Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2003)
G.-P. Bonneau, S. Hahmann, C. D. Hansen (Editors)

Vector Field Visualization using Markov Random Field Texture Synthesis

Francesca Taponecco

Marc Alexa

Department of Computer Science, Interactive-Graphics System Group, Darmstadt University of Technology
Fraunhoferstr. 5, 64283 Darmstadt,Germany
{ftapone,alexa}gris.informatik.tu-darmstadt.de

Abstract

Vector field visualization aims at generating images in order to convey the information existing in the data. We use Markov Random Field (MRF) texture synthesis methods to generate the visualization from a set of sample textures. MRF texture synthesis methods allow generating images that are locally similar to a given example image. We extend this idea for vector field visualization by identifying each vector value with a representative example image, e.g. a strongly directed texture that is rotated according to a 2D vector. The visualization is synthesized pixel by pixel, where each pixel is chosen from the sample texture according to the vector values of the local pixel. The visualization locally communicates the vector information as each pixel is chosen from a sample that is representative of the vector. Furthermore it is smooth, as MRF texture synthesis searches for best fitting neighborhoods. This leads to dense and smooth visualizations with the additional freedom to use arbitrary representation textures for any vector value.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation

1. Introduction

Vector fields arise from experiments, measurements and simulations in many scientific and engineering disciplines. The visualization of this data is important to understand the underlying nature of the processes exhibiting a particular field or to be able to predict the behavior of systems in the real world.

Most vector field visualization techniques generate raster images and require vector information for each pixel. However, not every pixel contains information about the field at its location; rather several pixels are used to generate anisotropic textures that, together, represent the direction and magnitude of the field.

The main problem of vector field visualization is to generate expressive textures (where conveying the vector information in a certain point requires large groups of pixels) while not missing important detail. If, for instance, glyphs are used for visualization (e.g. lines or arrows), the question is where to place them (see [22] for a possible answer). If they are too sparse, detail is missing, if they are too dense, they overlap and information is lost.

A common way of vector field visualization is to introduce regularity into an otherwise irregular pattern. The most prominent methods are spot noise [23] and line integral convolution (LIC) [5]. However, it seems that some degrees of freedom of the raster image are wasted for the presentation of noise, which does not directly contribute to the communication of information.

In this work, we present a vector field visualization technique that is capable of generating continuous visualizations of a vector field. The user may define textures for any vector value in the field. This information is used to generate a smooth image that respects the mapping from vectors to textures.

Our approach exploits recent Markov model texture synthesis methods[9, 25, 1, 15]. The basic idea is to generate an im-

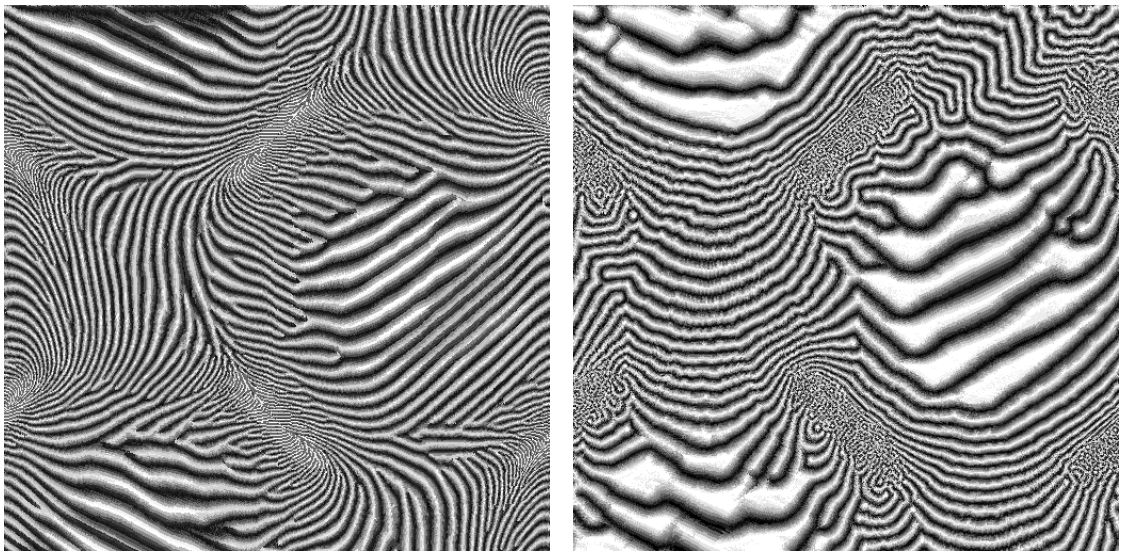

Figure 1: *Vector field visualizations synthesized using MRF texture synthesis with a gradient example texture that is rotated and scaled according to the vector field. The two images use different sample textures, which are characterized by lines of different orientations.*

