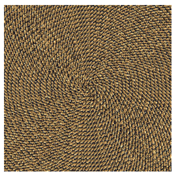
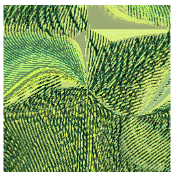
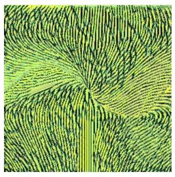
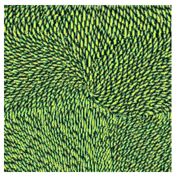
In	squared	circular	elliptical
			
			

Table 1: Controlled texture synthesis (fabric and green-scale patterns) along curved vector fields. Using a five-sized kernel, our elliptical weighting scheme (right) shows significant improvements with respect to the circular and the standard squared neighborhood.

Table 1 shows a comparison of results obtained using the different neighborhood structures. For each example, we generated the textures using the same input sample and the same neighborhood size. The anisotropic *elliptical* weighting scheme (right column) proves to achieve significant improvements in the quality of the images with respect to the standard approach based on squared neighborhoods (left) and also to the circular scheme (middle). We found that elliptical and circular schemes typically produce more detailed images with fewer artifacts than the squared neighborhood. Additionally, we can say that the more directional the pattern the more significant improvements are achieved with the elliptical scheme.

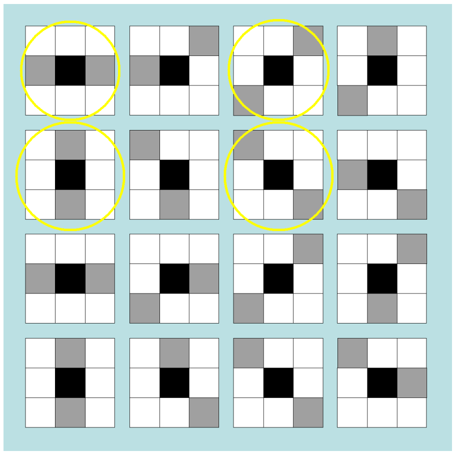


Figure 5: Anisotropic weighting of a 3-sized neighborhood, where grey pixels are assigned stronger weights to stress the direction of the vector field. The 4 orientation instances circled in yellow are usually sufficient to produce good results.

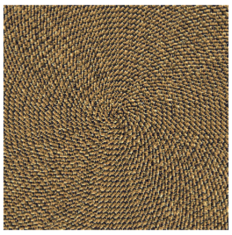
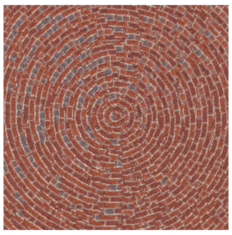
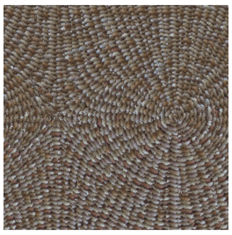
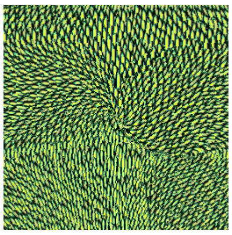
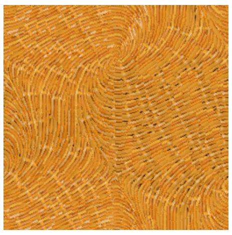
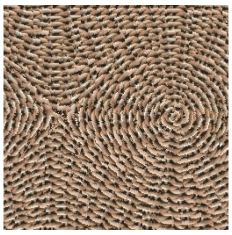
			
			
			

Table 2: Examples of a variety of textures synthesized for the effective visualization of flow fields.

It is necessary to note that good results can be obtained using standard neighborhood structures as well (examples can be seen in some previous work [Taponecco and Alexa 2004]); however, in order to achieve results with the same quality of those obtained with the elliptical scheme, larger neighborhoods are needed, leading to longer computational times and complexity.

4 Texture Mapping Streamlines

The computational complexity of a texture synthesis algorithm, particularly with a large search neighborhood, can be significant. In this section, we present an alternative to texture synthesis, *texture mapping*, for the display of textures for the representation of a vector field. We also examine using isotropic textures for flow visualization and the capabilities of textures and texture attributes to represent vector fields and correlated data, such as vector magnitude.

