

# Steerable Texture Synthesis for Vector Field Visualization

Vom Fachbereich Informatik  
der Technischen Universität Darmstadt  
genehmigte

Dissertation

zur Erlangung des akademischen Grades eines  
Doktor-Ingenieurs (Dr.-Ing.)

von

Dipl.-Ing. Francesca Taponecco

aus Sarzana

Referenten der Arbeit: *Prof. Dr.-Ing. Marc Alexa*, Technische Universität Berlin  
*Hon. Prof. Hans-Christian Hege*, ZUSE Institute Berlin  
*Prof. Dr. Bernt Schiele*, Technische Universität Darmstadt

Tag der Einreichung: 29. August 2006  
Tag der mündlichen Prüfung: 11. Oktober 2006

D17  
Darmstädter Dissertation 2006



# Acknowledgments

This dissertation could become a reality thanks to the help of numerous people who contributed in different ways supporting me in these years of research.

Foremost I want to express my special thanks to my advisor Marc Alexa. I would like to thank him for his guidance during my research at the Darmstadt University of Technology; he provided useful suggestions and support, giving me the freedom and the encouragement to pursue this research. Thanks to his experience and competence in every field of quest and thanks to his friendly way of leading the group, he represented an important reference point for me. He encouraged me in investigating interesting topics in computer graphics and related fields, supporting my cooperation with other researchers and giving me also the chance to spend a period of exchange research at the University of Minnesota, where I had the opportunity to work with experts involved in my research field. I am really grateful for the opportunity to work in his team and it has been a valuable experience for me.

I would like to sincerely thank Professor Hans-Christian Hege and Professor Bernd Schiele for kindly having accepted being my co-advisors, for showing interest and for taking the time and the trouble, and occupying themselves with my work.

I would like to express my deepest gratitude to Professor José Luis Encarnação for giving me the chance to work in his institute. Being in contact with many colleagues from the University and the research Institute for Computer Graphics and Data Processing has been an excellent opportunity, enabling me to learn and grow in an international, stimulating research environment.

Many thanks to Professor Vicki Interrante and her group at the University of Minnesota, for having welcome me so friendly and for the stimulating discussions during my stay in Minneapolis, it has been and it is a pleasure working with them. My appreciations also go to my colleagues at the Graphical Interactive Systems Group GRIS, Anders Adamson, Sabine Bartsch, Uwe Berner, Carola Eichel, Roya Foroughi, Rolf Lindner, Dietmar Hildenbrand, Andy Nealen, Thomas Rieger and Silke Romero, for the friendly and creative atmosphere in the graphics group. I would like to thank all my fellow scientists who helped me in my work through informal discussion, showing interest in my work and encouraging me to keep up with this research, I am particularly thankful to the researchers who allowed me to use some of their pictures, which I reprinted for the state of the art survey of this thesis.

I thank all my friends, in Italy, Germany and other countries, who also helped me with their support and friendship, and a special thank goes to Thomas for his support, understanding and patience. Finally, and most importantly, I would like to thank my family in Italy for their constant support throughout every phase of my life, being an essential point of reference and source of support for me also in this time away from home.



# Zusammenfassung

## Übersicht

In dieser Arbeit werden neue Methoden zur Visualisierung von Vektorfeldern und zur Generierung einer kontrollierbaren Textursynthese vorgestellt. Ein intuitiver Ansatz wird eingeführt, welcher flexibel und einfach erweiterbar ist.

Traditionelle Theorien der Bildverarbeitung für Textursynthese wurden für dieses Ziel verwendet und angepasst. Texturen sind musterbasierte Bilder, welche statistische und/oder strukturelle Eigenschaften als räumliche Verteilung von Pixel zeigen. Texturen sind charakterisiert von einem Satz von perceptiven, visuellen Dimensionen und sind deshalb besonders geeignet als Paradigma für eine flexible Informationsabbildung in der Visualisierung. Die Steuerung von anisotropen Texturen entlang einem Vektorfeld ermöglicht eine flexible und effektive Visualisierung von vektoriellen Distributionen. Somit können Teile von Texturen oder texturellen Elementen, auch Texel genannt, als Primitive benutzt werden, für den Zweck der Vektorfeldvisualisierung und feldergesteuerten Textursynthese. Der Hauptinhalt von vektoriellen Datensätzen, und bedeutsame Attribute können daher mittels intuitiver Transformationen über texture seeds kodiert werden. Die Kontrolle über den Syntheseprozess ermöglicht und generiert eine Varietät von Effekten in dem resultierenden Bildeergebnis.

## Grundidee

Die Grundidee ist, dass Vektorfelder mit dem folgenden Ansatz visualisiert werden können: die Farbe oder die Intensität der Outputpixel kann abhängig von kalkulierten Ähnlichkeitsfunktionen einfach gesetzt werden.

Diese Methode ist texturbasiert aber, anders als das blurring und smearing einer starting noise texture entlang der Richtung des Vektorfeldes, sie passt ein ausgewähltes Startmuster dem Vektorfeld an, und sie transformiert das Muster von einer anisotropen Beispieltexur um, zur Darstellung und Visualisierung der Eigenschaften und Variation der Vektorfelder.

Der traditionelle Textursyntheseprozess funktioniert wie folgt: ein kleines, stationäres und lokales Textursample ist gegeben und aus dem wird eine größere Textur (mit beliebiger Auflösung) synthetisiert, die zu dem Sample visuell ähnlich ist. Das heißt, der Output soll so aussehen ob er von dem gleichen unterliegenden Prozess generiert worden wäre; zwei Texturen werden gleich erscheinen, wenn einige bestimmte Statistiken von diesen Bildern korrespondieren. Der Syntheseprozess wird angepasst, um vektorielle Informationen einzuschließen.

## Ansatz: Vektorfeldvisualisierung und kontrollierbare Textursynthese

Die hier präsentierte Methode erzeugt eine Textur, welche durch pixel per pixel Synthetisierung generiert wird. In diesem Standardverfahren wird in einem Outputbild jedes Pixel in scan-line Reihenfolge gesetzt, in dem man die Pixelnachbarschaft mit den formgleichen Nachbarschaften aus dem Inputsample vergleicht. Eine Distanzfunktion wird berechnet, auf Basis von Farbewertähnlichkeit und Nähe zu dem aktuellen Pixel. Die Nachbarschaften des Pixels werden verglichen und der beste Pixelwert wird statistisch ermittelt. Das Pixel mit der ähnlichsten Nachbarschaft ist dann das wahrscheinlichste, welches an die aktuelle Position im Outputbild zu setzen ist. In dem beschriebenen Verfahren wurde der Syntheseprozess mit Multiresolutionsanalyse implementiert, welche auf Gaussian Pyramide basiert.

In diesem Ansatz wird nicht mehr in einem einzigen Inputsample nach den bestpassenden Pixel gesucht, sondern es wird in einem Set von Samples gesucht. Bei diesem Verfahren wird daher als Input ein Satz von Samples spezifiziert und benutzt. Dank der Benutzung von Filter und Transformationen, kann das Verfahren zur Benutzung und Erzeugung von beliebigen Texturmustern verallgemeinert werden. Der erweiterte Syntheseprozess ist generell und ermöglicht auch die Synthese von nicht-homogen Texturen.

Dieser Satz besteht aus verschiedenen Versionen eines ursprünglichen Musters, welches in Richtung, Amplitude und noch weiteren verschiedenen Parametern (so wie graduelle Farbänderung, Bildbearbeitungsfilter) verarbeitet wurde. Diese Parameter entsprechen den Pixelinformationswerten, welche zu einer modifizierten, bzw. filtrierte Version zu dem ursprünglichen Sample korrespondieren. Jeder Punkt des erzeugten Bildes ist abhängig von seiner Position im Vektorfeld. Dieser liefert die Werte für Phase und Amplitude, welche die Rotation und Skalierung des ursprünglichen Musters bestimmen. Im Outputbild wird jedes einzelne Pixel aus einem Satz von Feldparametern definiert: es liefert die Werte, die ein spezielles Inputsample bestimmen. Im Allgemeinen erlaubt die Methode jede prozedurale oder manuelle Art zur Definition einer Korrelation und Abbildung zwischen Vektorraum bzw. Parameterraum und Beispielbilderraum. Filtrierungen und Transformationen tragen dazu bei, dass Beispielbilder möglichst frei transformiert werden können. Folglich bietet der vorgestellte Ansatz beliebige Freiheitsgrade zur Darstellung der Repräsentation des resultierenden Feldes. Die vorgestellte Methode kombiniert die Eigenschaften der Intuitivität von iconic mapping (direkte Visualisierung) mit den Eigenschaften der Lokalität und einer mächtigen Kodierung, welche typisch sind für texturbasierte Techniken. Die Methode erlaubt ebenfalls die Extraktion von interessanten, physisch wichtigen Attributen, so wie es bei geometrischen und featurebasierten Visualisierungsmethoden gemacht wird. Dies trägt zur besseren Darstellung der Feldhaupteigenschaften bei.

Die Generierung von kontrollierten oder deformierten und gekrümmten Texturen kann auch mit dem vorgestellten Ansatz einfach erreicht werden. In diesem Fall, mit einem besonderen Augenmerk auf Bewahrung der Strukturkomplexität der Beispieltexur. In beiden Anwendungsfällen, für Vektorfeldvisualisierung und für constrained texture synthesis, werden speziell anisotrope Muster verwendet. Direktionale Muster zeigen deutliche Eigenschaften auf einer Achse auf. Aus diesem Grund sind sie hervorragend geeignet, da sie am besten Krümmungen und Bewegungen entlang von kontrollierten Richtungen auswerten. Somit betonen die anisotropen Muster die Information von dem Kontrollfeld. Aus diesem Grund wird ein neuer Ansatz zur Spezifizierung der Nachbarschaften und der Pixelgewichtung vorgestellt, zusammen mit einer resultierenden Reihe von neuen Nachbarschaftsmodellen, die geeignet für die Bewahrung und Betonung der Richtung sind.

## **Ausblick**

Der Hauptbeitrag von dieser Arbeit ist eine Generalisierung des Textursyntheseprozesses und die Einführung neuer Techniken zur Vektorfeldvisualisierung und der kontrollierbaren Textursynthese. In dem generierten Outputbild kann jeder Punkt, der von Interesse ist, auf spezielle Weise hervorgehoben werden. So können kritische Punkte oder Bereiche von besonderem Interesse verschieden synthetisiert werden. Man kann auch aus einer einfachen Textur eine komplizierte, neue Textur generieren, mit Hilfe verschiedener Bildbearbeitungsfiltern, die lokal anwendbar sind. Das Verfahren ist generell, vielseitig, flexibel und einfach zu verwenden. Transformationseffekte und künstlerische Variationen sind möglich; vielversprechende Erweiterungen sollen weiter erforscht und getestet werden.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Contributions . . . . .	2
1.3	Outline . . . . .	4
<b>2</b>	<b>Context of related work</b>	<b>5</b>
2.1	State of the art in Perception and Cognitive Science . . . . .	5
2.1.1	Introduction . . . . .	5
2.1.2	Gestalt principles of perception . . . . .	6
2.1.3	Human Vision Theory . . . . .	7
2.1.4	Preattentive, attentive and postattentive processing . . . . .	8
2.1.5	Comments: integrating perceptual issues in visualization . . . . .	11
2.2	State of the art in Texture Synthesis . . . . .	12
2.2.1	Introduction . . . . .	12
2.2.2	Problem statement . . . . .	12
2.2.3	Categorization of textures . . . . .	12
2.2.4	Classification of approaches . . . . .	14
2.2.5	Comments and open issues . . . . .	20
2.3	State of the art in Vector Field Visualization . . . . .	22
2.3.1	Introduction . . . . .	22
2.3.2	Definitions and notation . . . . .	22
2.3.3	Techniques classification . . . . .	22
2.3.4	Comments . . . . .	28
2.4	Summary . . . . .	29
<b>3</b>	<b>Steering Texture Synthesis for Vector Field Visualization</b>	<b>31</b>
3.1	Methodology: visualizing vector fields using statistical theory . . . . .	32
3.1.1	Texture synthesis fundamentals . . . . .	33
3.1.2	Markov Random Field theory . . . . .	34
3.2	Sample-based visualization . . . . .	35
3.2.1	Conveying vectorial information through anisotropic patterns . . . . .	36
3.2.2	Texture databases . . . . .	37
3.2.3	Input matrix seed . . . . .	39
3.3	Synthesis approach . . . . .	40
3.3.1	Control vector field . . . . .	41
3.3.2	Field-driven sample alignment and transformation . . . . .	42

3.3.3	Building causal neighborhood models . . . . .	45
3.3.4	Calculating distances and measuring similarity probabilities . . . . .	48
3.3.5	Finding the best matching pixel . . . . .	49
3.4	Algorithm description and implementation . . . . .	50
3.4.1	Steps of the procedure . . . . .	53
3.4.2	Pseudocode . . . . .	53
3.5	Results and discussion . . . . .	54
3.5.1	Comments: limitation and benefits . . . . .	54
3.5.2	Contribution . . . . .	57
<b>4</b>	<b>Directional enhancement in texture-based vector field visualization</b>	<b>61</b>
4.1	Motivation and contribution . . . . .	62
4.2	Synthesis neighborhoods . . . . .	62
4.2.1	Standard squared and L-shaped neighborhoods . . . . .	62
4.2.2	Different neighborhood shapes . . . . .	63
4.3	Non-uniform neighborhood filtering . . . . .	64
4.3.1	Anisotropic neighborhood model specifications . . . . .	64
4.3.2	Bilateral filtering . . . . .	64
4.3.3	Gaussian filtering and kernel coefficients . . . . .	64
4.4	Isotropic and anisotropic weighting schemes . . . . .	65
4.4.1	Circular neighborhood . . . . .	65
4.4.2	Elliptical neighborhood . . . . .	67
4.4.3	Further weighting schemes: blobs . . . . .	70
4.5	Extensions . . . . .	70
4.5.1	Spherical and ellipsoidal neighborhoods . . . . .	70
4.5.2	Sample textures bending . . . . .	70
4.6	Comments . . . . .	71
<b>5</b>	<b>Multi-valued visualization</b>	<b>73</b>
5.1	Multivariate fields . . . . .	74
5.1.1	Multi-parameter fields . . . . .	74
5.1.2	Height fields . . . . .	75
5.1.3	Temporal fields . . . . .	75
5.2	Higher order vector fields . . . . .	84
5.2.1	Tensors . . . . .	84
5.3	Multiple scalar and vector fields visualization . . . . .	86
5.3.1	Interweaving vector fields . . . . .	86
5.3.2	Integrating streamline-based and texture-based visualization . . . . .	86
5.3.3	Dual vector fields . . . . .	89
<b>6</b>	<b>Extracting and encoding data content</b>	<b>95</b>
6.1	Motivation and challenges . . . . .	95
6.2	Vector field parameters . . . . .	96
6.2.1	User selected vs. field intrinsic features of interest . . . . .	96
6.2.2	Deriving vector field attributes . . . . .	96
6.3	Topological analysis . . . . .	98



6.3.1	Basic notions	98
6.3.2	Piecewise linear interpolation	100
6.3.3	Eigen-analysis	101
6.4	Augmenting the MRF algorithm with data encoding	105
6.4.1	Texture visual dimensions and texture space	105
6.4.2	Visual representations and correspondences of features	109
6.5	Filters and convolution kernels	110
6.5.1	Filter banks	111
6.5.2	Convolution kernels	111
6.5.3	Filtering effects	113
6.6	Adaptive information layering	118
6.6.1	Showing composite information	119
6.6.2	Layering information	120
6.6.3	Comments	122
<b>7</b>	<b>Other applications</b>	<b>123</b>
7.1	Steerable texture synthesis	123
7.1.1	Introduction and motivation	123
7.1.2	Synthesizing non-homogeneous textures	124
7.1.3	Controlling the texture generation	125
7.1.4	Results and discussion	128
7.2	Time- and space-variant texture synthesis	129
7.2.1	Introduction and motivation	129
7.2.2	Motion in texture synthesis	130
7.2.3	Animating textured images	133
7.2.4	Results and discussion	137
7.3	Concluding remarks	139
7.3.1	Texture mixture and metamorphosis	140
7.3.2	Solid textures	140
7.3.3	Inpainting	140
<b>8</b>	<b>Conclusions</b>	<b>143</b>
8.1	Summary and contributions	143
8.2	Future work	146
<b>A</b>	<b>Multi-resolution synthesis</b>	<b>151</b>
A.1	Progressive refinement	151
A.2	Image pyramids: analysis and synthesis steps	151
A.2.1	Gaussian image pyramids	153
A.2.2	Laplacian pyramids	154
A.2.3	Steerable pyramids	156
	<b>List of Figures</b>	<b>164</b>
	<b>List of Tables</b>	<b>165</b>
	<b>Bibliography</b>	<b>181</b>



# Chapter 1

## Introduction

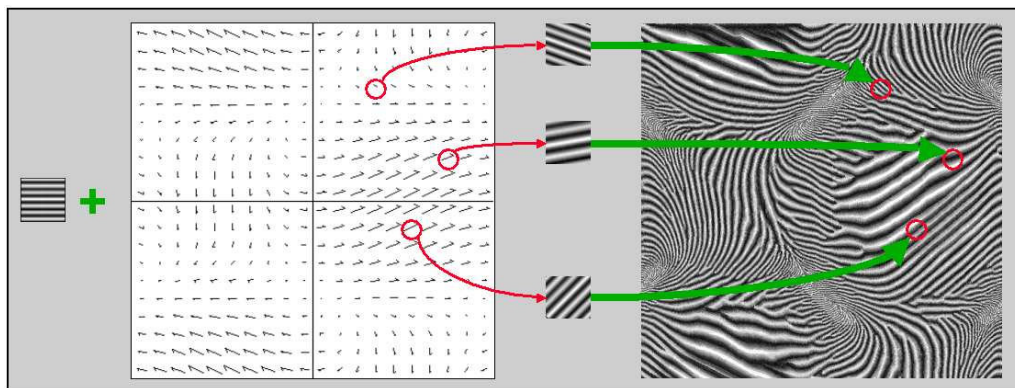


Figure 1.1: Using steerable texture synthesis for vector field visualization.