age by selecting pixels from an example based on similar neighborhoods. More specifically, each pixel in the output image is statistically chosen by comparing the (already synthesized) neighborhood in the output with all possible locations in the example image. Thus, structures in the example image are re-synthesized in the output image.

The main idea of this work is to use a space of example images rather than only one. In each pixel, the example image is determined by the vector in the vector field at the respective location. To visualize a direction field, for example, one might pick any anisotropic texture and rotate it in each pixel so that its major axes are aligned with the vector field. Scalar values in the field could be visualized by scaling the example images. An example is given in Figure . In general, any procedural or manual way to define a mapping from vector space to example image space is possible.

2. Related work

2.1. Vector field visualization

Vector field visualization is a large and diverse field. We will only briefly discuss work that is directly related to our approach. For a good overview of flow visualization see the STAR of Post et al.[18].

Direct visualization techniques use color coding or icons to represent samples of the vector field. The use of little arrows as icons has been termed hedgehod displays[16].

Geometric visualization is similar to direct visualization, however, uses geometric objects extracted from the field rather than fixed icons. Typical geometric objects are isovector objects (contours) or integral objects (streamlines).

Both approaches require the vector field to be sampled and the icons or objects are placed according to the sampling pattern. Sampling on a regular grid might lead to aliasing artifacts and one might want to optimize the pattern so that objects are evenly distributed over the resulting image[22].

Texture-based visualization could be seen as a geometric visualization approach that uses dense, regular sampling. In most techniques isotropic noise is *smeared* in the direction of the vector field. Spot noise[23] distributes a set of intensity functions (spots) over the domain, which are moved by small steps over time. Intensity functions can be chosen in a way that also magnitude information is displayed[7]. Line integral convolution (LIC)[5] starts from a white noise texture and integrates the gray values along lines. This results in a tangental smoothing of the noise texture so that the texture is smooth along the tangents and noisy along the gradient. LIC has been extended in several directions, mainly to make the computation faster and to incorporate additional information in the visualization. An idea related to LIC is to smooth the white noise image with a varying anisotropic filter, whose major axes are aligned with the vector field[8]. The most recent texture-based flow visualization techniques exploit modern graphics hardware to compute LIC-like textures in real-time[24].

Other approaches are based on algorithmic painting via vector-like brush strokes (see Haeberli[11] and Crawfis[6] for a 3d extension), on reaction diffusion techniques (see Turk[20],

Witkin[27]) and hyper-textures for a 3-dimensional visualization (for more details see Perlin[17]).

2.2. Texture synthesis

Our approach is based on recent texture synthesis methods. The goal of texture synthesis is as follows: Given a sample of a texture, synthesize a new texture of arbitrary resolution that appears to be generated by the same underlying process to a human observer. Approaches mostly differ in the model used to describe the stochastic process that generates the textures.

Recent approaches model textures as Markov Random Fields (MRF)[9, 25, 1, 15] and generate the output texture in a pixel by pixel fashion. The idea of these works is roughly the same. The new texture is generated in scan-line order (or, more generally, on a space filling curve). Each pixel is synthesized by comparing its neighborhood to all similarly shaped neighborhoods in the sample texture. These comparisons lead to a distance, which is used to compute the probability to choose the best matching pixel. Very similar neighborhoods result in highest probabilities. Random number generation together with the probability distribution lead to the selection of the neighborhood, which contains the pixel to be synthesized.

The most time consuming process during synthesis is the comparison of a given neighborhood with all similar blocks in the input sample. A look-up table can be used to speed up this process significantly[4]. Another way to synthesize large textures faster is to copy blocks rather than pixels in each step of the algorithm[10].