4.1 Geometry

Applying a texture to a streamline requires that the streamline be extended to include width as textures are inherently 2D and do not project well to streamlines or single pixels [de Leeuw and van Wijk 1995; Verma et al. 1999]. We construct a *thick streamline* by first calculating a 1D streamline given the vector field that defines the flow to be visualized. The streamline is then given width by considering the normal component to the streamline at each point. A user-specified width is multiplied by the normal component to give the location for each point of the thick streamline. This amount is added to both sides of every pixel in the streamline resulting in a thick, 2D streamline (figure 6). The coordinates of the thick streamline are used to construct polygons in which a texture can be easily applied using standard texture mapping techniques. Segmenting the thick streamline into polygons allows a texture-mapped streamline to effectively bend and curve in any direction.

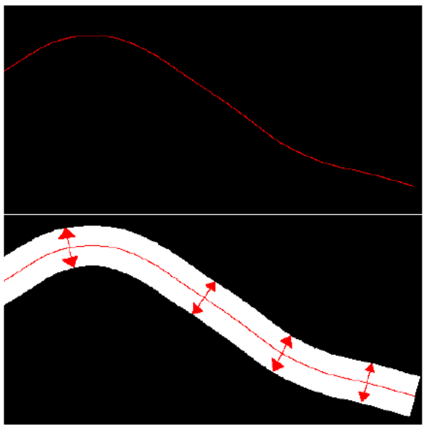


Figure 6: A thick streamline is constructed by calculating a 1D streamline (top). The normal component at each point in the streamline is multiplied by a user-defined width to calculate the coordinates for the thick streamline (bottom).

An adaptive step size is used during the streamline integration computation to construct polygons that can effectively represent the streamline around areas of high curvature [Stalling and Hege 1995]. Using the fourth-order Runge-Kutta formula and given a user-defined error tolerance, an adaptive step size approach chooses a large enough step size to define each polygon while observing the tolerance specified by the user. The effect of this approach is that smaller polygons are generated in areas of high curvature (figure 7).

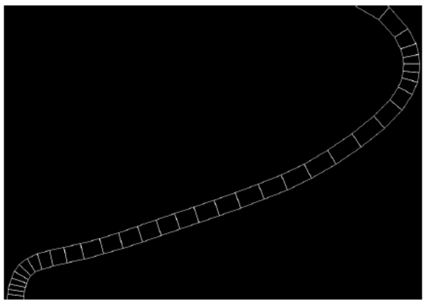


Figure 7: Polygons are generated according to an adaptive step-size algorithm that allows for smaller polygons to be generated around areas of high curvature.

4.2 Flow Fields

Controlling streamline density allows an entire field of thick streamlines to be created and equally spaced so that applied textures can be perceived. Turk and Banks [Turk and Banks 1996] first address the issue of streamline density. We employ the algorithm developed by Jobard and Lefer [Jobard and Lefer 1997] for ease of implementation and efficiency purposes.

Once an initial seed point in the 2D domain is selected randomly, a streamline is calculated in both the positive and negative direction.

The streamline is traced in each direction until one of the following occurs: a singularity is reached, the streamline reaches the edge of the domain, or the streamline comes within some user-defined distance of another streamline that has already been calculated. A new seed point is generated by randomly selecting a point on the defined streamline and moving it a distance greater than the width of the thick streamline in the direction normal to the flow at this point. Controlling the distance of a new seed point from the previous streamline allows flexibility in the density of the streamlines of the resulting image. To generate an image of dense streamlines, the seed point should be at a distance that is approximately the thick streamline width from the previously defined streamline. A distance that is greater than the streamline width will create more space between streamlines and result in a sparse final image. The algorithm continues by placing seed points in this manner until no more valid seed points can be found.

We have found it beneficial to construct streamlines with maximum possible length when creating the final images presented. Streamlines that do not have a sufficient length are not displayed and another seed point is calculated.