### 1.1 Motivation

Visualization is a fundamental field of research with uncountable applications, spanning from the fields of computer graphics and vision to the humanities. A key feature in visualization studies is thus the interdisciplinary nature of this research field, and it is interesting to note the numerous relative benefits and open directions where investigations can be guided.

In the last years, importance of visualization is constantly growing as, thanks to the fast computer advances, it is possible to collect and handle large data sets. Consequently, focusing on scientific visualization techniques still reserves much attention and, although several valid visualization methods exist, further investigation is required. A fundamental open issue is the need for more control. The broad variety of tasks and the different level of expertise of users also require degrees of freedom and adaptivity to allow customizing the visualization process, effectively representing data sets. Such features would be especially beneficial in computer vision and imaging applications as well as for textures generation. The need for local control and the ability to constrain the synthesis of textures are nowadays relevant issues due to the fundamental role that textures play in computer graphics, providing realism and variety in digital scenes and objects. Unfortunately, most synthesis approaches still just allow generating simple homogeneous and regular textures, leaving a great part of texture potential unexplored.

In this work, I propose novel techniques for the visualization of vectorial data sets, offering local as well as global control in the visualization process. I use statistical theory from texture synthesis and concepts from perception and cognition to optimize the resulting image and encode the information in the visualization. Furthermore, I introduce straightforward extensions to standard

texture synthesis algorithms, allowing the generation of constrained textures, field-driven textures and a variety of texture filtering and transformation effects.

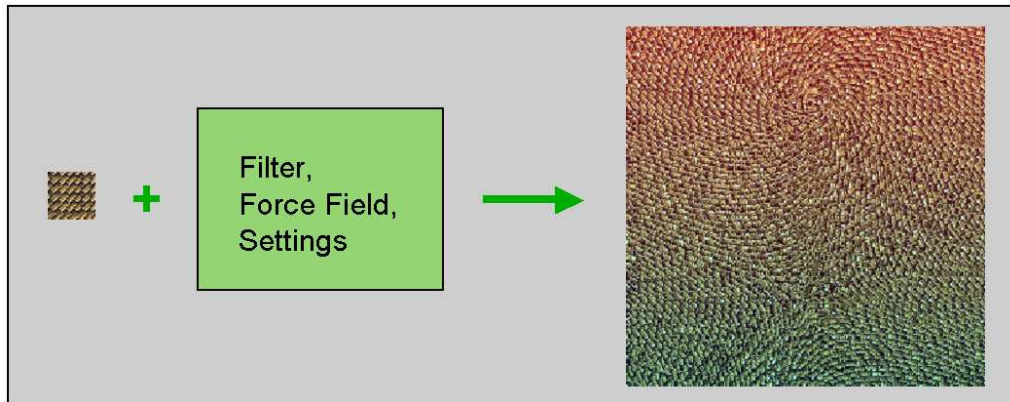


Figure 1.2: Generating controlled texture synthesis.

### Main idea

*Textures* are pattern-based images whose spatial distribution of pixels exhibits statistical, or structural properties, of both. Textures are characterized by a set of *perceptual visual dimensions* and are thus particularly suited as paradigm for powerful and flexible information mapping in visualization. *Steering* anisotropic textures along a vector field provides a flexible and effective visualization of the vectorial distributions. In this way, pieces of textures, or textural elements called *texels*, can be used as primitives for the scope of *vector field visualization* and *field-driven texture synthesis*. The main content of vectorial data sets and meaningful attributes are thus encoded using intuitive transformations over texture seeds.

This research introduces a novel approach to vector field visualization, using theory from statistics and texture synthesis. The work mainly deals with image generation, especially for application in the fields of multi-parameter scientific visualization (Fig. 1.1) and controlled texture synthesis (Fig. 1.2). Steering the synthesis process allows several degrees of freedom and the creation of a variety of effects in the resulting output.

## 1.2 Contributions

This work especially focuses on the design of visualization techniques, which are particularly useful for scientific multi-variate fields, offering a wide set of adaptive settings for intuitive data analysis. The visualization methodology is sample-based and makes use of textures as visual primitives to provide a flexible representation through information encoding. The visualization of a given data set is achieved through a synthesis algorithm, which uses theories from statistics and texture synthesis. The main novelty of this work is to provide a very flexible approach for the problem of scientific visualization. Using the proposed algorithm, complicated vector fields, steady or unsteady, fields with multiple variables and in higher dimensionality, as well as multi-fields or temporal evolution of flows, can be easily visualized having the advantages of both direct and dense visualization techniques. An adaptive scheme for layer-based presentation of the information is also proposed, allowing a task- and user-driven visualization technique. Perceptual theories have been taken into account in the design of the visual primitives and the mapping options. Being the method general, it results to be useful and straightforward to apply in many different application scenarios and for several traditional research disciplines. Moreover, the method can easily be extended, integrating further components in the presented visualization approach, or combining it together with other techniques.

This work concentrates on scientific visualization and controlled texture synthesis, it deals with diverse research areas, also attempting to find correlations between them. The resulting contributions have thus effects on disparate fields; a summary of the primary contributions and related publications is given in the following:

#### Vector field visualization:

- *Local control*: A novel algorithm that allows local control in vector field visualization is introduced; it provides several degrees of freedom for versatile information encoding [200, 192]. The proposed algorithm and techniques allow smooth continuous visualization of vector fields, both steady and unsteady.
- *Prominent field attributes*: Improvements are proposed to allow accurate representation of information also in particular areas of interest, such as singularities or critical areas of strong curvature. For this purpose, *ad hoc* specification of the input textural elements has been done to improve the results, guaranteeing smoothness and accuracy [204].
- *Multivariate data*: In case the number of variables increases, visualization of multiparameter data and flexible information encoding can be achieved using the proposed approach [200]. *Animation*: Extensions for temporal evolution of vectorial data are presented, and frames animation is possible using an adapted simple scheme of the algorithm [195]. A layer-based approach is also introduced [197].
- *Multi-field visualization*: The approach has been also extended for the visualization of multi-fields, and especially dual-fields, generating an effective interwoven representation of two co-existent vectorial data sets [205].

#### Texture synthesis:

- *Non-homogeneous textures*: Non-homogeneous textures can be easily *ad hoc* designed, texture blending, metamorphosis and mixture can be integrated. Manipulation and locality in the control of the image generation are provided [193, 203].
- *Steerable texture synthesis*: Through a control vector field, a given texture pattern can be modified, transformed, deformed according to field-driven transfer functions and aligned along the newly specified directions [201]. *Flow textures and texture animation*: Extensions of the algorithm allow the generation of flow textures and their relative animation [196].
- *Neighborhood models and weighting scheme*: For this new approach to constrained texture synthesis, as opposed to standard regular texture synthesis, novel neighborhood models have been proposed, together with appropriate weighting schemes, to enhance texture feature of directionality [204].

#### Perceptually motivated visualization:

- *Intuitive feature correspondence*: the approach allows a simple way to map field attributes onto visual representation for adequate mapping and representation of the vectorial information. The textural elements used as seed in the visualization approach yield a particularly easy and effective feature specification and encoding.
- *Design of visual primitives*: textures are well suited to serve as paradigm for effective visualization. Taking advantage of their various visual dimensions, expressive and meaningful information encoding is possible. In this work I propose some solutions, which can be interesting and useful for several applications. In this way it is possible to define a texture input set as primitive and powerful instrument for visualization [197].

### 1.3 Outline

An overview of the structure of this dissertation is here presented; the thesis is organized as follows:

After the brief introduction of Chapter 1 where motivation of this work is discussed and the main contributions and areas of investigation are presented, a discussion on prior related work is provided to introduce the context of this research.

The state of the art in Chapter 2 surveys perceptual issues (introducing some notions of perception, cognition and vision theory) and the most relevant techniques existing in the literature for the fields of texture synthesis and scientific visualization. I especially review some works that at most are related to my research or that in part gave me an inspiration in this research direction.

I then introduce my approach in Chapter 3. I start introducing the main novel ideas that are at the base of this work. I propose resulting visualization techniques and explain their usefulness and benefits. In the Chapter, details are given about the presented algorithm, its implementation steps and system structure. Mathematical notation as well as analytical formulæ are provided and the concepts are illustrated. The algorithm is at first explained in detail for simple vector fields in single resolution.

I extend the approach to its multi-resolution version and provide the necessary explanations in Appendix A, together with necessary basic notions from texture synthesis theory and multi-scale images.

Next, I investigate some solutions for optimization in Chapter 4. Here, improvement are proposed to enhance the directional properties of the represented images. Novel definitions for the specification of neighborhood structures and relative weighting schemes are presented.

In Chapter 5, I expand the approach to multi-variate multi-dimensional fields. The special case of temporal evolution of vector fields is analyzed as sub-case of multi-valued visualization. Brief functional formalism is given for a basic definition of tensorial entities, which can be treated in a vectorial way extending the steerable approach to multi-parameter visualization. The case of multi-fields visualization is also considered and solutions are presented. In particular, a novel approach to interweave co-existent distributions is proposed.

Especially motivated by the possible presence of multiple parameters, combinations of approaches are proposed and more sophisticated concepts for information extraction and encoding are surveyed and presented in Chapter 6. A discussion on possible visual representations to be used as primitives for the data mapping is offered. Vector field variables and attributes of interest are selected or extracted using a topological approach. Relative notions about eigen-analysis are here introduced and explained in more detail. An adaptive schema for information layering and proposals for customization options are also presented.

Similar considerations are valid for textures; in Chapter 7, I illustrate a novel approach to non-homogeneous texture generation, based on the field-driven synthesis which I called steerable texture synthesis. Extensions, results and various possible fields of application are shown, and a discussion is provided.

Finally, Chapter 8 concludes summarizing the research done, discussing the main novelty and contributions, considering benefits and limitations, and presenting possible future directions for improvements, extensions and further work.

## Chapter 2

# Context of related work

In visualization, and especially when dealing with scientific visualization of vectorial data sets, it is important to provide an expressive and effective representation of the data content; it is necessary to extract the most relevant part of the contained information and to successively map it onto adequate visual representations.

For this scope, considerations from different research fields need to be done; this is useful in order to better handle and represent complicated large data sets, and especially multivariate vector fields, highlighting their main features of interest. Perceptual criteria and concepts from cognition and vision validate several assumptions that are done in this thesis. Some existing synthesis procedures for automatic generation of textures have in part inspired the proposed approach: starting from statistical principles, standard synthesis theory has been here extended and adapted to produce vector field visualization. Various visualization techniques from the literature have been then taken into account to combine their benefits into a general and flexible visualization algorithm.

Therefore, I introduce in this chapter concepts from *perception and cognition*, following with theory from *texture synthesis* and finally reviewing some previous work done in the field of *scientific visualization*. Comments are finally given to place my research in this context, highlighting the main challenges, open questions and problems, which still need to be solved or deeper investigated.

## 2.1 State of the art in Perception and Cognitive Science

### 2.1.1 Introduction

In order to produce really effective visualization algorithms and tools, it is necessary to apply the basics of human visual perception to visualization. The general purpose of a visualization system is to transform numerical or functional data into images in which structures of interest become perceptually apparent, and through which it is possible to develop physical intuition of the visualized quantities. An understanding of perception can significantly improve both the quality and the quantity of information being displayed [235]. In order to convey information effectively, the design of visualization systems has to be influenced by perception theories and conducted user studies. To deeply understand when and why a given technique is effective is a very interesting point, which bases on perception principles. Perceptual theory is particularly motivated by multivariate visualization; with the increase in the amount and dimensionality of collected data, new approaches to the design of displays of such data have become essential. The general aim is to provide ways for raw data, and the statistical and mathematical structure they comprise, to be instantaneously understandable and, thereby, enable scientists to conduct *exploratory*, in addition to *confirmatory* analyses of the data<sup>1</sup>. On such basis, an advanced and combined visualization tech-

---

<sup>1</sup>Exploratory data analysis (EDA) was founded by John Tukey [222, 223], as opposed to confirmatory data analysis (CDA), which assumes an underlying structure to the data and proceeds with inference on the basis of such assumptions.

nique should be able to effectively convey correct information based on the characteristics of the data set being displayed and the questions being asked by the viewer. This can in many instances be predicted, based on an understanding of how the visual system perceives color, 2D regions, 3D shape, motion, and other salient features.

### 2.1.2 Gestalt principles of perception

*Gestalt* is a German word that means conformation, figure, shape; it refers to the way a thing has been placed or put together and is used to indicate coherent perception in psychology<sup>2</sup>. To the *Gestaltists* the context plays a fundamental role in perception. Fundamental gestalt principles are:

**Figure and Ground** The terms *figure* and *ground* explain the use of elements of a scene which are similar in appearance and shape and how to group them together as a whole (Fig. 2.1).



Figure 2.1: Figures and ground interchanging in the pictures.

**Similarity, Proximity or Contiguity, Continuity** *Similarity*: The principle of similarity states that things which share visual characteristics such as shape, size, color, texture, value or orientation will be seen as belonging together. *Proximity or Contiguity*: The principle of proximity or contiguity states that things which are closer together will be seen as belonging together. *Continuity*: The principle of continuity predicts the preference for continuous figures. In the picture to the left, in Fig. 2.2, the figure is perceived as two crossed lines instead of 4 lines meeting at the center. (Refer to Fig. 2.2).

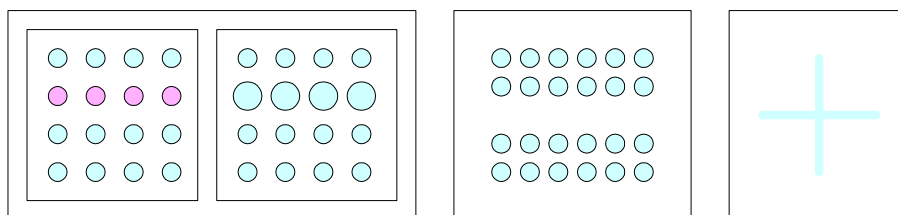


Figure 2.2: Similarity in color and shape (left), proximity (center), and continuity (right).

Tukey often related EDA to detective work, since the role of the researcher is to explore the data in as many possible ways as possible until a plausible "story" of the data emerges. Often, both the approaches are complementary, since human expectation contributes and plays an important role in data visualization. In fact, the confirmatory approach bases on cognition and an experimental logic, and experimental logic in turn results from observation logic.

<sup>2</sup>*Gestalt theory* first arose in 1890 as a reaction to atomism, which was the prevalent psychological theory of the time. *Atomism* examined parts of things with the idea that these parts could then be put back together to make wholes. *Atomists* believed the nature of things to be absolute and not dependent on context. *Gestalt theorists*, on the other hand, focused on the way our mind perceives wholes out of incomplete elements [15, 141].



**Closure, Area, Symmetry** *Closure*: The principle of closure applies when we tend to see complete figures even when part of the information is missing. *Area*: The principle of area states that the smaller of two overlapping figures is perceived as figure while the larger is regarded as ground. *Symmetry*: The principle of symmetry describes the instance where the whole of a figure is perceived rather than the individual parts which make up the figure. (Refer to Fig. 2.3).

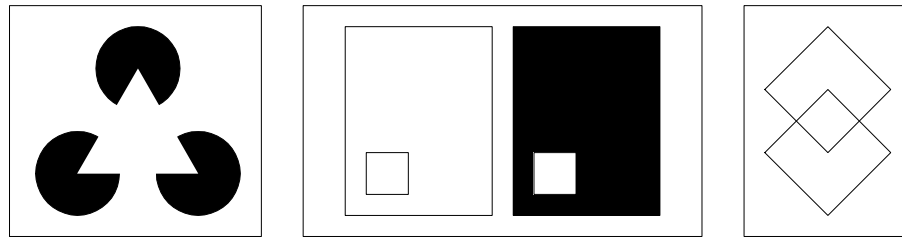


Figure 2.3: Concepts of closure (left), area (center), and symmetry (right).

Furthermore, the human visual system (HVS) is not immune to confusion. Specifically, it is susceptible to a number of illusions [162, 167]. These include:

1. The perceived size of an object may be influenced by its color.
2. The perceived hue of a color may be influenced by its saturation.
3. The perceived saturation of a color may be influenced by its hue.
4. The perceived depth of an object may be influenced by its color.
5. The perceived color of an object may be influenced by the color of surrounding objects.

Whenever possible, the conditions which give rise to these illusions should be avoided. Albers [6] motivates some color adjacency effects (Fig. 2.4) saying that "ground subtracts its own hue from color which it carries and therefore influences".