The size of the neighborhood depends on the size of the structure in the example texture. Large structures require large neighborhoods, which leads to slow processing. If the example texture exhibits structures on several scales even large neighborhoods might fail to capture large and small features of the texture. A better approach to capture features on several scales is to use image pyramids and a multiresoltuion synthesis process[12, 2, 25]: The output is first generated on lower resolution using a low-pass filtered version of the example texture. The resolution of the output is then refined using examples with more detail. The process is repeated until the finest level of the example is reached. With this approach, large-scale features are in the coarse image, thus, avoiding neighborhoods with a large number of pixels.

Some works consider the idea of using more than one example texture or to adapt the texture to local properties. In particular, works that synthesize texture directly on manifold surfaces embedded in 3-space[19, 21, 26] use a direction field over the surface and adapt an anisotropic example so that it conforms with the direction field. This is in part somewhat similar to our approach. However, here we focus on the visualization of the properties of a given vector field, while texturing a manifold surface allows to adapt the direction field to the purpose of texturing.

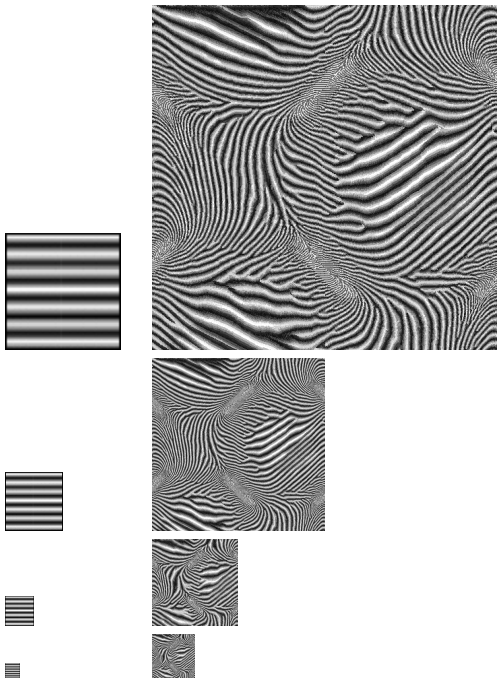

Figure 2: *Pyramid levels for the synthesis process of the image in Figure 1*

3. Approach

Our approach to vector field visualization is a generalization of Markov Random Field texture synthesis methods. Most texture synthesis methods use one texture example and aim at producing an image that is locally similar to the sample. Our idea is to assume that each vector value in the vector field has an ideal candidate texture, which reflects properties of this vector such as direction or magnitude. Consequently, the visualization is generated by synthesizing each pixel using an example texture according to the vector value in the location of the pixel.

More formally, let Φ be a vector field in d dimensions over $\mathbb{R}^2$:

$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^d \tag{1}$$

Each vector $\vec{v} = \Phi(x,y)$ defines an example texture image, i.e.

$$\tau(\vec{v}) : [0,1]^2 \rightarrow [0,1]^c \tag{2}$$

with $c = 1$ for gray level and $c = 3$ for colored textures.

To generate a visualization of a certain part of the vector field one defines the pixel counts of the output image, which in turn define the sampling of the vector field. The pixel at (x,y) is computed using an appropriate neighborhood (or neighborhood pyramid) of that pixel and the MRF texture synthesis method with example texture $\tau(\Phi(x,y))$.

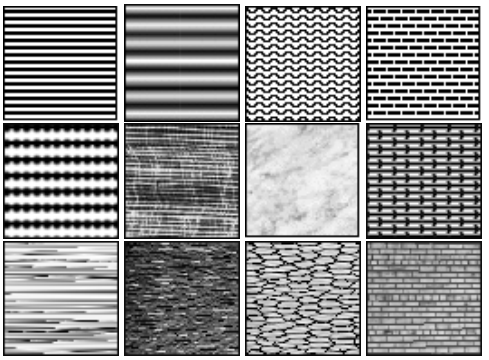

Figure 3: *Examples of sample patterns for imput images.*

Note that this approach combines the ideas of glyph based visualization with dense, texture-based approaches. Each vector value could have its own glyph/texture – the texture synthesis technique assures that these glyphs are combined in a seamless way.

This approach is quite general and allows arbitrary example images for different vector values. To achieve expressive results, however, the mapping from vector values to example textures has to be continuous and intuitive.

In most cases one wants to visualize the properties (i.e. direction and magnitude) of the vector field. The magnitude can be easily computed using an appropriate norm of the values, i.e.

$$A(x,y) = ||\Phi(x,y)|| \tag{3}$$

Assigning a direction requires a projection of the vector onto the image plane. Let $\Phi(x,y)_x$ and $\Phi(x,y)_y$ be those projection then

$$\theta = arctan\frac{\Phi(x,y)_y}{\Phi(x,y)_x} \tag{4}$$

is the angle of the tangent in the vector field relative to the x-axis.