Several artifacts can occur when texture mapping streamlines. To avoid artifacts that may occur with a repeated texture on the same streamline, a sufficiently large texture sample is used. To avoid artifacts that may occur with repeated texture being applied at the same interval on neighboring streamlines, a random texture offset is used when constructing the first texture-mapped polygon of the thick streamline. Additionally, where portions of streamlines overlap, pixels are assigned an opacity value of zero, giving priority to streamlines already defined. The result is the ability for streamlines to effectively "merge" due to convergence or divergence of the flow but not to obstruct a previously placed streamline (figure 8).

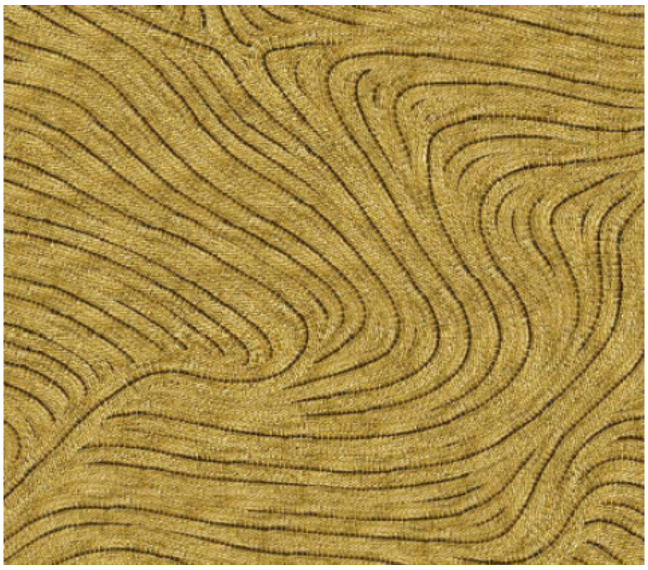


Figure 8: Using texture-mapped thick streamlines to visualize a flow field.

4.3 Texture Outline

Texture mapping a texture to a field of thick streamlines may not create an effective visualization if the orientation of the applied texture is not obvious. Figure 9 (top) shows the result of applying an isotropic texture to streamlines and the ambiguous orientation of

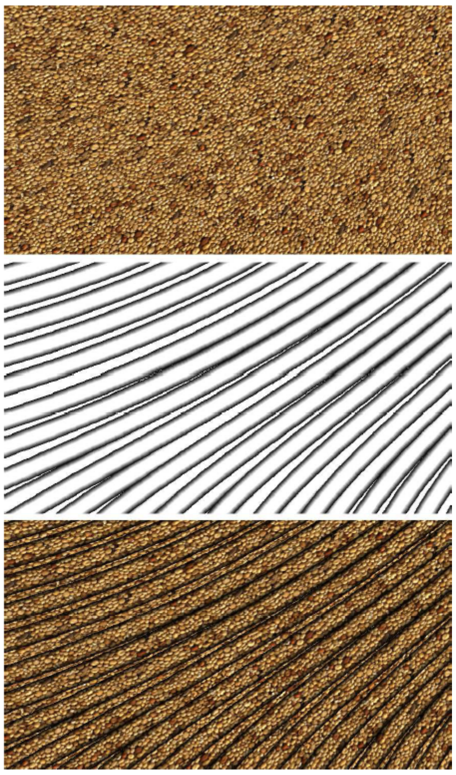


Figure 9: An illustration of texture outlining used to disambiguate streamline orientation. Top: a birdseed texture applied to streamlines. Middle: the outlining texture applied to streamlines. Bottom: combination of the above two images.

streamlines that results. The orientation of the streamlines can be specified by combining the texture with an outline of the calculated streamlines. The outline of the streamlines is constructed by mapping an *outlining texture* to the calculated streamlines defined by the vector field. The outlining texture consists of a luminance ramp, from black to white, emanating from each side of the texture. The intention is to mimic a diffuse lighting effect that would be created if the thick streamline were three dimensional and tubular in shape. The effect of applying this outlining texture to streamlines is displayed in figure 9 (middle). Finally, the two images can be overlaid allowing the orientation of the flow field to be displayed (figure 9 bottom).

It is important to limit the proximity between streamlines in creating the outlined streamlines texture. If the outlines of thick streamlines are allowed to overlap, areas of high overlap produce distracting regions when the luminance ramp of the streamline is abruptly halted where one streamline overlaps another. To avoid this artifact, the computation of a streamline is stopped if it approaches another streamline within one half the width of the thick streamline.