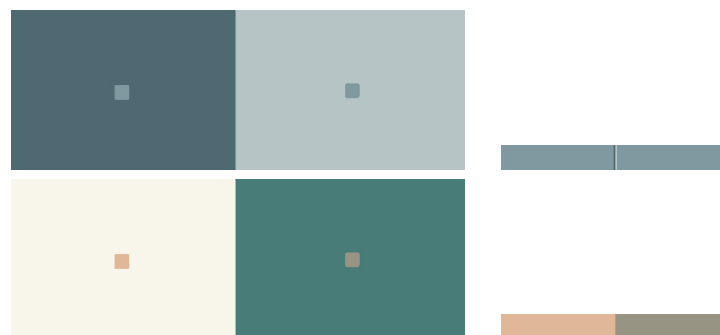


Figure 2.4: Joseph Albers' experiments: the same color (top-right) in the two central boxes looks different under different backgrounds (top-left); different colors (bottom-right) in the two central boxes look alike under some given backgrounds (bottom-left).

### 2.1.3 Human Vision Theory

"The complexity of the visual system and our incomplete understanding of its operation inhibit our ability to develop effective visualizations of complex multi-variate and multi-dimensional data"

[68]. A deep understanding about the way we see and perceive an image and identify its basic properties significantly has to contribute to create representations that take advantage of the human visual system. Briefly, *rods* and *cones* function as sensors that detect light falling on the retina<sup>3</sup>. Human vision theory dictates that neural signals from the rods and cones in the retina are transformed by neural connections in the visual cortex into three opponent color channels: a luminance channel (black-white) and two chromatic channels (red-green and yellow-blue). The luminance channel conveys the most information, letting us see form, shape, and detailed patterns to a much greater extent than the chromatic channels. Perception in the chromatic channels tends to be categorical. That is, we tend to place colors into categories like red, green, yellow, and blue. However, we see hues such as turquoise or lime green more ambiguously. There are three different types of cones, each sensitive to different frequencies of light. Peak sensitivity is in the yellow-green range. Figure 2.5-left shows how the eye can detect more shades of green than any other color. CIE chromaticity diagram of Fig. 2.5-right describes the range of human color perception.

**Color perception:** Human eyes are exquisitely sensitive to color variation: a trained colorist can distinguish among 1.000.000 colors, at least when tested under contrived conditions of pairwise comparison. Some 20.000 colors are accessible to many viewers, with the constraints for practical applications set by the early limits of human visual memory rather than the capacity to discriminate locally between adjacent tints. For encoding abstract information, however, more than 20 or 30 colors frequently produce not diminishing but negative returns.

Color scientists proposed several discrimination metrics (*e.g.* CIELAB) and color appearance methods (*e.g.* RLAB, Hunt and Nayatani's work). Another relevant theoretical point is that simultaneous contrast (the phenomenon by which perceived color is affected by surrounding colors) occurs in all three opponent channels. This can cause large errors when viewers try to read values in the data based on color. Hence, it is possible [117] to draw some conclusion regarding the design of color sequences:

1. If we want the color sequence to reveal form (such as local maxima, minima, and ridges), or if we need to display detailed patterns, then we should use a sequence with a substantial luminance component.
2. If we want to display categories of information, then we should use a chromatic sequence.
3. If we want to minimize errors from contrast effects, then we should arrange a sequence to cycle through many colors.

This is demonstrated by experiments that design and validate the guidelines in applied settings [234]. Beside color, textures are also a fundamental instrument in conveying shape and information. Evidence from the psychophysics [210] suggests that certain kinds of surface texture can facilitate shape perception, however, many points to exactly characterize this mechanism are still unknown. In addition, context problems have to be further investigated as well: it is necessary to discover complicated interactions between surface texture and shading, between texture orientation and surface geometry, and between aesthetics and convention.

#### 2.1.4 Preattentive, attentive and postattentive processing

Color and texture attributes can be classified in pre- and post-attentive features depending on the way the human visual system perceives them.

---

<sup>3</sup>Perception of intensity (rods) is logarithmic: *e.g.*, the amount of difference between intensities of 0.1 and 0.2 appears the same as the difference between 0.2 and 0.4. Color (cones) is perceived with less spatial resolution than intensity.

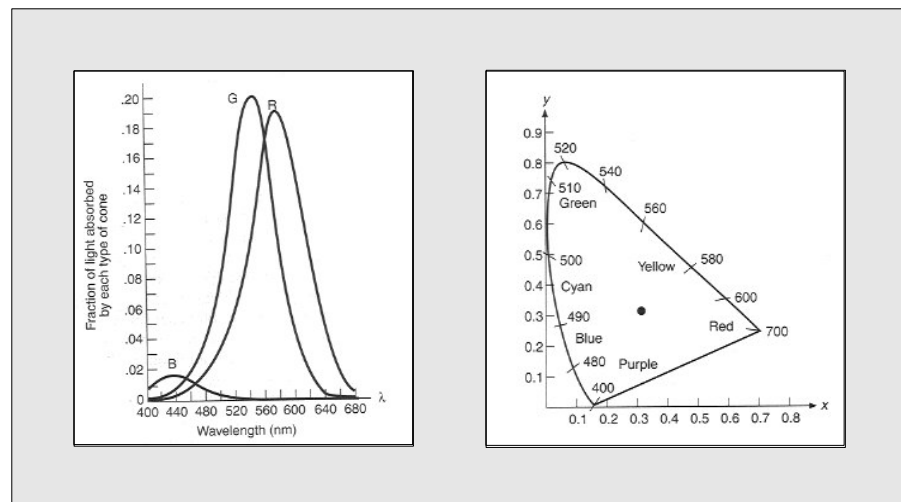


Figure 2.5: Color sensitivity (left) and CIE chromaticity diagram (right).

### Preattentive vision

Vision and psychology researchers have been investigating how the human visual system analyzes images. A fundamental result highlighted the fact that, during image observation, a limited set of visual properties are detected very rapidly and accurately by the low-level visual system. These properties were initially called preattentive, since their detection seemed to precede focused attention<sup>4</sup>. This suggests that certain information in the display is processed in parallel by the low-level visual system. On the contrary, when preattentive features are missing, subjects revert to serial search.

Experiments in psychology have used these features (Fig. 2.6) to perform the following preattentive visual tasks:

**target detection** : users rapidly and accurately detect the presence or absence of a "target" element with a unique visual feature within a field of distractor elements.

**boundary detection** : users rapidly and accurately detect a texture boundary between two groups of elements, where all of the elements in each group have a common visual property.

**region tracking** : users track one or more elements with a unique visual feature as they move in time and space.

**counting and estimation** : users count or estimate the number of elements with a unique visual feature.

### Julesz theories

Bela Julesz was also instrumental in expanding our understanding of what we "see" in an image [103, 104, 105, 109, 106, 107, 108]. Julesz's initial investigations focused on statistical analysis of texture patterns. His goal was to determine whether variations in a particular order statistic were

<sup>4</sup>We now know [82] that attention plays a critical role in what we see, even at this early stage of vision. The term preattentive continues to be used, however, since it conveys an intuitive notion of the speed and ease with which these properties are identified. Typically, tasks that can be performed on large multi-element displays in less than 200 to 250 milliseconds (*msec*) are considered preattentive. Eye movements take at least 200 msec to initiate, and random locations of the elements in the display ensure that attention cannot be prefocused on any particular location, yet viewers report that these tasks can be completed with very little effort.

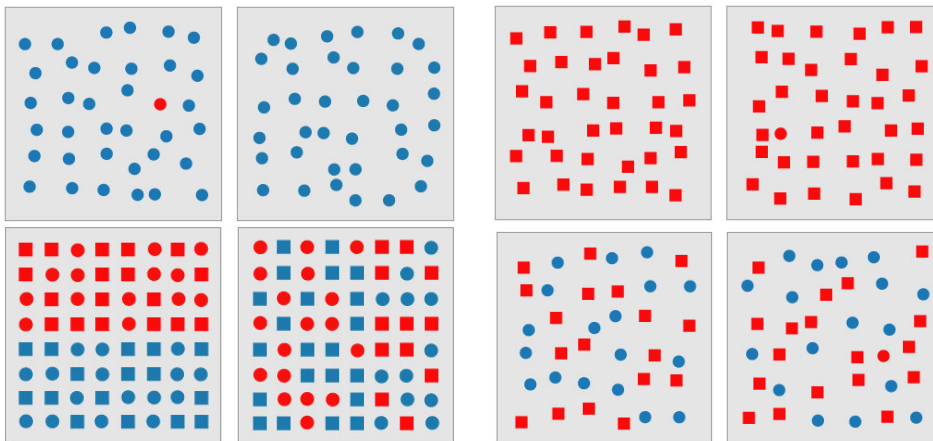


Figure 2.6: Treisman's preattentive experiments: target searching based on a difference of hue (up-left) and shape (up-right). In the bottom-left image, on the right, horizontal boundary detection is detected, while vertical boundary defined by conjunction of features (red circles and blue squares on the left, blue circles and red squares on the right) is not intuitive. In the bottom-right image, an example of a conjunction search for a target red circle is presented

seen (or not seen) by the low-level visual system. Examples of variations in order statistics include contrast (a variation in a texture's first-order statistic), orientation and regularity (a variation of the second-order statistic), and curvature (a variation of the third-order statistic). First-order variations were detected preattentively. In addition, some (but not all) second-order variations were also preattentive, as were an even smaller set of third-order variations. Based on these findings, Julesz modified his theory of how preattentive processing occurs. He suggested that the early visual system detects a group of features called *textons*. Textons can be classified into three general categories:

1. Elongated blobs (e.g., line segments, rectangles, ellipses) with specific properties such as hue, orientation, and width.
2. Terminators (ends of line segments).
3. Crossings of line segments.

Julesz believed that only a difference in textons or in their density can be detected preattentively. No positional information about neighbouring textons is available without focused attention. Like Treisman, Julesz suggested that preattentive processing occurs in parallel and focused attention occurs in serial.

### Further theories of preattentive processing

Besides the more popular Treisman and Julesz theories, a number of competing theories have been proposed to explain how preattentive processing occurs within the visual system. In general [81], four well-known models are: feature integration theory (Treisman), texton theory (Julesz), similarity theory [160], and guided search theory [249].

### Postattentive Vision

Research conducted studies [250] show that attention to different objects may allow a viewer to learn what is in a scene (if the objects are familiar and recognizable), but it does not allow the viewer to see the scene in a different manner. Wolfe argues that if multiple objects are recognized simultaneously in the low-level visual system, it would involve a search for links between the objects and their representation in long-term memory (LTM). LTM can be queried nearly instan-

taneously, compared to the 40 – 50msec per item required to search a visual scene or to access short-term memory (concepts derived from Cognitive Psychology<sup>5</sup>). In general thus, cognition plays a fundamental role in image understanding, also because viewers often see some of what they expect to see, they better understand what they are used to, and this is in large measure related or based on their past experiential explorations of the world. In 1613 Galileo Galilei [69] published the first telescopic observation of Saturn and its rings; he makes use of images to supplement words, even typeset among them, and such images, never seen before, become a new affective sentence element (Fig. 2.7).

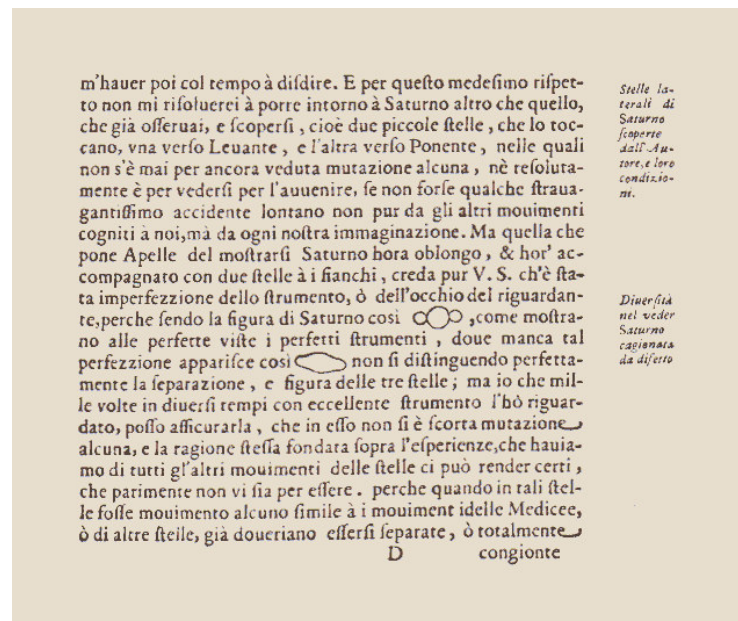


Figure 2.7: Galileo's text and illustration.

## Visual analytic

*Visual Analytic* has a particular focus on human intuition and interaction. Studying the users' mental behavioral process, in some cases concentrating not only on a successful outcome, but in spite of it on how a graph is interpreted, provides insights on the human analytical reasoning, useful to better design visualization techniques.

### 2.1.5 Comments: integrating perceptual issues in visualization

Vision theory and perception are very complex fields of study; deeper investigation is still required to answer questions and provide valid solutions to open problems. The disciplines that deal with these themes are numerous and they approach the problem in different ways. Providing a complete summary of those contributions is under the scope of this thesis, so I remand to perception and vision literature for deepening.

In summary, I take into account some well-accepted concepts to better motivate some choices done later. It has been proved and it can be stated that preattentive features, such as color, directionality and contrast, are easy and intuitive to perceive. For this reason, mapping data information onto such features and characterizing in this way texture primitives can contribute to effective visualization. Additionally, also cognition plays a fundamental role in perception: an arrow is for

<sup>5</sup>In cognitive psychology the human brain is assumed to consist of a number of processing centers that interact with one another. Experimental methods are designed to test models of these processing centers. The classic example of a cognitive theory is the division of memory into long term and short term components.

instance associated to information of direction and orientation, and is traditionally used in this way for iconic mapping. Similarly, further symbols and information encoding can be done. Using adaptive schemes to encode features in visual representations allows analytical reasoning and flexibility.

## 2.2 State of the art in Texture Synthesis

### 2.2.1 Introduction

Synthetic texture generation has been and still is an increasingly active area of research. Textures are fundamental in Computer Graphics, Computer Vision, Image Processing and many other disciplines, they enrich objects and computer generated scenes decorating them and conferring them realism as they can describe a wide variety of appearances. For this reason, textures are used to improve the perception of synthetic images or computer rendered objects, without increasing their geometric complexity. Sense of depth and curvature, as well as shape or material perception, are relevant examples. In these last years, lot of research focused on the analysis and synthesis of textures: especially due to storage requirements it is needed to synthesize large textures starting from a little sample.

### 2.2.2 Problem statement

The target of texture synthesis can be stated (at least under a probabilistic point of view) as follows: given a little sample of a pattern, or example texture, one wants to generate a new one that is a larger version of it, in arbitrary resolution, and which is perceived to be generated by the same underlying stochastic process by a human observer, appearing to be characterized by the same structure as the original one, still being sufficiently different from the original, which means that it should incorporate a sort of randomness to avoid repetitive artifacts (Fig. 2.8). That is, two texture images are perceived by human observers to be the same if some appropriate statistics of these images match: this has to be achieved by the synthesis algorithm. Briefly stated, the new texture should differ from the original one, yet having perceptually identical texture characteristics.

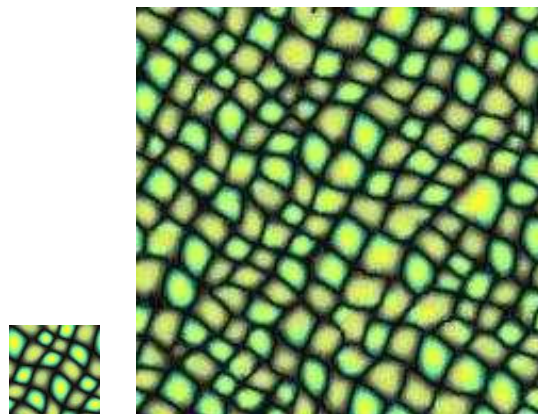


Figure 2.8: Synthesizing arbitrary resolution textures (right) starting from a little sample (left).

### 2.2.3 Categorization of textures

#### Textures in visual perception

The term "texture" originates from the Latin word "textura", from "texere", which means "to weave". According to common consideration [40], thus, textures can be defined as following:

"the term texture generally refers to repetition of basic texture elements called texels. A *texel* contains several pixels, whose placement could be periodic, quasi-periodic or random. Natural textures are generally random, whereas artificial textures are often deterministic or periodic. Texture may be coarse, fine, smooth, granulated, rippled, regular, irregular, or linear" [95]. Other possible definitions are to find in [247]: "textured regions are spatially extended patterns based on the more or less accurate repetition of some unit cell (*texton* or subpattern)", or in [183]: "textures are homogeneous patterns or spatial arrangements of pixels that regional intensity or color alone does not sufficiently describe. As such, textures have statistical properties, structural properties, or both. They may consist of the structured and/or random placement of elements, but also may be without fundamental subunits". Bar-Joseph [12] extends this definition in the following way: "a texture is a signal that exhibit the following property. Using any window of size larger than some critical size, the *information content* exhibited in the window is invariant to the window's position within the given sample". Another suitable definition of texture is that by Zhu *et al.* [256]: "a texture is a realization from stationary stochastic process with spatially invariant statistics".

It can be said that texture is the visual cue due to the repetition of image patterns, if seen in a deterministic way, which may be perceived as being directional or non-directional, smooth or rough, coarse or fine, regular or irregular, etc. (Fig. 2.9).