A straightforward approach for the mapping from vector values to example images would be to use the information in A and θ to scale and rotate an example image. This is also the approach that we have used to generate all examples in this work. Typical example images should have a certain directional structure and scale features so that their scale and rotation is easy to perceive. The set of example images we have used is depicted in Figure 3. Figures 3.9 - 3.12 are derived from [3]. We have mostly used simple gray scale images that are constant along one direction and smoothly vary along the other. Figures and 7 show the visualization results obtained for the same vector field using different example textures (i.e. using the first two samples of Figure 3. See Figure 9 for further examples.

Critical points are prominent features in vector fields. We have generated visualizations of isolated critical points to evaluate how prominent those features become in the visualization. The result is depicted in Figure 5.

In addition, one could perform a local analysis and determine critical points (using the eigenvalues of the Jacobian[13, 14]). Special textures could then be devoted to the different classes of critical points.

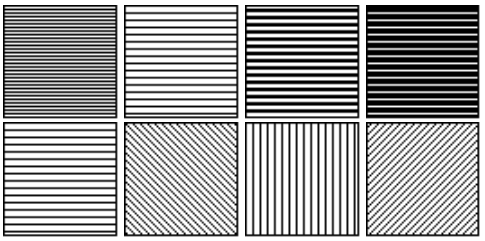

Figure 4: *Scaled and rotated sample input images.*

To sum up, the typical procedure of vector visualization using MRF texture synthesis uses the following steps:

- An example image defines the visualization primitive. The primitive should be anisotropic and scale-dependent.
- The dimensions of the output image are set; the dimensions also define the sampling of the vector field. It is assumed that these samples are accessible.
- For every pixel in the output image, the sample image is modified according to the vector values, that is, the input texture is rotated and scaled by these parameters (see Figure 4).
- For every pixel an L Neighborhood is defined, which is comprised of neighboring pixels (the size is user defined).
- Every pixel in the output image is generated in scan order with a routine, which searches the most similar pixel in the modified sample image, according to a probability model.
- These probabilities are defined from comparing the distancies between the neighboring pixels inside the neighborhoods.

In particular, the synthesis algorithm is described by the following pseudo-code, where is for reasons of brevity restricted to the case of single resolution:

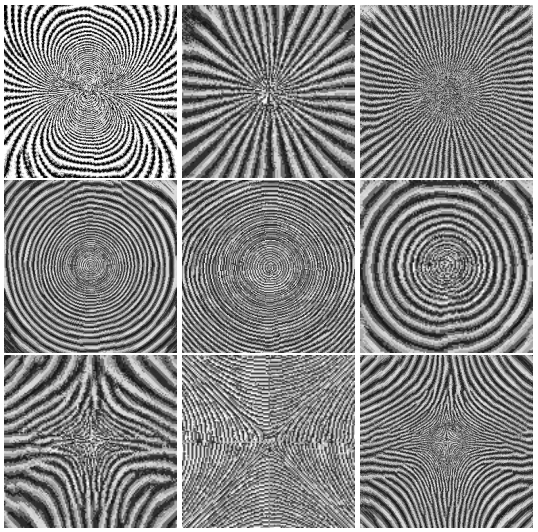

Figure 5: *Visualization of critical points.*

function *synthesizePixel*

```
1  for(x = 0; x < outWidth; x++) {
2    for(y = 0; y < outHeight; y++) {
3      outN = CalculateNeighborhood(x, y);
4      A = CalculateAmplification(x, y);
5      θ = CalculateAngleRotation(x, y);
6      AmplifyInput(A);
7      RotateInput(θ);
8      for(i = 0; i < inWidth; i++) {
9        for(j = 0; j < inHeight; j++) {
10         inArrN = CalculateNeighborhood(i, j);
11         distance = CompareNeighborhoods(outN, inArrN);
12         }
13       }
14     bestMatch = GetBestPixel(distance);
15     SynthetizeOutputPixel(bestMatch);
16   }
17 }
```

Table 1 and Table 2 explain the variables and the functions used in the pseudo-code.