4.4 Texture Attributes

We have the ability to show a scalar field in addition to the flow field by changing how the texture is mapped to the streamline. The ability to choose texture parameters freely independent of polygonal geometry provides a great amount of flexibility in depicting scalar quantities.

Starting at the beginning of the streamline, the geometric coordi-

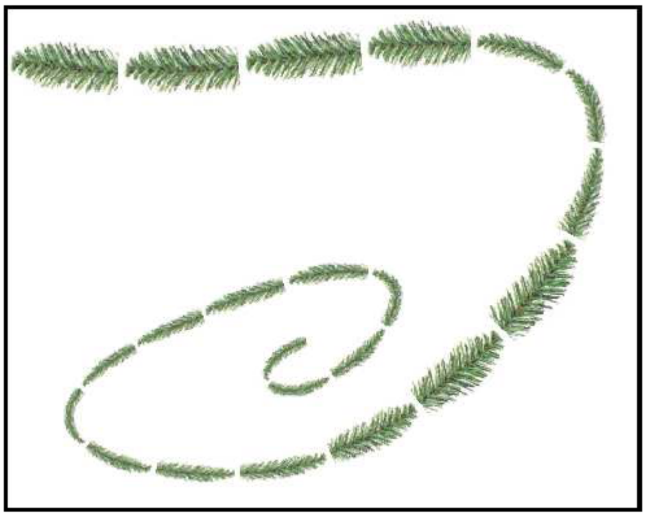


Figure 10: Texture parameters can be adjusted to display a scalar distribution in addition to the vector field. The thickness of the pine texture directly corresponds to the velocity magnitude along the streamline.

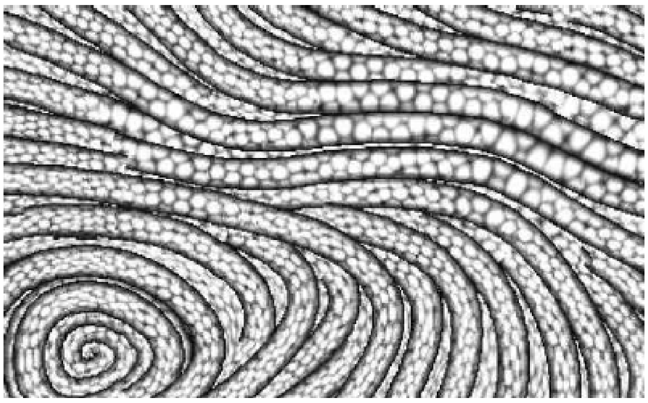


Figure 11: Illustration of using texture attributes to represent a scalar distribution. The scale of the texture is directly related to Reynolds shear stress – a scalar field used to characterize regions where drag is generated in turbulent boundary layers.

nates for the thick streamline are calculated and vertices for the first polygon are defined. The length of the texture, u, or the width of the texture, v, can be scaled according to the a scalar value and applied to the polygon. Texture continuity between polygons is preserved by ensuring that each polygon starts with the texture coordinates most recently used by an adjacent polygon. Proportionally varying both the u and v components of the texture influences the relative scale of the texture (figure 10). This technique creates a difference in the spatial frequency of the texture, reflecting the magnitude of a scalar distribution. Figure 11 shows how the scale of a texture can be used to display the magnitude of Reynolds shear stress – a scalar field used to characterize regions where drag is generated in turbulent boundary layers.

4.5 Multiple Textures

Texture mapping streamlines gives great flexibility in the number of different appearances that a vector field representation can have. Figure 12 shows a sample of how textures are capable of many different appearances as applied to streamlines.