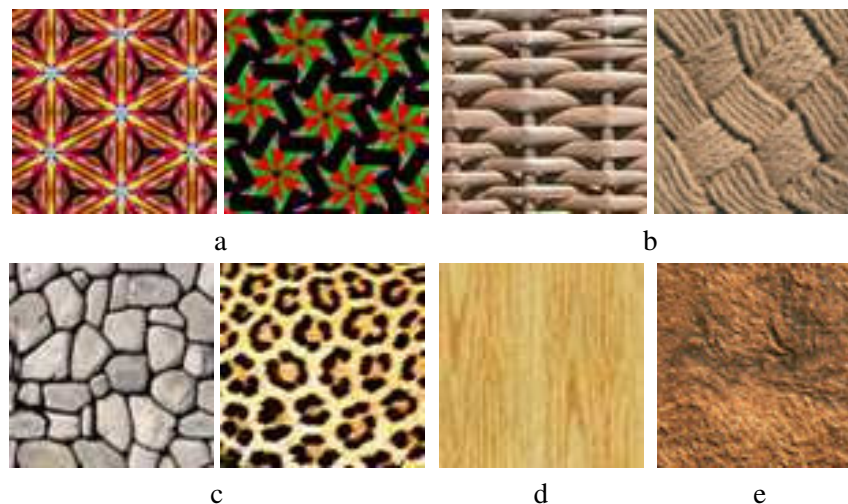


Figure 2.9: Samples of textures: regular (a), near-regular (b), irregular (c), near-stochastic (d) and stochastic (e).

More in general, textures can thus be divided in several groups, covering the texture spectrum from *regular* (strongly structured patterns), *near regular* (quasi-periodic) and *irregular*, to *near stochastic* and *stochastic* (random, such as many natural textures). Briefly, a *deterministic* or *structural* texture is characterized by a set of primitives (texture elements or texels) and a placement rule, while a *stochastic* texture does not have easily identifiable explicit primitives. Many natural textures that occur in the real world are neither deterministic nor stochastic, but they are characterized by some mixture of both these aspects.

### Texels and Textons

A *texel*, or texture element, represents the smallest graphical element in a two-dimensional texture. A texel is similar to a pixel (picture element) because it represents an elementary unit in a graphic. In stochastic textures, for example, each such texel behaves like a seed, and a texture is generated by randomly scattering the seeds over the image lattice. In a stochastic texture, texels appear with no explicit placement rules and have mostly weak spatial interrelations. In this class, a synthetic texture can be formed by sampling, repeating and randomly placing of the distinct training texels. In structured or semi-structured textures, instead, repetitions of various texels create periodicity,

structure and symmetry in a texture. A special placement rule is derived to model spatial relationship between texels, *i.e.* how the image locations of multiple occurrences of a single texel or different texels are related in forming a texture pattern.

As alternative term, in psychology, basic texture elements are called *textons*. They refer to small objects or characteristic regions that comprise a texture. Research shows that only the difference in textons or in their density can be detected pre-attentively by human early visual system [106]. Conceptually, each texton, as a feature descriptor in the spectral domain, represents a particular statistical spectral feature describing repetitive patterns of a texture in the spatial domain.

Nevertheless, although most researchers use the terms *texel* and *texon* interchangeably, a useful distinction can be made. Texels, by analogy with pixels, define a partitioning (or tiling) of the texture, with each texel having a finite, non-overlapping spatial extent. On the other hand [134], Julesz's textons serve the role of statistical features without concern for overlap.

## 2.2.4 Classification of approaches

Texture synthesis methods can be divided in groups according to several criteria. A common way to classify texture features, and texture analysis and synthesis methods, is for instance to loosely divide them into two categories - *statistical* and *structural* [78], plus *model-based* and *signal processing* methods [219]. Additionally to this first classification, methods can be categorized in base of the kind of approach used to define a model for the texture: *physical simulation*, *MRF sampling*, *feature matching*. Most synthesis approaches have been thus designed resulting to be mainly specialized or best suited to synthesize one or another family of textures.

### Statistical methods

*Statistical methods* define textures in terms of local grey-level statistics which are constant or slowly varying over a textured region. Different textures can be discriminated by comparing the statistics computed over different sub-regions. With statistical methods, the stochastic properties of the spatial distribution of grey levels in an image are characterized. Many of the methods are based on the fact that the human visual system uses statistic features for texture discrimination, which are broadly classified into first-order, second-order, and higher-order statistics. Statistical algorithms treat textures as realizations of probability distributions, and generate new textures by sampling from such distributions. The main challenge is in fact to model how to estimate the stochastic process that describes the input pattern, and to define the sampling procedure to produce the new texture from a given model [239, 241]. For example, statistical random fields like Markov random field and Gibbs sampling are widely employed to model textures [153].

### Structural texture models

*Structural (geometrical) texture models* try to determine the primitives which compose the texture. The extracted primitives and their placement rules can be utilized not solely to recognize textures but also to synthesize new images with a similar texture.

### Model-based texture methods

*Model-based texture methods* try to capture the process that generated the texture. With model-based features, some image model is assumed, its parameters estimated for a sub-images, and the model parameters or attributes derived from them, are used as features. There are currently three major model-based methods: *Markov Random Fields (MRF)* by Cross and Jain [40] and Dubes and Jain [51], *fractals* by [64, 147, 126], and the *multi-resolution autoregressive (AR) features* introduced by Mao and Jain [136]. For detailed discussions of image models see also Kashyap [110], and Chellappa *et al.* [36].



### Signal processing methods

*Signal processing methods* are another way to model a texture. They analyze the frequency content of the image. They perform frequency analysis of the textures using spatial filters or through filtering in the frequency domain [125].

In human visual psychophysics research, the focus of texture perception studies has been on developing physiologically plausible models of texture discrimination. These models involve determining to which measurements of textural variations humans are most sensitive. Typically based on the responses of oriented filter banks, such models are capable of detecting variations across some patches perceived by humans to be different textures [105, 105, 135, 18, 17, 19, 16]. The approach of De Bonet [26] later uses these resulting psychophysical models to provide constraints on a statistical sampling procedure.

### Physically-based approaches

Primary relevant approaches represent textures via developing procedural models that emulate the underlying physical generative process of the texture (*physical simulation*) [53]. They use models based on *cellular texturing* [62, 251] and *reaction diffusion interactions* to simulate seashells [248] or animal skin and fur [224]. Nevertheless, they are hard to use, as they are difficult to control and require hand-crafted parameters, and are only limited to and suitable for certain specific textures (such as biological patterns, wood, marble, or animal skin). A main advantage of such *procedural methods* - noise-based - is that they are based on a underlying mathematical function and they may be evaluated in any order and sampled arbitrarily.

### Feature or histogram matching

Some algorithms model textures as a set of features, and generate new images by matching the features in an example texture. They reduce the amount of calculation and are more efficient than MRF-based methods. Some works consider textures as samples from probabilistic distributions and depend on the structure of the probability density estimator used in the sampling procedure [26, 256]. In [133], multi-resolution *Markov Random Fields* are used to model relationships between spatial frequencies within texture images. Heeger and Bergen [83] model stochastic textures by matching marginal distributions (or *histograms*) of the image pyramids (filter outputs). They iteratively modify a random noise image so that its intensity histogram matches the histogram of the sample texture across each of the subbands in a pyramid representation of each image. Their technique is capable of generating highly stochastic textures but fails on more structured ones.

In the work of De Bonet [26], the sampling methodology is based on the hypothesis that texture images differ from typical images in that there are regions within the image which, to some set of feature detectors, are less discriminable at certain resolution than others. By rearranging textural components at location and resolution where the discriminability is below threshold, new texture samples are generated which have similar visual characteristics. He uses multi-resolution image pyramids to match texture statistics at multiple frequencies simultaneously. De Bonet [26] synthesizes then new images by randomizing an input texture sample while preserving the cross-scale dependencies. This method works on structured textures, but it can produce visible boundary artifacts if the input texture is not tileable.

Simoncelli and Portilla [182, 154] generate textures by matching the joint statistics of the image pyramids. Their method can successfully capture global textural structures but fails to preserve local patterns.

### Texture synthesis from samples: practical approaches classification

Although a number of different classifications, as described above, are possible, often texture synthesis methods are more practically and generally grouped (*e.g.* [48]) in *patch-based* and

*pixel-based*, with the further extension to *hybrid methods*. Both these approaches compute global statistics from sample textures and then generate new texture images, which hold the same statistics. This characterization is mainly based on the way different elements are cut and copied from the input to the output texture. Pixel-based methods perform an iterative processing for every output pixel, while patch-based methods copy larger patches from the input to the output. This results in different performances and applicability to classes of textures. Obviously, further sub-classifications are possible.

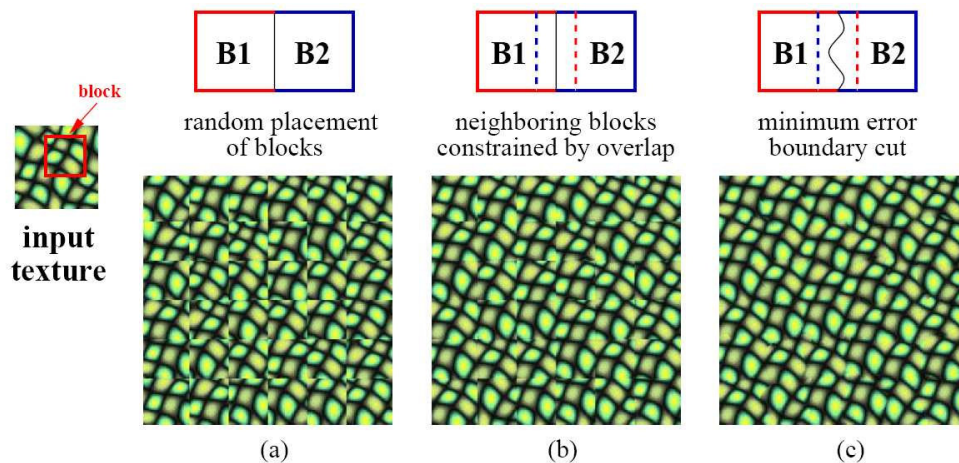


Figure 2.10: Patch-based approach: Efron and Freeman's *image quilting* (image courtesy of Alyosha Efros).

### Patch-based synthesis

Patch-based approaches basically consist in tiling the sample texture and then use the blocks derived from the sample to compose a larger output texture. Obviously, the main problem of such approach is that, even if the sample allows seamless tiling, this will produce repetitive artifacts. Patch-based methods result to be particularly fast and computationally efficient, since they synthesize the output textures in patches. Therefore several methods also exist, as extension, for the synthesis of textures directly on three dimensional surfaces (see also below).

Efros and Freeman [54], using *image quilting*, stitch together random blocks from the sample image and modify them in a consistent way. The block substitution is here optimized by piecing together small adjacent patches of existing images and reduce boundary artifacts by minimizing the error on the boundary cut where the patches join (minimum error boundary cut). In Fig. 2.10, square blocks from the input texture are patched together to synthesize a new texture sample: (a) blocks are chosen randomly, (b) the blocks overlap and each new block is chosen so as to *agree* with its neighbors in the region of overlap, (c) to reduce blockiness the boundary between blocks is computed as a minimum cost path through the error surface at the overlap. Liang *et al.* [74, 128] use similar patch-based sampling, alpha-blending the overlap regions (feathering). Interesting alternative methods to generate images from examples are the fast patch-based technique of Xu's *chaos mosaic* [75] and Kwatra's [120] *Graphcut*, and more recently the *Wang Tiles* by Cohen *et al.* [38].

### Pixel-based synthesis

Pixel-based synthesis algorithms synthesize textures pixel by pixel, which makes them rather flexible and easy to extend for application in different areas.

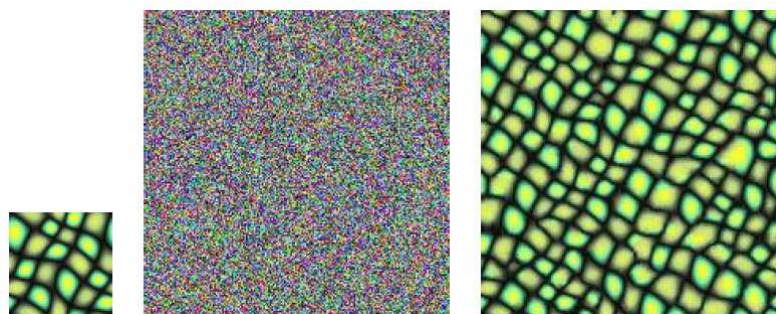


Figure 2.11: Wei and Levoy's pixel based approach (image courtesy of Li-Yi Wei).

The new texture is here generated in a pixel-by-pixel fashion. *Non-parametric sampling* by Efros and Leung [55] iteratively finds and copy the best matching pixel from the input to the output. They model textures as a Markov Random Field and perform an exhaustive search for each synthesized pixel. The neighborhood of a pixel is defined by a square window around the pixel, and the size of this window is a constant that can set by the user. This parameter interprets how stochastic the texture is and it roughly corresponds to the "degree of randomness" in the synthesized texture. The work of Wei and Levoy [241] is a fundamental step in this direction: they model the texture using *Markov Random Fields*, combining the classical Efros and Leung's approach [55] and De Bonet sampling theory on *image pyramids* [26]. They use now a fix-sized neighborhood, and use deterministic searching. They generate the output texture pixel by pixel, determining each pixel value so that local similarity is preserved (Fig. 2.11). Starting with an example texture patch (left), and a noise texture (center), they force this random noise texture to look like the sample by transforming each pixel in a raster scan ordering (right). The initialization of the output image with random noise (typically uniform white noise) seeds the algorithm with the necessary amount of randomness or entropy required for the synthesis of a realistic texture, introducing uncertainty in the neighborhood matching process, to avoid repetitive artifacts. The technique is based on neighborhood comparison and reproduces the input texture by synthesizing colors at each vertex in the output. Vertices are colored by flattering their local neighborhoods and regularly sampling the already synthesized texture. At each step, a best matching pixel is found inside the input sample, which is both stationary and local, and copied over to the current output vertex. Techniques to speed up these calculations have been proposed (*multiresolution*, *tree-structured vector quantization TSVQ*). Particularly effective is the *coherent search* developed by Ashikhmin [10], and later generalized by Tong *et al.* in the *k-coherent synthesis* [212], which reduces the search space significantly. The synthesis process is faster, but only suits particular types of textures well. Ashikhmin recognizes that the position in the input, around which nearby output pixels were copied, are likely good matches for new output pixels. He 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. Hertzmann *et al.* [87] (*Image Analogies*) combine [241] and [10]; they use *principal component analysis PCA*, and approximates *nearest neighbor search ANN* to accelerate the search process. They additionally use a parameter to weight and choose between the contributions of each (*texture-by-numbers*). Recent pixel based approaches almost achieve interactivity: the *jump maps* of Zelinka and Garland [253, 254] accelerate texture synthesis, partitioning the synthesis task into a slower analysis phase (pre-processing stage) and a faster synthesis phase. The key idea of their approach is to divide the method in two phases: in the first one, a *jump map* is generated, in the second one, the jump map, which is a lookup table, is used to assign addresses from the input texture. At each pixel in the input texture, the jump map stores a set of links to pixels with similar neighborhoods. Each jump is weighted according to the similarity of the neighborhoods. In this way, *via* an approximated pre-computing of the possible matches, they avoid performing the costly neighborhood matching at run-time.

## Hybrid synthesis

The hybrid method by Nealen and Alexa [142] is a compromise between the above two classes of methodologies: it starts copying larger patches from the input to the output, as done in patch-based approaches, and successively refines the output at a per-pixel level, where the error reaches a given user-defined tolerance value.

In the hybrid algorithm, the use of large patches improves the reproduction of global structure, and the remaining errors in the overlap regions are eliminated using pixel-based re-synthesis (Fig. 2.12). The method uses the Fourier domain for finding the best match, that is, the patch from the input texture which minimizes overlap error with the existing synthesis result. The approach is based on two steps: *overlap re-synthesis* and *adaptive patch sampling*. In the first step each new patch overlaps already synthesized regions. In these overlap regions an error is computed for each pixel and mismatched pixels are re-synthesized using a per-pixel texture synthesis strategy. To ensure sufficient valid neighborhoods for these pixels, they are ordered using morphological dilation of the valid regions. In the second step they adapt the patch size so that the error in overlap regions is bounded. The error bound allows a trade-off between preserving global structure and avoiding detail artifacts: increasing the error threshold leads to generally larger patches and a higher probability that large structures are preserved, however, at the cost of having more invalid pixels in the overlap region that need to be fixed, which is not always possible. The method has been later speeded up using a k-nearest neighbor data structure [143].

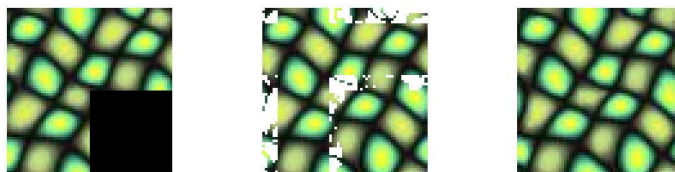


Figure 2.12: Hybrid texture synthesis (image courtesy of Andy Nealen).

## Further texture synthesis methods

**Texturing Surfaces:** Most 3d texturing methods are direct extension of pixel- and patch-based techniques. In [225], Turk proposes the use of oriented patches for surface texture synthesis (Fig. 2.13). Neyret and Cani [144] map 2D textures on 3D surfaces, but extension to anisotropic patterns is not obvious. Praun *et al.* [157] propose *Lapped Textures*, an approach that uses overlapping irregular sample patches and iteratively copies them over a surface, using then texture blending to help hide boundary artifacts. They extend the *chaos mosaic* to surfaces with a pre-computed vector field to direct anisotropy. Anyway patterns should contain enough high frequency detail and natural irregularity to avoid patches overlap artifacts. Ying *et al.* [252] synthesize per-texel using a texture atlas of the polygonal mesh. Gorla *et al.* [72, 73] add textures to surfaces, letting the dominant orientations of texture patterns follow the surface shape along the principal directions of curvature. Soler *et al.* [185] demonstrated how a mesh can be seamlessly textured with only the input texture and a set of texture coordinates for each vertex. Zhang *et al.* [255] realize a progressively variant synthesis for texture mapping on surfaces. Bhat *et al.* [23] later synthesize a variety of geometric textures on a model surface. Another significant approach that synthesizes textures directly on surfaces is from [242]; they also densely tessellate the input mesh and then perform a per-vertex color synthesis (Fig. 2.14).