Variable	Description
outWidth	Horizontal size of sample output image
outHeight	Vertical size of sample output image
outN	"L" Neighborhood
A	Vector field amplitude at current output position
θ	Vector field phase at current output position
inWidth	Horizontal size of sample input image
inHeight	Vertical size of sample input image
inArrN	Array of neighborhoods for input image pixels
distance	Difference (in terms of pixel values) between the "L" neighborhoods
bestPixel	Best match based on distance comparison

Table 1: *Table of variables*

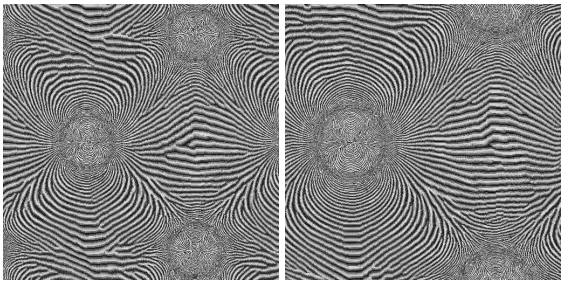

Figure 6: *Different parts of the vector field visualized using the same output resolution.*

4. Results & Discussion

We have implemented the ideas using a pyramid-based version of MRF texture synthesis. Figures 6 and 8 show some of the results obtained using a simple sample texture.

We feel that the approach has several notable features. These include:

Accuracy The synthesis method works pixel by pixel – this guaranties a smooth and continuous output.

Generality The approach is fully general: every vector field can be visualized, given an arbitrary mapping from vector values to texture samples. This allows the generation of pop-out visual features for application dependent critical values.

Ease of use It is simple to define a meaningful mapping from vectors to examples by using phase and amplitude of the field to rotate and scale a single sample texture.

The size of a sample image plays a critical role in rotating/scaling of the image and has to be chosen appropriately. Small examples allow details to be visualized, however, it is hard to achieve continuity for changing vector values (see Figure 7). Large structures allow to use larger neighborhoods for comparison and are likely to yield smoother results. Yet, this comes at the price of locality.

The time needed for generating the visualization is identical to the texture synthesis method used. Rotated and scaled versions of the examples are precomputed and fetched from look-up tables. Most MRF texture synthesis methods require several minutes to several hours to generate the results depicted in this paper. We have not yet optimized our code to include very recent variants that promise significantly reduced computation times[4].