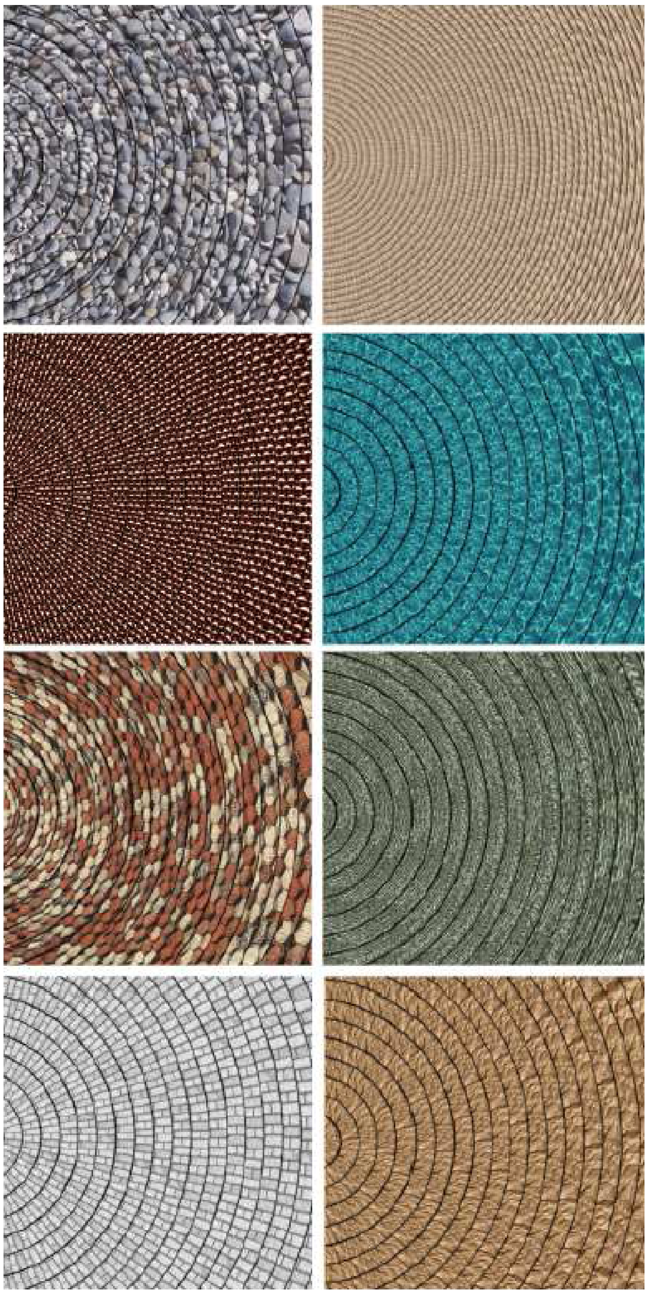


Figure 12: Examples of the diversity of natural textures that can be applied to a vector field. A circular flow is used demonstrate each example.

5 Comparison of Techniques

We have presented two complimentary techniques for using textures for the visualization of 2D vector fields and the generation of user-controlled textures: texture synthesis (section 3) and texture mapping (section 4). Both are effective methods for creating a texture that accurately define the complex patterns that occur in vector field visualizations.

The texture synthesis algorithm presented allows for a controlled, pixel-by-pixel approach that mimics the sample texture effectively in areas of high curvature as well as throughout the entire domain. By using an elliptical kernel oriented in the principal direction of flow, the texture synthesis approach effectively captures the flow orientation, minimizing artifacts, while maintaining the statistical properties and appearance of the input texture. A potential limitation of the synthesis technique is that the quality of the output texture is largely dependent on the size of the search neighborhood. While the elliptical kernel allows for a high-quality image to be produced and uses a smaller neighborhood than traditional techniques, the algorithm is not yet interactive.

The texture mapping approach allows for an efficient, interactive visualization of a vector field. Through the use of modern graphics hardware, textures can be applied to a vector field interactively – giving the user the ability to create many different appearances for the same flow field. Also, as described in section 4.3, the use of an outlining texture allows textures samples that do not have an inherent principal direction to be used in the visualization. The outline of the thick streamline depicts the flow orientation while various components of isotropic textures can be used to depict scalar variables of the flow (e.g. Reynold's stress, vorticity magnitude, or velocity magnitude).

6 Conclusions and future work

The use of textures allows for a consistent and highly-detailed representation of a vector field, allowing an observer to both analyze and better understand the dynamics of fluid flow. We introduce textures to convey this information in a way that preserves the integrity of the vector field while also taking advantage of the many perceptual dimensions that textures can contribute such as regularity, directionality, contrast, and spatial frequency.