**Bidirectional Texture Function Synthesis:** In Liu *et al.*'s work *reflectance texture synthesis* [129], geometry is recovered, synthesized and used to generate templates for each viewing and lighting setting. These templates then guide the actual BTF synthesis, thereby preserving global *mesostructure*. Tong *et al.* [212] extend Ashikhmin's algorithm [10] by adding the k-nearest neighbors to each candidate pixel of the Ashikhmin-set (k-coherence search).

**Geometrically Motivated Texture Synthesis:** Dischler and Ghanzfarpour have published

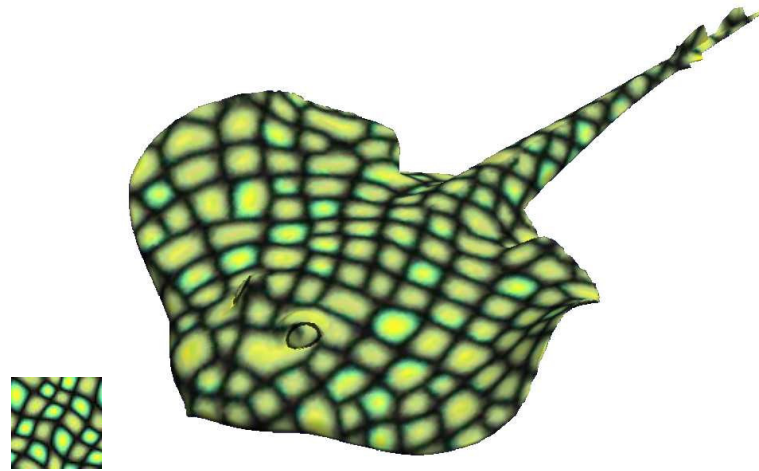


Figure 2.13: Texturing surfaces (image courtesy of Greg Turk).

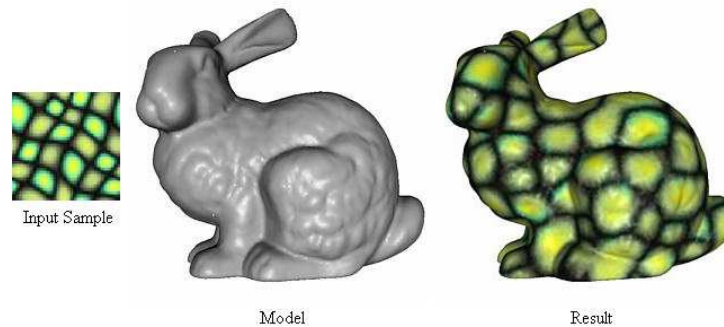


Figure 2.14: Wei and Levoy's surface texture synthesis: given a texture sample (a) and a model (b), they synthesize a similar texture directly over the model surface (c) (image courtesy of Li-Yi Wei).

many geometrically and structurally motivated algorithms which are of semiprocedural nature, yet also resemble the input so closely that they could be classified as texture synthesis algorithms. In their work [47], a highly structured texture is analyzed with some user intervention, from which the algorithm generates seemingly random structured texture. Texture Particles [48] are gathered by segmentation and the analysis of spatial arrangements using morphological operations. These are then procedurally assembled in the synthesis stage. They extract particles from the sample texture by color thresholding and then paste them onto surfaces.

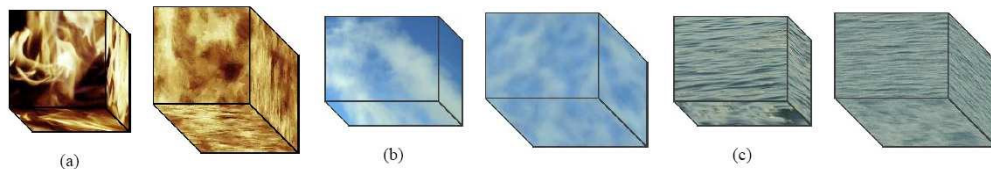


Figure 2.15: Wei's solid textures from 2d view: (a) fire, (b) smoke, (c) ocean waves (image courtesy of Li-Yi Wei).

**Further approaches:** Besides standard methods for texture synthesis, in the last years there were several attempts to introduce some kind of user control in the synthesis process, also turning

to different applications in imaging. Perlin [148] and Peachey [146] independently invented the *solid texture* as a function that returns a color value at any given point in 3d-space. Solid textures are ideal for simulating surfaces that have been carved out of a block of material such as wood or marble, as done for *stereological textures* [94]. Perlin also introduced the 3D noise function, which can be used to create patterns such as water waves, wood grain and marble. The early work of Perlin ("An Image Synthesizer") already stated the need for synthesizing naturalistic looking textures. He creates stochastic functions that, once combined, yield a remarkably rich set of visual textures. The use of a training set for image generation is a solution to a broad class of application, as for example *texture transfer* [54, 10] by Efros & Freeman (Fig. 2.17) and Ashikhmin (Fig. 2.16), and *image analogies* [87]. Ashikhmin produces the effect of rendering a given image with the texture appearance of a sample image. Hertzmann *et al.* [87] process images by examples: their method involves two stages: a design phase, in which a pair of images (one is a filtered version of the other) is presented as training data, and an application phase, in which the learned filter is applied to a new target image in order to create an analogous filtered result. Brooks and Dogson [31] propose *self-similarity texture warping* to replicate operations (painting, warping, cloning) globally over the image for texture editing. Freeman *et al.* [66] perform *super-resolution by example*: they train using different pairs of images with low and high resolution. A further approach for creating textures with local variations is through chemical or biological simulations. Turk [224] proposes generating textures (on surfaces) using reaction diffusion differential equations; this approach works particularly well for textures modelled after organic processes. Liu *et al.* [130] analyze near-regular textures that deviate from periodic tilings. Relevant applications are also *constrained texture synthesis* [241] or *fragment-based image completion* [50], where damaged images can be corrected, or a unwanted foreground elements or portions of the original image can be removed and then replaced by synthesizing the background texture over them. Bertalmio *et al.* [20, 21] also fill in missing regions of a picture, for example for restoration and reconstitution - *inpainting* - of images. Inpainting is a technique for extending an image across the pixel region damaged or left blank from removing unwanted objects from the picture. Their approach works for the structure and for the texture (image details) of the picture as well. Wei [240] suggests creating *solid textures* (Fig. 2.15) from multiple 2D-views and *texture metamorphosis*: from a pair of samples, an average one is computed and used for the transition area. Zhang *et al.* [255] also investigate metamorphosis by weighted blending and texon masks, inspired by the work of Tonietto and Walter [213, 214], who use synthesis of textures in several scales to offer tileable textures with arbitrary size. However, in those works usually only a couple of parameters (typically just the resolution) can be modified and no arbitrary control is offered.

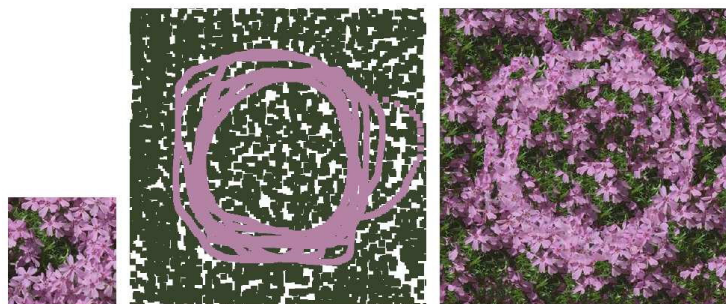


Figure 2.16: Texture transfer (image courtesy of Michael Ashikhmin).

### 2.2.5 Comments and open issues

Texture synthesis approaches seem to have reached a mature state. Recently developed algorithms are stable and efficient and each of them is particularly effective for given classes of textures. Both the patch- and pixel-based approaches, plus the hybrid method, which is basically patch-based, are characterized by advantages and disadvantages, and the one or the other could be preferable to



Figure 2.17: Efros and Freeman’s *texture transfer*: they transfer the rice texture (left) onto another image (center) for a strikingly different result (image courtesy of Alyosha Efros).

solve a specific synthesis problem, according to the texture and task under consideration.

Patch-based synthesis algorithms tend to be faster and more stable, and do not suffer from blurring and garbage growing. They produce reasonable results for a wide variety of texture classes, they preserve global structure, but often introduce unwanted visual artifacts along patch boundaries. They are, furthermore, less flexible since they generate textures by copying whole patches from the input. Pixel-based synthesis algorithms, on the other hand, tend to maintain a consistent texture impression, and, what is of relevance for my research, they can allow more freedom in applying local control up to a per-pixel level. Pixel-based methods are extensible in several directions and this potential controllability is one reason why I use some of these concepts for my visualization approach.

### Steerable solution

One of the main open issue in texture generation is the integration of more degrees of freedom in the synthesis process. Adding local control and user intervention is still challenging and promises a variety of interesting applications. Although some of the works cited above propose ideas to edit or modify textures according to the user desires, they provide little control over the amount of texture variability. For this reason, a lot of work and a deeper investigation still need to be done in this direction. Especially the generation of non-uniform textures remains an open problem, which deserves lot of interest and promises significant applications. *Non-homogeneous* textures are to find everywhere in nature and in the real world: while we often just have a synthetic sample to enlarge using a synthesis procedure, a complete method, which could also be able to directly generate a non-homogeneous texture is still missing. This should be flexible enough to allow the user to define, choose or design variables and transformation functions in an appropriate manner, also taking into account lighting effects and surrounding information. The main limitation in standard texture synthesis methods is the lack of *local control*. Moreover, homogeneous textures often result too synthetic, while non-homogeneous textures can better represent real appearance of materials, which is usually not regular. Being able to generate non-uniform, varying patterns is important for instance when joining two different textures, and when desiring to vary scale, orientation, color, shape and further texture attributes to *ad hoc* create new textures. Some proposal for incorporating local control and degrees of freedom in the texture synthesis process are proposed in Chapter 3 and 7, together with some reference to pioneer related works and a more detailed discussion on possible applications. It would be of relevance in computer graphics to extend these ideas also to animations and deformations of textures, allowing arbitrary control and manipulation over a given starting pattern. I believe this task, although demanding, can results in many interesting and

relevant applications.

## 2.3 State of the art in Vector Field Visualization

### 2.3.1 Introduction

Vector field visualization plays a fundamental role in science and engineering. Almost all scientific disciplines deal with several kinds of vectorial data sets: electromagnetism, physic, computational fluid dynamic, fluid mechanics, dynamical systems, climate modelling are just a few relevant examples. Hence, the analysis and visualization of vector fields have drawn constantly increasing attention in the last decades. Through the visualization of vectorial data sets, a wide variety of physical phenomena can be described and visually represented. Often vector fields exhibit quite a complex structure, and their equations or abstract data sets are difficult to be interpreted; consequently, the information contained in the data can be studied and analyzed by means of adequate visualization approaches.

### 2.3.2 Definitions and notation

The term *field* is used to refer to a process which associates a physical quantity with each point in a region of space. Fields can be scalar, vector, or tensor valued. A *vector field* on  $\mathbb{R}^n$  is a map or a function  $\Phi : A \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$  that assigns an  $n$ -dimensional vector  $\Phi(\mathbf{x})$  to every point  $\mathbf{x}$  in its domain  $A$ .

In two dimensions though, we speak about *planar vector fields*. Hence, a vector field in the plane is defined in a planar domain  $D \subset \mathbb{R}^2 = \{(x, y) | x, y \in \mathbb{R}\}$  and is a function  $\Phi$  that assigns to each point  $(x, y)$  in the plane a vector  $\Phi(x, y) = \langle \Phi_x(x, y), \Phi_y(x, y) \rangle$ .

$$\begin{aligned} \Phi : D &\rightarrow \mathbb{R}^2 \\ (x, y) &\rightarrow \begin{pmatrix} \Phi_x(x, y) \\ \Phi_y(x, y) \end{pmatrix} \end{aligned} \quad (2.1)$$

Hence, visualization methodologies should be able to adequately depict the vector field values  $\Phi(x, y)$  at the various locations  $(x, y)$  in the output domain. Common requirements for the visual representation include accuracy, intuitivity, locality. A plethora of different techniques approaches the problem in various ways, depending on the specific application or audience. In the following, I present existing techniques classification and discuss features and limitations of the approaches.

### 2.3.3 Techniques classification

A categorization of the numerous existing techniques for vector visualization can be done, usually according to the different requirements of the user and application. A common conceptual classification, besides classifications done on the base of dimensionality or steadiness *vs.* unsteadiness, distinguishes visualization techniques in

- *direct*
- *geometric*
- *feature-based*
- *texture-based*

Nevertheless, as explained below, such families of methods are often overlapping and no complete distinction can be actually done, since some of these approaches conceptually work in a similar way, or originate the ones from the others.



### Direct visualization, global imaging techniques

*Direct*, or *global*, visualization is performed using as direct as possible translation of the data into visualization cues, such as by drawing arrows or color coding velocity at discretely sampled grid locations. Such kind of visualization aims at providing immediate investigation of the vectorial data, without a lot of mental translation effort. They present the complete data set, or a large subset of it, at a low level of abstraction. The mapping of the data to a visual representation is direct, without complex conversion or extraction steps. Difficulties may arise in case the long-term behavior induced by the data needs to be investigated, this could then require cognitive integration of visualization results. Another problem is given by the fact that too sparse representation of the field may lead to a loss of detail and information (for instance a singularity could be missed and a small vorticity could not be displayed).

**Glyphs, arrow plots:** Arrow plots provide the simplest way for vector field visualization; an arrow can be used to depict field direction at a unique point (Fig. 2.18). Proper scaling and coloring allow encoding scalar quantities. Arrows are a very natural and intuitive cue to visualize vectors. When arrows are not normalized (unit length), and thus have varying length, which is proportional to the flow velocity, they can visualize flow velocity besides flow direction. This is the so called *hedgehog* visualization, because of the bristly result [115]. Extension to the temporal domain are possible, but with lack of continuity. Further simple direct visualization techniques map a line or a glyph to each sample point in the field, oriented according to the flow field. Usually a regular placement of cues is used in 2d, for example, on an evenly-spaced Cartesian grid.

**Color coding:** Together with arrow plots, color coding is a widely distributed, standard technique for visualization of vectorial data sets. A common solution is to map vector attributes such as velocity, pressure, or temperature to color. Since color plots are widely distributed, this approach results in very intuitive depictions. At this point, it is crucial to carefully choose the color scales for the mapping with respect to perceptual differentiation. Color coding for 2d flow visualization extends to time-dependent data very well, resulting in moving colors plots according to changes of the flow properties over time.

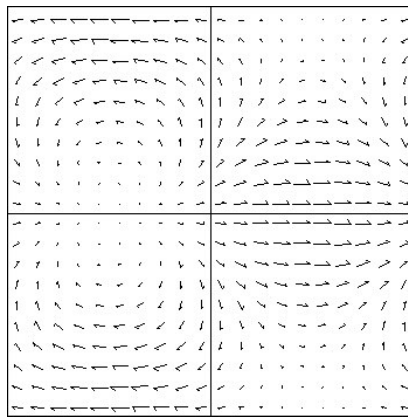


Figure 2.18: Direct visualization.

### Geometric visualization

Many vector field visualization algorithms use spatial resolution to represent the vector field. These include sampling the field, such as with streamlines or particle traces, and using icons at every vector field coordinate. Streamlines and particle tracing techniques critically depend on the placement of the "streamers" or the particle sources. Depending on their placement, eddies or currents in the data field can be missed. Consequently, the main disadvantage of these method is the loss of accuracy. Icons, on the other hand, do not miss data, but use up to a considerable amount of spatial resolution limiting their usefulness to small vector fields. Here the main disadvantage is the

loss of controllability. *Geometric* visualization techniques entail extracting geometric objects for which their shape is directly related to the underlying data. Integration-based approaches can be used to integrate the data and extract integral geometric objects (such as streamlines, streaklines, pathlines) that reflect the properties of the field, and that are finally visualized in place of the complete data-set for a better communication of the field behavior. Here the seeding problem must be addressed to realize a satisfying distribution of the integral objects. In particular, a best choice of initial conditions needs to be performed.

*Streamlets*: if flow vectors are integrated for a very short time, streamlets are generated.

*Streamlines*: if longer integration is performed, streamlines or integral lines or curves are gained. They are a natural extension of glyph-based techniques and offer intuitive semantics: users easily understand that flow evolves along integral objects. They are tangent to a vector field in every point.

*Streaklines, timelines, and pathlines*: when unsteady flow data are investigated, several distinct integral objects are used for flow visualization. A *pathline* or particle trace is the trajectory that a single particle follows in a fluid flow. A *timeline* joins the positions of particles released at the same instant in time from different insertion points, *i.e.*, joints points at a constant time  $t$ . A *streakline* is traced by a set of particles that have previously passed through a unique point in the domain (the set of particles emitted from the same insertion point). *Streaklines* relate to continuous injection of foreign material into real flow. Note that in steady flows pathlines, streamlines and streaklines are identical curves.