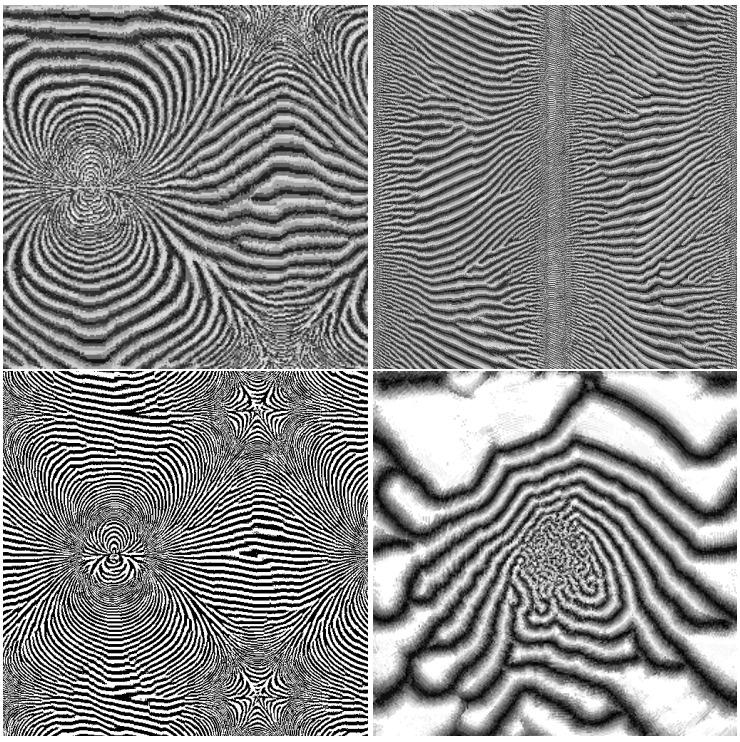

Figure 8: *Examples of synthetized vector fields.*

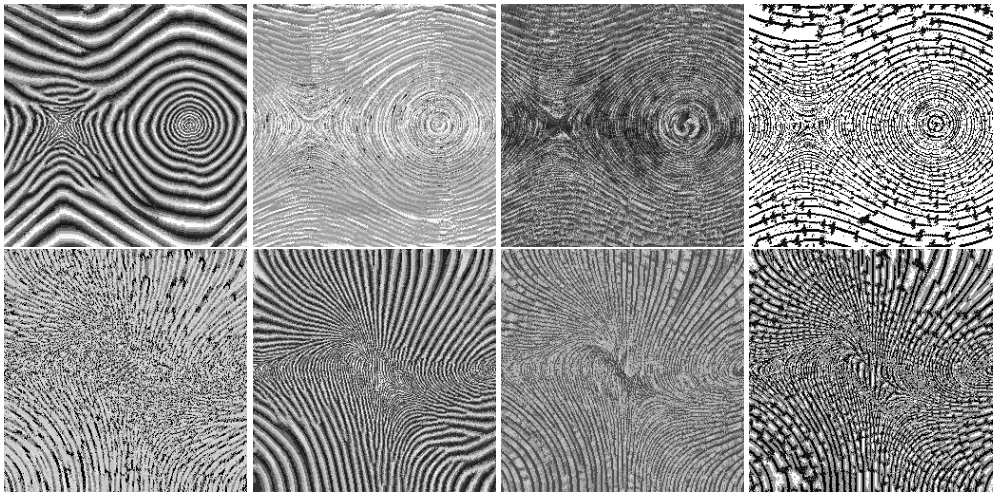

Figure 9: *Synthetized outputs: two vector fields are obtained both usind two sets of different input sample images chosen from Figure 3*

5. Conclusions

We have presented an approach to vector field visualization that combines the flexibility of direct, icon-based methods with the effective use of display area typical for texture based methods. In a sense, the method generalizes texture-based methods to use arbitrary texture samples rather than only noise.

Although it is fairly straightforward, we have not yet exploited the idea of using special example textures for critical points or special, application specific values in the vector

Function	Description
CalculateNeighborhood	calculates the neighborhood of current output pixel
CalculateAmplification	calculates the vector field amplitude in the output image at (x,y) location
CalculateAngleRotation	calculates the vector field phase in the Output Image at (x,y) location
AmplifyInput	amplifies the input sample by its argument
RotateInput	rotates the input sample by its argument
CompareNeighborhoods	compares the current output pixel neighborhood with those of the input image pixels
GetBestPixel	chooses the pixel (in input image), whose neighborhood best matches that of the current output pixel (in terms of minimum distance)
SynthetizeOutputPixel	sets the current output pixel to the value of bestMatch

Table 2: *Table of functions*

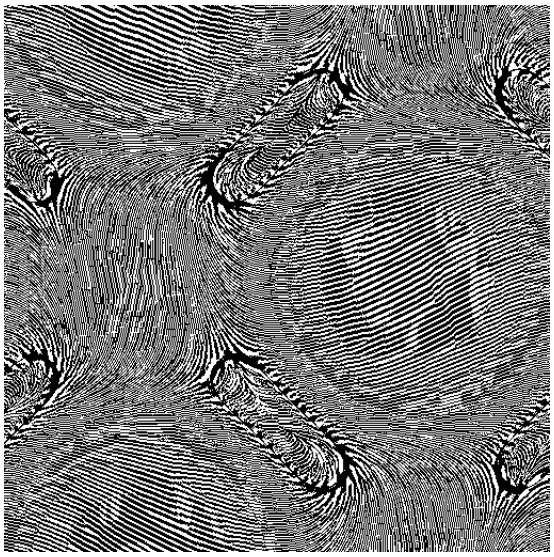

Figure 7: *Using a small scale example texture might lead to aliasing artifacts in the synthesized visualization.*

field. Incorporating this feature should lead to stronger visual results.

This approach is also promising for the visualization of higher dimensional data or tensor fields if some reasonable mapping from values to example textures could be defined. In general, we feel that more investigation for suitable mappings from data values to example textures is needed.

Finally, our current implementation would benefit from using the latest possibilities in speeding up the texture synthesis computation.

Acknowledgments

We thank Wolfgang Müller for discussions in the early phase of this project.