We have presented two techniques that allow for textures to accurately reflect the form and behavior of a flow field. We have presented the use of an anisotropic filter for texture synthesis that, unlike conventional algorithms, results in a higher quality texture as it is synthesized over a vector field. This technique allows the integrity of the texture to be maintained while it accurately reflects the underlying curvature of the flow. We have also analyzed the use of texture mapping to accurately depict flow fields. We discussed how texture mapping allows anisotropic textures can be utilized for flow visualization and how variables within the flow can be represented.

Our current and future work deals with the use of such techniques, streamline-based as well as pixel-based, to better represent multi-parameter data sets taking advantage of the perceptual dimensions of textures.

Additional areas of future research include extending the proposed neighborhood specifications for texture synthesis to the three-dimensional domain, where the weighting functions can be usefully adapted adding a third extra dimension. Additional fields of applications can be found in the visualization of tensors, as well as in generating solid textures. Furthermore, we are interested in visualizing multi-fields within the same image.

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. *Proceedings of 2001 Symposium on Interactive 3D Graphics*, 217–226.

CABRAL, B., AND LEEDOM, C. 1993. Imaging vector fields using line integral convolution. *Proc. of SIGGRAPH 93*, 263–269.

CIE-Lab System 1976. Commission International de l' Éclairage, CIE.

DE LEEUW, W. C., AND VAN WIJK, J. J. 1995. Enhanced spot noise for vector field visualization. *Proceedings of IEEE Visualization 95*, 233–239.

DURAND, F., AND DORSEY, J. 2002. Fast bilateral filtering for the display of high-dynamic-range images. *Proceedings of Siggraph 21*, 3, 257–266.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001*, 341–346.

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. *International Conference on Computer Vision*, 1033–1038.

GORLA, G., INTERRANTE, V., AND SAPIRO, G. 2002. Texture synthesis for 3d shape representation. *IEEE Transactions on Visualization and Computer Graphics*, 512–524.

JOBARD, B., AND LEFER, W. 1997. Creating evenly-spaced streamlines of arbitrary density. *Proc. of the 8th Eurographics Workshop on Visualization in Scientific Computing*, 43–55.

SHEN, H.-W., LI, G., AND BORDOLOI, U. 2004. Interative visualization of three-dimensional vector fields with flexible appearance control. *IEEE Transactions on Visualization and Computer Graphics 10*, 4, 434–445.

STALLING, D., AND HEGE, H.-C. 1995. Fast and resolution-independent line integral convolution. *Proceedings of SIGGRAPH 95*, 249–256.

TAPONECCO, F., AND ALEXA, M. 2003. Vector field visualization using markov random field texture synthesis. *Proceedings of Eurographics / IEEE TCVG*, 195–202.

TAPONECCO, F., AND ALEXA, M. 2004. Steerable texture synthesis. *Proceedings of Eurographics 2004*, 57–60.

TOMASI, C., AND MANDUCHI, R. 1998. Bilateral filtering for gray and color images. *Proceedings of the 1998 IEEE International Conference on Computer Vision*, 839–846.

TURK, G., AND BANKS, D. 1996. Image-guided streamline placement. *Proceedings of SIGGRAPH 96*, 453–460.

TURK, G. 2001. Texture synthesis on surfaces. *Proceedings of ACM Siggraph*, 347–354.

VAN WIJK, J. J. 1991. Spot noise: Texture synthesis for data visualization. *Proceedings of SIGGRAPH 91*, 309–318.

VAN WIJK, J. J. 2002. Image based flow visualization. *ACM Transactions on Graphics 21*, 3, 745–754.

VERMA, V., KAO, D., AND PANG, A. 1999. Plic: Bridging the gap between streamlines and lic. *Proc. Symposium on Data Visualization 1999*, 341–348.

WANG, B., WANG, W., YONG, J., AND SUN, J. 2005. Synthesizing 2d directional moving texture. In *SCCG '05: Proc. of the 21st spring conference on Computer graphics*, 177–183.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. *Proceedings of SIGGRAPH 2000*, 479–488.