### Integration-based approaches

Integration-based approaches allow a better communication of the long term behavior induced by flow dynamics. These methods first integrate the flow data and use resulting integral objects as basis for visualization, *e.g.* streamlines. A streamline [102] is a sequence of so-called sample points. Each sample point is obtained by integrating its position as a function of the position of the previous sample point. A streamline is an integral curve that is everywhere tangent to a given vector field.

The main problem in geometric visualization is to adequately choose the seeds as starting points of the streamlines, avoiding to miss interesting areas of the field and attempting to generate a properly sparse field. In their work "Image-guided streamlines placement" [226], Turk and Banks propose an approach to accurately control the streamlines density (Fig. 2.19). They 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. He reduces the energy, and thereby improves the placement of streamlines, by changing the positions and lengths of streamlines, joining streamlines that nearly abut, and creating new streamlines to fill sufficiently large gaps. They discuss an approach to select a certain number of stream lines to be automatically and equally distributed all over the computational domain, in order to characterize in a sketch type representation the significant aspects of the field. An energy minimization process is used to generate the actual distribution of streamlines. Further approaches have been introduced to improve streamlines placement [100], attempting creating evenly-spaced streamlines.

Streamlines are a very intuitive visualization method, anyway the placements of such visualization cues is a critical issue [155]:

- 1) an even distribution of seed locations usually does not result in an even distribution of integral objects.
- 2) occlusion could raise special challenges: dense placement often results in severe cluttering within rendered images.
- 3) when using feature-based strategies, placement needs to be coupled (and aligned) with the feature extraction parts of the visualization.

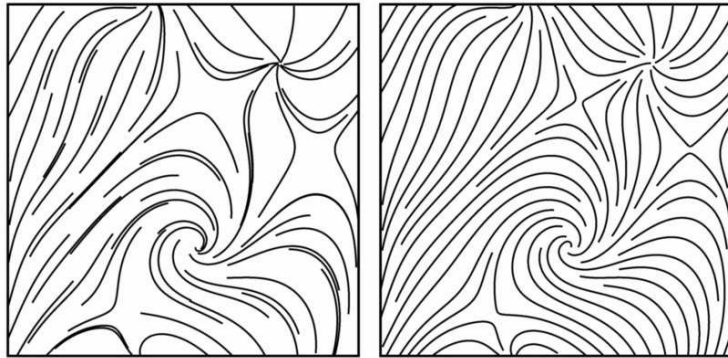


Figure 2.19: Grid-seeded streamlines (left) and optimized placement (right) (image courtesy of Greg Turk).

### Feature-based visualization

Visualization methods based on topology are an alternate approach to vector field visualization. They rely on the *extraction* of topological *features* such as critical points and separatrices (vector field topology methods), and consequently they visualize the vectorial data at a high level of *abstraction*. Especially in case of data that are complicated to read and analyze [155], techniques based on feature extraction, vector field clustering and topology extraction may result useful to simplify the visualization, separating and highlighting relevant information contained in the data, *e.g.* topological features. This results in a high level of abstraction. During the abstraction step, interesting objects and meaningful patterns need to be determined, quantified, described, and then extracted from the data set, and can be finally visualized efficiently and without the original data. Therefore, just the features of interest are visible and a huge data reduction is achieved; this makes this approach suitable for large data sets, while the features extraction requires a lot of time and is performed during the data preprocessing.

Common features of interest to extract (*e.g.* in fluid dynamics applications) are vortices, shock waves, separation and attachment lines, recirculation zones and boundary layers. Although most feature detection techniques are specific for a particular type of feature, in general the feature-based techniques [155] can be divided into three approaches: based on image processing, on topological analysis, and on physical characteristics.

### Texture-based visualization

Dense, *texture-based* visualization can also be conceptually seen as obtained from geometric visualization through a dense seeding strategy. That is, densely seeded geometric objects result in an image similar to that obtained by dense, texture-based techniques. Likewise, the path from dense, texture-based visualization to visualization using geometric objects is obtained using something such as a sparse texture for texture advection.

Texture-based techniques can provide dense spatial resolution images. They are effective, versatile, and applicable to a wide spectrum of applications. They mainly apply the directional structure of a flow field to random textures. The basic idea is to advect a noise texture along the flow by integrating the texture coordinates back in time. Traditional visualization approaches, such as vector plot, particle tracing, stream surfaces, volume rendering, and so on, often provide a rather coarse spatial resolution. This problem can be tackled by texture-based algorithms, which can provide the vector and flow visualization by high resolution output texture. These dense methods reveal to be effective, versatile, and suitable for a large spectrum of applications.

Here, a disadvantage here could be represented by loss of controllability, since the approaches heavily depend on the form of the texture.

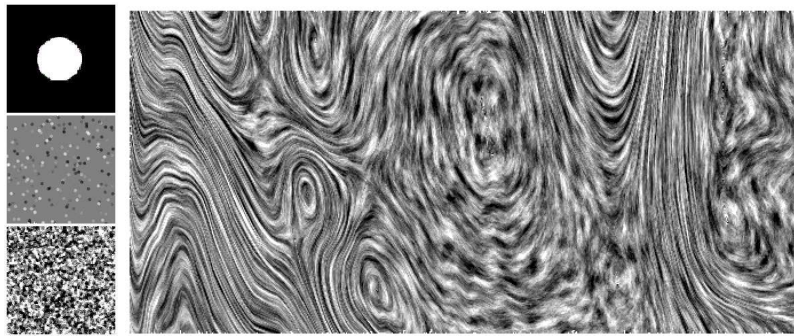


Figure 2.20: Spot noise (image courtesy of Jack van Wijk).

**Spot noise:** Spot noise by van Wijk [230] is a technique for texture synthesis, which is very useful for vector field visualization. Spot noise generates a texture by distributing a set of intensity functions, or *spots*, over the domain. Each spot represents a particle moving over a small step in time and results in a streak in the direction of the local flow from where the particle is seeded (Fig. 2.20). One limitation was the lack of velocity magnitude information in the resulting texture. Enhanced Spot noise [45] helps better visualizing highly curved vector fields, by adapting the shape of the spots to the local velocity field. Further, filtering of spots is proposed to eliminate undesired low frequency components from the spot noise texture. Spot noise creates noise-like textures by compositing many replicas of a shape, the *spot* (Fig. 2.21). The spots are then stretched, or elliptically shaped, according to a given vector field to illustrate its direction. Nevertheless, the technique heavily depends on the form of the texture itself and specifically it does not easily generalize to other forms of textures that might be better suited to a particular class of vector data.

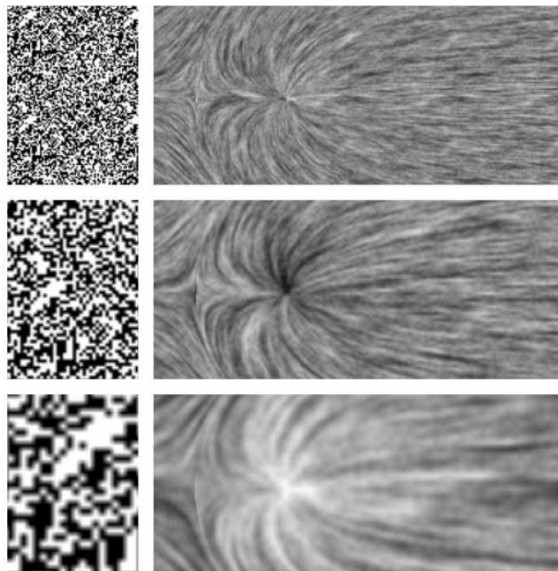


Figure 2.21: Texture-based techniques: Image based flow visualization (image courtesy of Jack van Wijk).

**DDA (Digital Differential Analyzer) convolution:** Generalizing traditional DDA line drawing algorithms [28], each vector in a field is used to define a long, narrow, DDA generated filter kernel tangential to the vector and going in the positive and negative vector direction some fixed distance<sup>6</sup>.

<sup>6</sup>The basic idea of the DDA is to draw a digital line determining the pixel y coordinate by evaluating the change in

A texture is then mapped one-to-one onto the vector field. The input texture pixels under the filter kernel are summed, normalized by the length of the filter kernel, and placed in an output pixel image for the vector position. The input texture image is white noise. Anyway, this algorithm is very sensitive to symmetry of the DDA algorithm and filter. Additionally, it is inherently inaccurate in the case of points where the local radius of curvature is smaller than the length of the DDA line, being now the straight line a bad approximation. This is also a problem of spot noise, when stretched along the vector field direction: in case the major axis of the elliptical spots exceeds the local length scale of the vector field, the spot noise will inaccurately represent it. Cabral and Leedom [35] recognize that an accurate measure of local field behavior would require a global analysis of the field. Such technique currently does not exist for arbitrary fields [35]. Another disadvantage is the loss of accuracy.

**Line Integral Convolution (LIC):** Differently than in DDA, here a local streamline is used instead of a vector, to generate the filter. LIC by Cabral and Leedom [35] is a very popular technique that has become a standard in vector field visualization (Fig. 2.22-left). Briefly, it filters an input image along local stream lines defined by an input vector field and generates an output image. The system takes as input the vector field on a Cartesian grid and a white noise of the same size. In LIC a random texture is locally smoothed or blurred along the path of the field lines (streamlines) of a stationary 2d vector field using a one-dimensional filter kernel. The procedure stretches a given image along paths that are directed by the vector field. In this approach, linear and curvilinear filtering techniques are used to locally blur textures along the vector field. Hence, convolving the filter with regular images generates motion blur through a smearing effect.

A limitation of LIC is that it only visualizes local vector field tangents, but not their direction. Using filtering, and especially periodic motion filtering, they simulate motion by use of special convolutions. Due to its simplicity, LIC has become very popular and has been broadly extended and accelerated, and further developed to a high degree of sophistication. Well-known extensions include Fast LIC [188], enhanced LIC [84], and FROLIC (*Fast Rendering of Oriented LIC*) [238].

**Reaction diffusion techniques:** Reaction diffusion techniques visualize vector fields mapping vector data onto differential equations, since the controlling differential equations are inherently vectors in nature, to come up with a vector visualization technique.

Perona and Malik [150] firstly introduced a continuous diffusion model which allows the denoising of images together with images enhancing. Using anisotropic nonlinear diffusion for flow visualization, Diwald, Preusser and Rumpf smooth an initial noisy image along streamlines, sharpening the image in the orthogonal direction [46]. The method is based on a continuous model and requires the solution of a parabolic PDE (partial differential equation) problem. It is discretized only in the final implementation step. They identify that a common problem of some popular approaches as LIC or spot noise is that when a coarser or finer scale in the visualization is needed, LIC requires a recomputation: when for instance a finer or coarser scale of the convolution pattern is required, the LIC computation has to be restarted with another adequate initial image intensity. In case of spot noise, different sized (larger) spots have to be selected and their stretching along the field has to be increased.

**Texture advection, Lagrangian-Eulerian Advection (LEA):** Jobard and Lefer [101] use a motion map data structure for animating steady state flow fields. The motion map contains both a dense representation of the flow and the information required to animate the flow. Another valid texture-based method for the visualization of unsteady fields is the one proposed by Jobard *et al.* [98]. This consists in a hybrid scheme (LEA) that combines the advantages of the Eulerian and Lagrangian frameworks. The algorithm encodes the particles into a texture that is then advected (Fig. 2.22-right). It is a kind of dense visualization, which can be applied also to time-dependent vector fields. They treat every particle equally, handling texture advection and dye advection; they blend successive frames to achieve spatial and temporal correlation. Unlike the most approaches for vector field visualization, LEA can handle also unsteady fields. A dense set of particles is stored as coordinates in a texture. Each iteration is defined by a *Lagrangian step* (backward time integration of the particles) and an *Eulerian step* (update of the image pixel colors). They achieve

---

the  $y$  position of the ideal line when  $x$  increases by 1.

temporal and spatial correlation through the blending of successive frames.

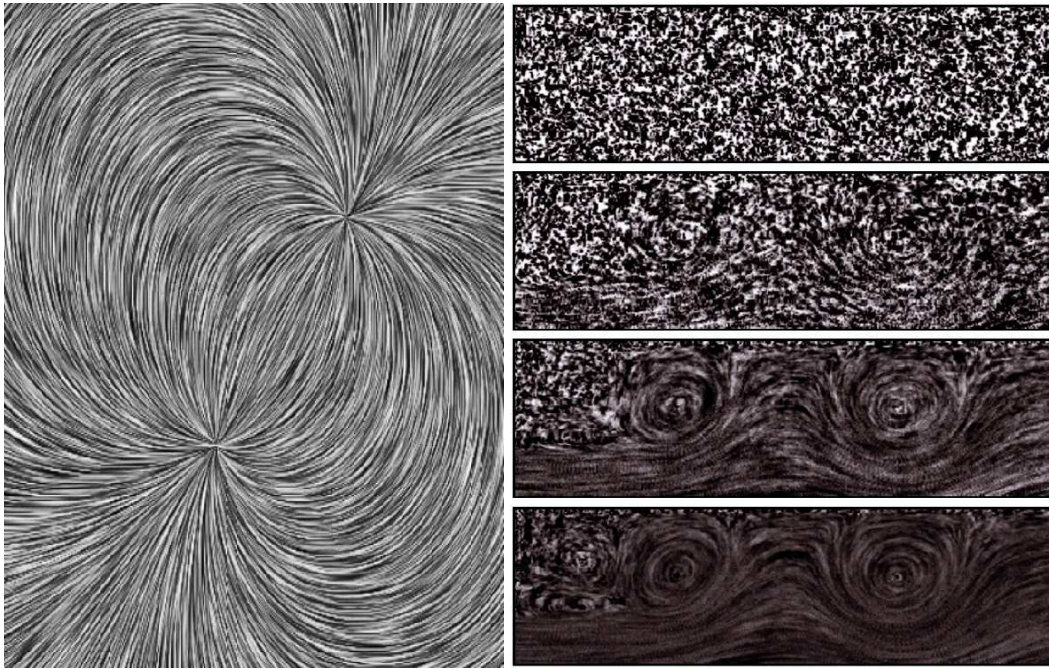


Figure 2.22: Left: LIC-based vector field. Right: frames obtained varying the opacity value of the advected noise array (image courtesy of Bruno Jobard).

### 2.3.4 Comments

Although numerous valid techniques have been proposed, vector field visualization remains a challenging subject. Direct techniques are intuitive, but may miss important features and are often inaccurate. Texture-based approaches are well suited for dense, accurate field visualization, but at the same time they are not so accurate in depicting field features and encoding parameters. Also, they are not so successful at showing, in a single, static image, 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. Cabral and Leedom [35] recognize that few techniques can image vector fields in a general manner; they are in some cases effective, but break down when operating on very dense fields and do not generalize to other applications. Line Integral Convolution (LIC) 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. In their STAR [156], Post *et al.* note that most visualization techniques, especially feature-based and geometric approaches, are generally very specific for a certain type of problem (such as vortex detection), the relation with the original raw data is indirect, and the reduction is achieved at the cost of loss of other information, which is considered non relevant for the purpose. But it is opinable to have techniques that generalize well to analysis of data, leading to condensed visual summaries. Attempting to combine the features of different techniques could be a possible solution for a more complete and general visualization method.

As demonstrated by the LIC method, texture-based approaches provide a dense visualization; textures have the advantage, as opposed to discrete direct visualization, of providing a continuous representation of a vector field. Nevertheless, texture is a visual primitive, which has been much less explored. By exploring methods for conveying vectorial data using features derived from natural texture patterns, it is possible to profoundly expand the range of possibilities in texture-based vector field representation, and to introduce the possibility of using texture type itself as a

visual variable for conveying information about the field. In this work, textures, and especially directional textures, are employed in a sample-based approach for vector field representations. In this way, textures are not textured noise to smear or advect through integration in the direction of the vector field, but serve as visual cue that is continuously transformed according to the values and attributes of the vector field. Texture images are in this way used in a statistical synthesis approach based on neighborhood comparisons to set the colors of the pixels in the resulting output image. Directional textures provide an intuitive natural way for representing vector fields at high spatial resolution.

Intrinsically, it is possible to recognize lots of intern correlation between textures and vector visualization methods. Nevertheless, the potential of textures has not been deeply investigated and used for this task. The user study by Laidlaw *et al.* [123] proved that existing methods may have problems in representing relevant vector fields attributes. Till now, the use of textures has been mostly limited to white noise and spot elements, to smear or animate. But a texture can be additionally used as a powerful visual primitive, can itself carry a great amount of information, taking advantages of its perceptual attributes to exploit numerous encoding possibilities.

In a sense, the visualization approach proposed in this thesis is a sort of hybrid method among the direct and the texture-based visualization techniques. Attempting to combine the advantages and benefits of such complementary methods, the main goal of this work is to provide a vector imaging algorithm, which is able to produce high resolution images in a straightforward way, at the same time being able to reveal vector characteristics, such as direction, orientation, magnitude, in an effective meaningful manner.

## 2.4 Summary

My work is thus placed in this context, as a sort of hybrid solution, attempting a combination and integration of concepts from the field of texture synthesis, statistics and image processing for applications in scientific visualization and in steerable texture synthesis, taking into consideration theory from cognition and perception, psychology and human vision for more rigorous validation. Several remarkable contributions to the field of computer graphics are presented in the following chapters.