References

1. Michael Ashikhmin. Synthesizing natural textures. In *2001 ACM Symposium on Interactive 3D Graphics*, pages 217–226, March 2001. ISBN 1-58113-292-1.
2. Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. *Proceedings of SIGGRAPH 97*, pages 361–368, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.
3. P. Brodatz. *Textures: A photographic album for artists and designers*. Dover Publications, New York, 1966.
4. Stephen Brooks and Neil Dodgson. Self-similarity based texture editing. *ACM Transactions on Graphics*, 21(3):653–656, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).
5. Brian Cabral and Leith (Casey) Leedom. Imaging vector fields using line integral convolution. *Proceedings of SIGGRAPH 93*, pages 263–272, August 1993. ISBN 0-201-58889-7. Held in Anaheim, California.
6. Roger Crawfis and Nelson Max. Direct volume visualization of three-dimensional vector fields. *1992 Workshop on Volume Visualization*, pages 55–60, 1992.
7. C. W. de Leeuw and J. J. van Wijk. Enhanced spot noise for vector field visualization. In *IEEE Visualization '95 Proceedings*, pages 233–239. IEEE Computer Society, October 1995.
8. U. Diewald, T. Preußer, and M. Rumpf. Anisotropic diffusion in vector field visualization on euclidean domains and surfaces. *IEEE Trans. Vis. and Comp. Graphics*, 6(2):139–149, 2000.
9. A. Efros and T. Leung. Texture synthesis by non-parametric sampling. In *International Conference on Computer Vision*, pages 1033–1038, 1999.
10. Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 341–346.

ACM Press / ACM SIGGRAPH, August 2001. ISBN 1-58113-292-1.

11. Paul E. Haeberli. Paint by numbers: Abstract image representations. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):207–214, August 1990. ISBN 0-201-50933-4. Held in Dallas, Texas.
12. David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. *Proceedings of SIGGRAPH 95*, pages 229–238, August 1995. ISBN 0-201-84776-0. Held in Los Angeles, California.
13. James L. Helman and Lambertus Hesselink. Representation and display of vector field topology in fluid flow data sets. *IEEE Computer*, 22(8):27–36, August 1989.
14. James L. Helman and Lambertus Hesselink. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics & Applications*, 11(3):36–46, May 1991.
15. Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 327–340. ACM Press / ACM SIGGRAPH, August 2001. ISBN 1-58113-292-1.
16. R. Victor Klassen and Steven J. Harrington. Shadowed hedgehogs: A technique for visualizing 2d slices of 3d vector fields. *Visualization '91*, pages 148–153, 1991.
17. Ken Perlin and Eric M. Hoffert. Hypertexture. *Computer Graphics (Proceedings of SIGGRAPH 89)*, 23(3):253–262, July 1989. Held in Boston, Massachusetts.
18. Frits H. Post, Robert S. Laramee B. Vrolijk, H. Hauser, and H. Doleisch. Feature extraction and visualisation of flow fields. *IEEE Computer*, pages 69–100, September 2002. In Dieter Fellner and Roberto Scopigno, editors, Eurographics 2002 State of the Art Reports. The Eurographics Association, Saarbrücken, Germany.
19. Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. *Proceedings of SIGGRAPH 2000*, pages 465–470, July 2000. ISBN 1-58113-208-5.
20. Greg Turk. Generating textures for arbitrary surfaces using reaction-diffusion. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):289–298, July 1991. ISBN 0-201-56291-X. Held in Las Vegas, Nevada.
21. Greg Turk. Texture synthesis on surfaces. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 347–354. ACM Press / ACM SIGGRAPH, August 2001. ISBN 1-58113-292-1.
22. Greg Turk and David Banks. Image-guided streamline placement. *Proceedings of SIGGRAPH 96*, pages 453–460, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.
23. Jarke J. van Wijk. Spot noise-texture synthesis for data visualization. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):309–318, July 1991. ISBN 0-201-56291-X. Held in Las Vegas, Nevada.
24. Jarke J. van Wijk. Image based flow visualization. *ACM Transactions on Graphics*, 21(3):745–754, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).
25. Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. *Proceedings of SIGGRAPH 2000*, pages 479–488, July 2000. ISBN 1-58113-208-5.
26. Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. In ACM, editor, *SIGGRAPH 2001 Conference Proceedings, August 12–17, 2001, Los Angeles, CA*, pages 355–360, New York, NY 10036, USA, 2001. ACM Press.
27. Andrew Witkin and Michael Kass. Reaction-diffusion textures. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):299–308, July 1991. ISBN 0-201-56291-X. Held in Las Vegas, Nevada.