## Chapter 3

# Steering Texture Synthesis for Vector Field Visualization

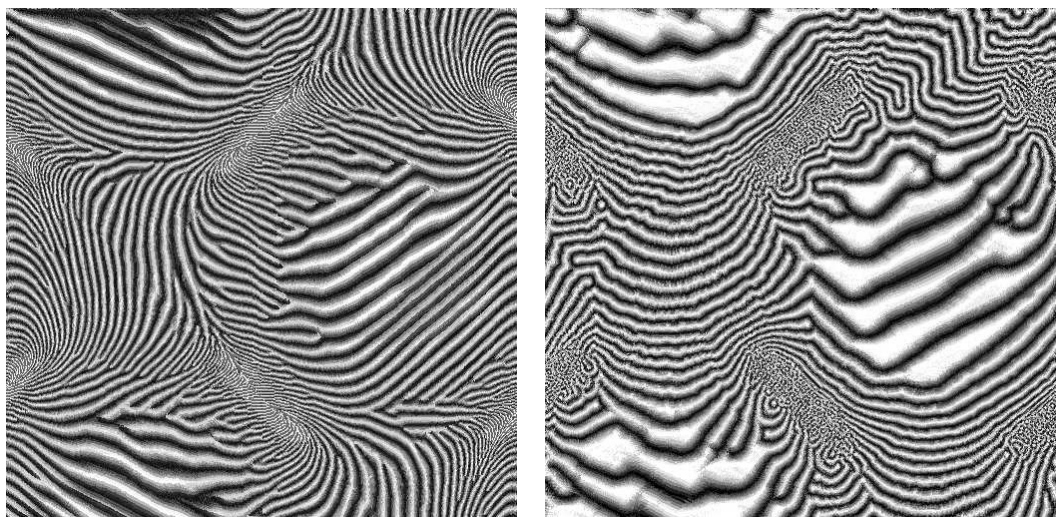


Figure 3.1: Vector field visualizations synthesized using MRF texture synthesis with a gradient example texture that is rotated and scaled according to the vector field. The two images use different sample textures, which are characterized by lines of different orientations.

This thesis focuses on methodologies and algorithms for scientific visualization, with applications also in the field of texture synthesis. In the following, I present my research done for user- and task-driven vector field visualization.

Although numerous valid methods for scientific visualization and texture synthesis exist, there is still a strong need for local and global control in the image generation process. Intuitively representing the information contained in complicated and abstract data sets is a fundamental and challenging task with uncountable applications. Interaction is often needed, and the broad variety of possible applications and scenarios also requires a flexible and general user-centered approach. In this chapter, I start illustrating the basics of a novel approach to vector field visualization, which has also resulted in some papers publications [200, 192]. The ideas and research proposed here are valid for a variety of applications and are further broadly extensible; I explain further valuable case studies in Chapter 5 and application to controlled texture synthesis in Chapter 7.

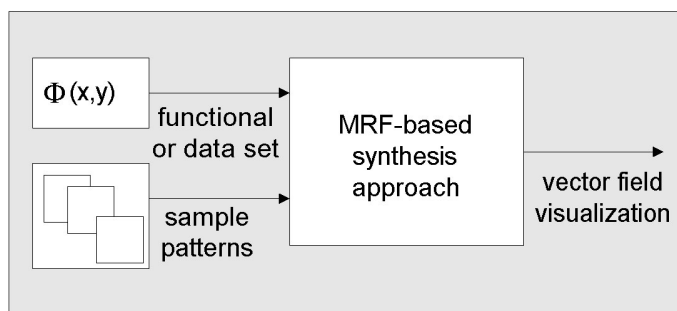


Figure 3.2: Basic visualization block scheme.

### 3.1 Methodology: visualizing vector fields using statistical theory

Vector field visualization aims in general at generating images in order to convey the information existing in the data. In this approach, I use *Markov Random Field (MRF)* texture synthesis theory to generate the visualization result, starting from a set of sample textures (Fig. 3.2). MRF-based methods allow generating images that are locally similar to a given example image. This idea is here extended for vector field visualization (Fig. 3) by identifying each vector value with a representative example image, *e.g.* a strongly directed texture that is rotated according to a 2D vector  $\Phi(x,y)$ . The visualization is synthesized pixel by pixel, where each pixel is chosen from the sample texture according to a similarity assessment conducted with respect to the vector values measured at the local pixel. The visualization locally communicates the vector information, because each pixel is chosen from a sample that is representative of the vector (Fig. 3.3). Furthermore, the resulting image is smooth, as MRF texture synthesis searches for best fitting neighborhoods. This leads to dense and continuous visualizations with the additional freedom of using arbitrary textures as representation for any vector value.

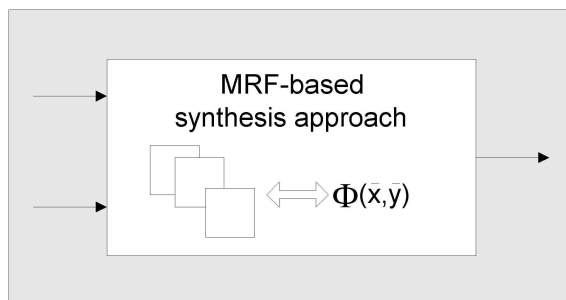


Figure 3.3: Correspondences between sample images and vector field values lead to a meaningful representation of the data set.

#### Idea and approach

Briefly, I can summarize the main novelty of the idea as follows: instead of commonly using given vectorial data sets to integrate particle positions along the vector directions and performing the required amount of operations to visualize the behavior of a vector field (*cf.* Section 2.3), I use the abstract data set to directly control the appearance of a directional example image, which is re-aligned and modified according to the local vectorial information (Fig. 3.4). This leads to a natural and intuitive visual representation of the data information, and it additionally contributes to provide a variety of degrees of freedom to better depict the vector field.

As introduced, this novel approach to visualization of vectorial data sets is based on statistical concepts and borrows theory from texture synthesis. For this reason, I briefly review below some

basic notions and fundamental concepts about *synthesis theory* and *Markov Random distributions* to subsequently explain the proposed algorithm in detail.

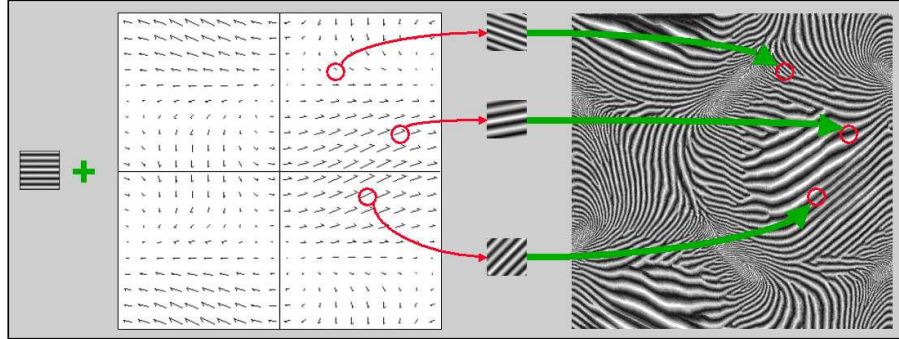


Figure 3.4: Steerable texture synthesis approach to vector field visualization.

### 3.1.1 Texture synthesis fundamentals

Since the nineties, texture analysis and synthesis approaches (including [26, 154]) began to adopt theory from statistics, and adapted it to texture generation, defining algorithms that are able to recognize, learn and reproduce pattern structures on the base of their frequential information. Also recent texture synthesis methods, as those referred in the State of the Art chapter (§ 2.2), use statistical theory to achieve the goal of generating a new texture in arbitrary resolution, starting from a little sample. The new synthesized texture should replicate the input appearance, appearing 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. Pixel-based techniques, as described in 2.2.4, are well suited for a large class of textures and patterns, due to their locality and accuracy in finding the best matching pixel at each output location.

Recent approaches model textures as Markov Random Fields (MRF) [55, 241, 10, 87] 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, and each pixel is synthesized by comparing its neighborhood to all similarly shaped neighborhoods in the sample texture (Fig. 3.5). These comparisons lead to a distance function, which is used to compute the probability to choose the best fitting 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 following equation indicates how the value of each pixel  $(x,y)$  in the output texture  $I_{out}$  depends, via a function  $\mathcal{F}$ , on the input sample  $I_{in}$  and on its neighborhood  $N_{x,y}$ :

$$I_{out}(x,y) = \mathcal{F}(I_{in}, N_{x,y}) \quad (3.1)$$

In this chapter, I propose a pixel-based algorithm; the possibility to control the synthesis of each single point in the output image is particularly important to provide a large number of additional degrees of freedom and, thus, a strong controllable framework. Extending some starting concepts that are valid for the generation of uniform textures, a novel algorithm is now designed with particular focus on flexibility and user intervention. In this schema, the vector field is visualized using theory from texture synthesis, and the following equation indicates how each pixel  $(x,y)$  in the output image  $I_{out}$  depends on a given input sample  $I_{in}|_{(x,y)}$  chosen within an input set  $\{I_{in}\}$ , on its neighborhood  $N_{x,y}$  and on the vector field value  $\Phi(x,y)$  assumed at  $(x,y)$ :

$$I_{out}(x,y) = \mathcal{F}(I_{in}|_{(x,y)}, N_{x,y}, \Phi(x,y)) \quad (3.2)$$

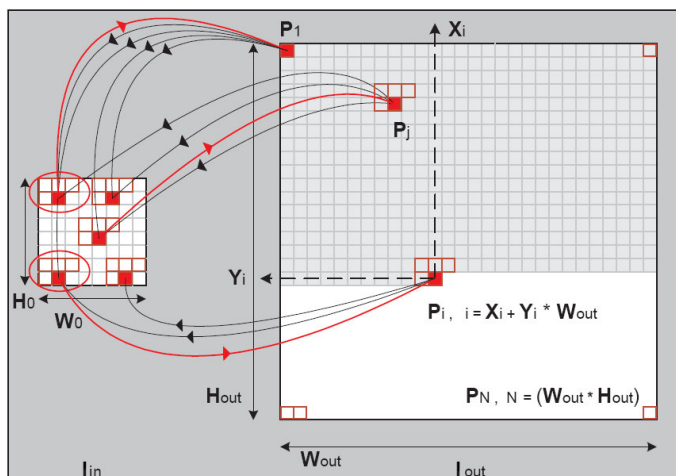


Figure 3.5: Standard pixel-based approach to texture synthesis: the pixels  $P_i$  of the output  $I_{out}$  are set checking the most probable pixels in the input  $I_{in}$ .

### 3.1.2 Markov Random Field theory

Markov Random Fields (MRF) models have been successfully introduced in many fundamental areas of image analysis and computer vision, such as image restoration and segmentation, edge detection, computer tomography, surface reconstruction, stereovision, motion analysis, or scene interpretation. In texture synthesis they define an efficient and powerful framework to specify nonlinear interactions between features of the same nature or of a different one. They provide a flexible mechanism for modelling spatial dependence. For this reason, they promise interesting results also in the research field discussed here: I investigate their application to vector field visualization and present in the following the developed approach, together with results and discussion.

Random field models analyze spatial variations in two dimensions. *Global random field models* treat the entire image as a realization of a random field, whereas *local random field models* assume relationships of intensities in small neighborhoods. A widely used class of local random field models type are *Markov random field models*. The MRF model for textures assumes that the texture field is stochastic and stationary, and satisfies a conditional independence assumption.

Markov Random Fields have been proven to cover the widest variety of usable texture types [241], that is, they are a good approximation to model a broad range of textures, they are general and produce good results. Algorithms model textures by Markov Random Fields, or in a different mathematical form, Gibbs Sampling, and generate textures by probability sampling [55]. MRF models are used to describe the probability distribution governing the intensity values of pixels in a specific neighborhood<sup>1</sup> also known as a *clique*<sup>2</sup>. The technique uses texture primitives to guide the synthesis process. The use of multiple primitives in my approach is necessary to exhibit different field features. The pixels of a randomly initialized image are iteratively updated until the representation of the vector field emerges.

In general, the value of each pixel  $p$  in a texture should depend on the pixels of its neighborhood  $N_p$ :

$$p(x,y) = \mathcal{F}(N_p(x,y)) \quad (3.3)$$

<sup>1</sup>Figure 3.6 shows how textures differ from images. (a) is a general image while (b) is a texture. Movable windows with two different positions are drawn as black squares in (a) and (b), with the corresponding contents shown below. Different regions of a texture are always perceived to be similar (b1, b2), which is not the case for a general image (a1, a2). In addition, each pixel in (b) is only related to a small set of neighboring pixels. These two characteristics are called stationarity and locality, respectively.

<sup>2</sup>A *clique* is a collection of *sites*, or *places*, which are neighboring elements.

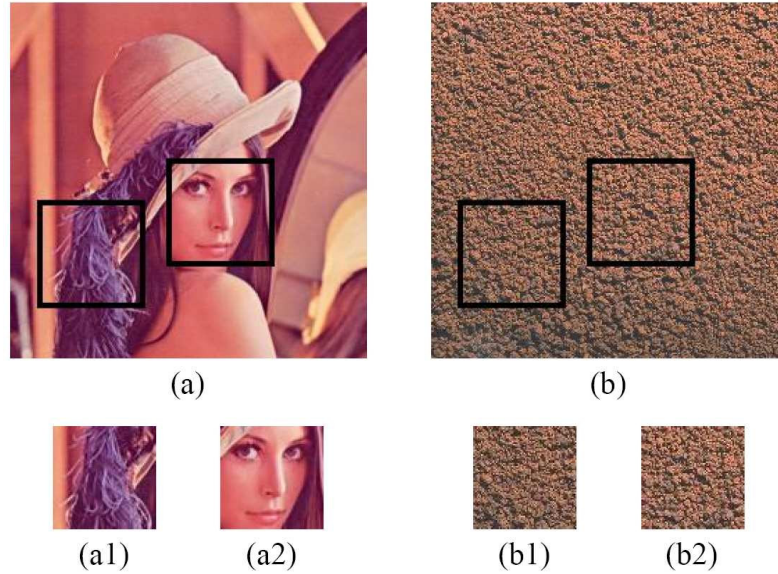


Figure 3.6: Locality and stationarity properties in textures (image courtesy of Marc Levoy).

The method of [55] assumes a MGRF model of textures so that a pixel  $p \in I$  only depends on the pixels in a local neighborhood  $N_p \in I$ . The distance  $d(N_q, N_p)$  between neighborhoods  $N_p$  and  $N_q$ , e.g. the *sum of square differences (SSD)*, provides a metric of pixel similarity between  $p$  and  $q$ . To synthesize a pixel  $p$ , the algorithm seeks for a pixel  $q$  from the training texture that minimizes the distance between  $N_p$  and  $N_q$ , and then uses the value of pixel  $q$  for pixel  $p$ :

$$q = \underset{j \in I}{\operatorname{argmin}} d(N_q, N_p) \quad (3.4)$$

An approximation to the conditional probability distribution

$$P(p|N_p) \quad (3.5)$$

where the value of the pixel  $p$  is conditioned to its neighborhood  $N_p$ , needs to be constructed, and then it is possible to sample from it. Modelling a texture using MRF, leads to the fact that the value of  $p$  is independent from the rest of the input sample  $I$ , given its neighborhood  $N_p$ . In this method, the Markov property of a texture is preserved in a non-parametric way; it ranks samples of observed pixel neighborhoods and selects the closest one based on a similarity metric. Essentially, the sampling attempts to maintain local integrity of texture. Every pixel in the image can be expressed by a color value that is dependent on the surrounding pixels, lagged both in space and in time (see also § 5.1.3).

In the next sub-chapters, I introduce the requirements that the approach should fulfill and I explain the algorithm, starting illustrating the conditions to satisfy, and the specifications for the input parameters and settings.

## 3.2 Sample-based visualization

In the sense of texture synthesis theory, the proposed approach to vector field visualization is thus *sample-based*. The algorithm learns and reproduces the characterizing structure of the pattern of the input image, transferring it from the little example texture to the larger output (Fig. 3.7). In this way, personalization of the visualization process is offered; this versatility is directly connected to the freedom of choosing the starting seed patterns. Input images may be chosen from a pre-defined

set or can be user-specified. They may be *ad hoc* designed to better reflect and transmit given characteristics and special features to the output image, allowing for instance both photorealistic and non-photorealistic visualization. This in general permits conditioning the illustration style of the resulting visualization and therefore offers a flexible way to control the output appearance.

In this section, details are provided to appropriately specify the samples that the visualization algorithm takes as input. The sample set is then organized in a matrix of seed samples (§ 3.2.3).

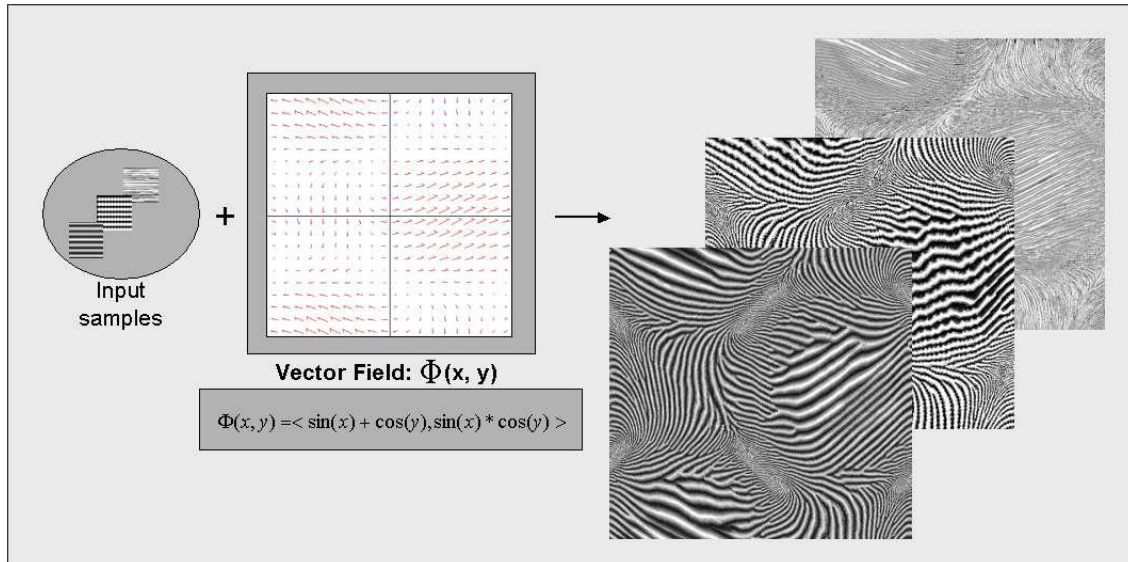


Figure 3.7: Different input appearances lead to different appearances in the resulting vector field.

### 3.2.1 Conveying vectorial information through anisotropic patterns

A key point in the presented approach is the variation of the input sample in relation to the vectorial data set that one wants to visualize. When visualizing a vector field, particular relevance needs to be conferred to the change of curvature and direction of the lines of flow that characterize the vector field. For this reason, special attention needs to be paid to the specification of the samples. Samples that exhibit a main direction are best suited to this visualization approach, since they can be well adapted to follow the field lines.

#### Anisotropic patterns

Anisotropic samples that present a major direction, and smoothly vary along the other one, are best suited for use in this visualization approach. They result in fact to be well adaptable to rotations and changes of curvature that often occur in a vectorial visualization. Hence, using directional samples guaranties an accentuated directional displacement of the information in the output, and thus the vectorial information is easier to perceive. A suitable set of possible samples is shown in Figure 3.8.

#### Anisotropic oriented patterns

Besides just using directional patterns, it can be useful to generate directional oriented example textures; this contributes to convey and enhance both vectorial information of direction and orientation (an example is the 8<sup>th</sup> pattern in Fig. 3.8). Taking inspiration from line integral convolution (LIC), and especially OLIC and FROLIC (*oriented-* and *fast rendering of oriented line integral*

*convolution*), a ramp-like convolution kernel can be used to produce streamlets with varying intensity along the trace, using asymmetric convolution kernels (*e.g.*: the 9<sup>th</sup> pattern in Fig. 3.8). Such a filter can be applied to white noise, by integrating it along a constant direction, to generate a sparse example texture, where the varying intensity of the streamlets makes it possible to recognize orientation in flow fields. Alternatively, simple image manipulation and 2d graphical programs can be used to produce similar input examples.

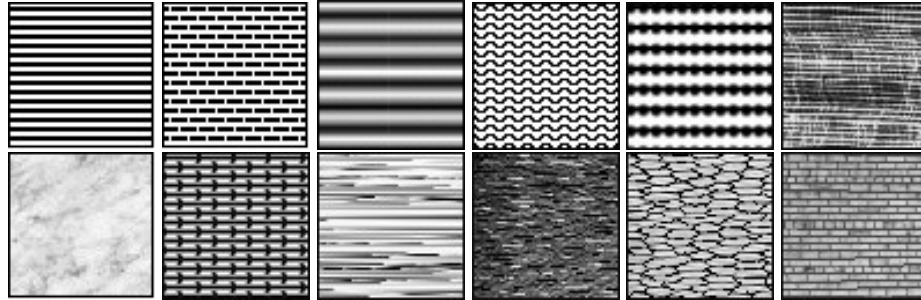


Figure 3.8: Possible set of grey-scale anisotropic patterns used as input seeds.

### Anisotropic patterns and rotation-invariant features

Samples that present rotation-invariant characteristics, like in the 5<sup>th</sup> pattern of Fig. 3.8, are less sensible to edge- and step-wise artifacts, while too naïvely designed samples (such as the 1<sup>st</sup> pattern in Fig. 3.8) could be susceptible to aliasing (Fig. 3.29). They can limit or avoid aliasing during rotation, and in this way more continuous visualization is warranted. For the same reason, good quality is achieved also using samples that are characterized by a gradual and smooth change of color in the direction perpendicular to the major one (*e.g.* the 3<sup>rd</sup> pattern in Fig. 3.8).

### Quasi-anisotropic patterns plus anisotropy and orientation encoding

When additionally taking into account considerations on information mapping and relative encoding options (more details in Chapter 6), also almost-isotropic or isotropic input samples can be used for the proposed approach. A possible example is given by an isotropic pattern describing isotropic granular material in a vectorial data set representation; its appearance can be augmented via color coding to create the impression of movement along the vector field direction. A further example is given by a hybrid mapping of velocity magnitude to directionality. In this case directional patterns highlight particles of the flow field with high velocity, while isotropic patterns represent areas with low or zero velocity. In this way, different velocities can be simulated using different patterns.

### Shape of the example textures

Regarding the shape of the input samples, they are for simplicity squared in standard cases. In the proposed approach I also mostly use squared input samples; however, no restriction exists, and rectangular as well as differently shaped samples can be used without any problem.

### 3.2.2 Texture databases

Several texture databases are available in the Computer Graphics and especially in the Image Processing literature. They are mostly used to classify textures for algorithm testing, image segmentation and retrieval. These databases have been tested for the presented approach, and especially for the controlled texture synthesis described in Chapter 7.

### Brodatz textures

*Brodatz textures* [29, 30] are a well known set of natural textures, which have broadly been used for texture synthesis and image processing. These samples are in gray scale and mainly represent natural materials. The Brodatz collection comprises of a relative large number of texture classes; the popular album [29] consists of homogeneous categories of naturally occurring textures. Although it has become a standard for evaluating texture algorithms, the Brodatz set is very limited in variety; it is all monochrome, and consists mostly of homogeneous patterns photographed under studio lighting at an angle parallel to the film plane, for this reason, it may be better integrated with further databases for a more complete evaluation.

### VisTex

The *VisTex collection* (available at [3]) has been assembled and maintained at the Vision and Modeling Group at the MIT Media Lab. These data are freely distributed; VisTex has been built using textures in large sets of natural color scenes, taken under arbitrary lighting and perspective, categorized into mutually exclusive groups. For instance, their lighting conditions include daylight, artificial-florescent and artificial-incandescent. A sub set is shown in Figure 3.9. The VisTex set consists of heterogeneous categories of texture images, that is, each class may have more than one type of texture. For example, the flower category may have flower images at three different resolutions, thus making the set more difficult to classify, but also offering more flexibility.



Figure 3.9: A sub-set of the VisTex texture database.

### MeasTex

*MeasTex* (the MEASurement of TEXture classification algorithms) [1] is an image database and a quantitative measurement framework for image texture analysis algorithms. The textures are available for the classes: asphalt, concrete, grass and rock. They are mostly isotropic and stochastic (Figure 3.10), for this reason they are not of particular interest for the approach proposed here. A number of texture sets (artificial and natural textures) in MeasTex have been compiled by other texture databases such as the Brodatz texture database and the VisTex database.

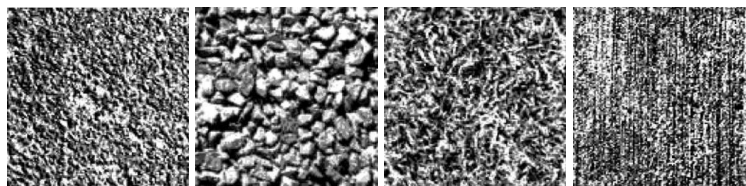


Figure 3.10: A sub-set of the MeasTex texture database.



### Deriving or designing a specific sample

Further texture databases exist; several issues, for instance the issue of homogeneity of an image, can be examined by defining a database containing a given amount of perceptually similar, most uniform, images.

However, for the scope of this thesis, no restriction to a specific database or texture categorization needs to be done. When choosing the input samples, the sole considerations to guarantee good results are *de facto* the directional specifications explained above. Therefore, I prevalently tested anisotropic or quasi-anisotropic samples and suggest the use of such patterns, as they can in general produce perceptually intuitive results. The samples can thus be, without distinction, either hand-designed, derived from photographs, or from existing databases. In addition, texture samples can be typically created by scanning in existing artworks, by using a 2d paint program, or they can be synthesized from procedural image models.

### 3.2.3 Input matrix seed

One of the novelties of the presented approach is that it makes use of a large set of input samples instead of using just a single input. I adapt the MRF-based synthesis to accept a space of input seeds and adequately modify them with respect to the vector field to visualize. Some previous 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 [157, 225, 242] 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.

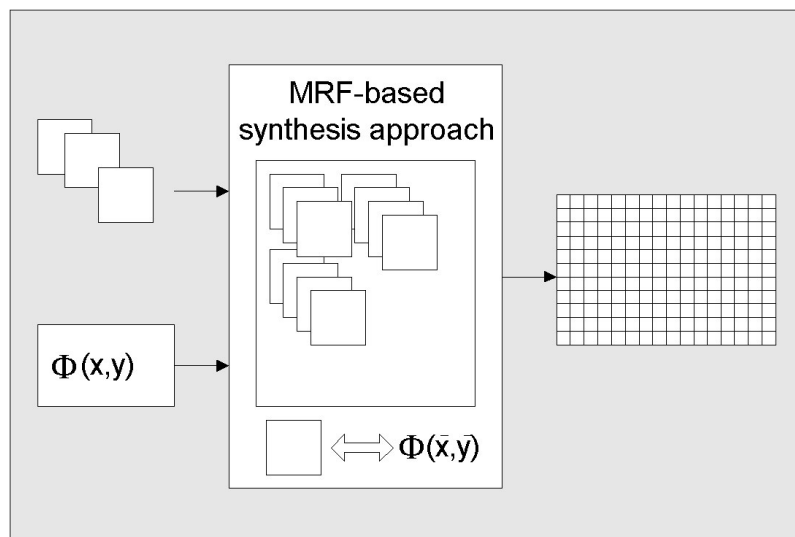


Figure 3.11: Correspondences (potentially bijective) between samples and vector field values.

This set of samples constitutes the array - or better the *matrix* - of input samples that are used as seed for the visualization (Fig. 3.11). The variety of the samples that build such structure derives from the generation of field-dependent rotated and resized versions of the original chosen sample. Auxiliary filter banks can be then applied to generate variation in the samples and to transform them progressively (see also § 6.5). Additionally, also uncorrelated samples can compose the sample set for applications of texture mixing and metamorphosis (see Chapter 7).

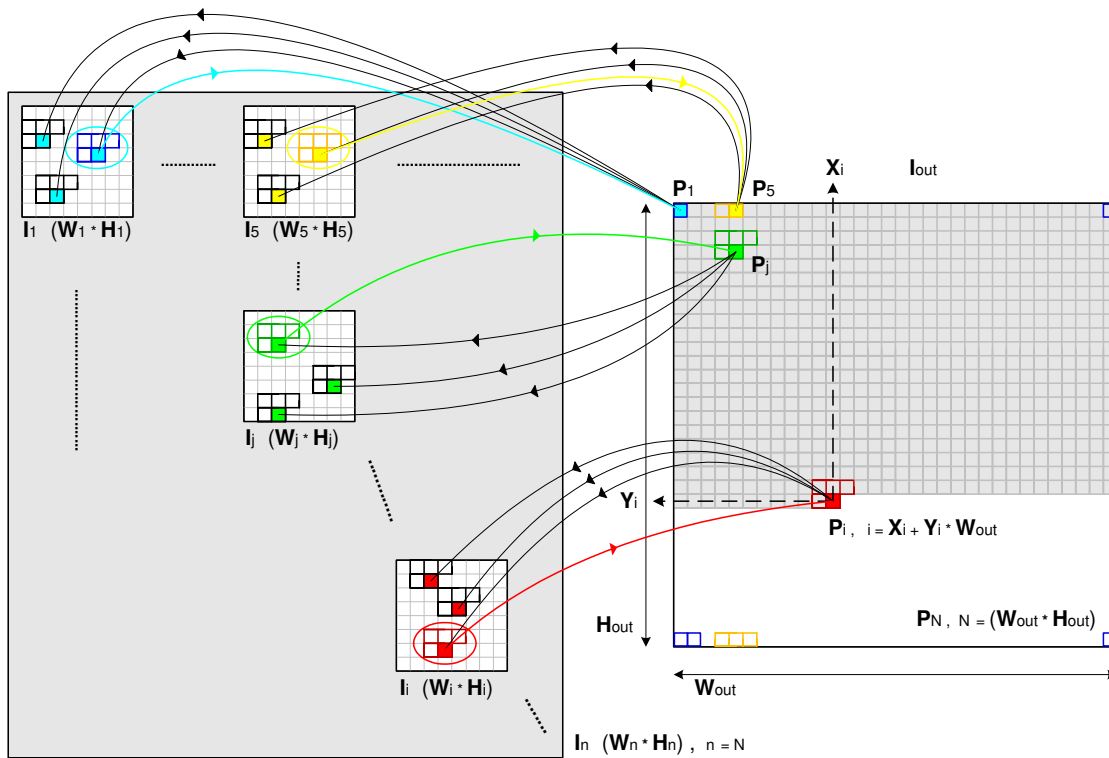


Figure 3.12: Synthesis process (cf. Fig. 3.5) using the matrix of samples seeds.

### 3.3 Synthesis approach

Once the matrix of samples has been defined, either through selection of samples or *via* the use of filter banks, the appearance of the resulting image is already in part and implicitly conditioned. In a later synthesis phase though, it is still possible to decide which visual representations will map the vector field values and special features of interest (Chapter 6).

The next step is now to synthesize the vector field visualization using the per-pixel procedure. In this section, the role of the control vector field is described and details are provided about the MRF-based modelling of the texture examples, as well as about the deterministic search of best fitting pixels.

#### One-pass algorithm: single resolution

For simplicity and clarity of explanation, I begin introducing the idea and the implementation of the algorithm for the case of two-dimensional vector field visualization. The explanations are here related to the *single resolution* version of the algorithm. Later on, in Appendix A, I also explain the extension to a multi-resolution approach (Fig. 3.13) and the relative implementation. In such scheme, the one-pass algorithm is repeated in a *multi-resolution* approach using *Image Pyramids*.

As briefly introduced above, for each output pixel a neighborhood-based search for the best matching pixel within the input sample under consideration has to be run. The most time consuming process during the synthesis is indeed 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 [31]. Another way to synthesize large textures in a faster manner is to copy blocks rather than pixels in each step of the algorithm [54]. Recent methods can use texture theory to synthesize outputs in an interactive way [254]. They use a computationally expensive pre-processing step where all neighborhoods are pre-tested; then the synthesis step can efficiently synthesize the output texture.

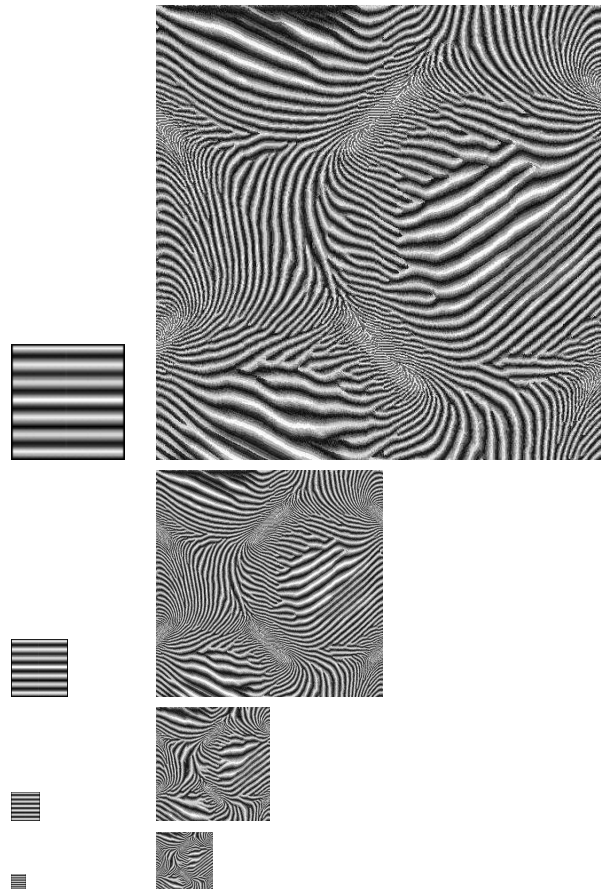


Figure 3.13: Pyramid levels for the multiresolution synthesis process of the image in Figure 3.

Unluckily this computationally expensive pre-processing step cannot be applied to the presented approach, since, differently from Zelinka and Garland approach, where they only need to run it once, here constantly different samples are used, which may be even modified and processed in run-time.

### 3.3.1 Control vector field

Together with the specification of the *sample seeds* (§ 3.2), the vectorial data set one wants to visualize is the primary input entity for the synthesis procedure. The *vector field* plays here the role of a *deformation* or *control* field over the chosen sample pattern. I call it a deformation field because, in a sense, and especially if it is seen as extension of the standard approach to static texture synthesis, a given directional anisotropic pattern - taken as input - will be curved, deformed and aligned along the new directions specified by the vectorial data set.

The information contained in the vector field should be visualized in a perceptually intuitive way (*cf.* § 2.1). To achieve this, this approach lets the data set directly condition the value of each output pixel in an easy way. The algorithm controls the appearance of the starting original sample, modifying it accordingly to the values that the vector field assumes at each output location. This means that a given point at a general location  $(x, y)$  in the output will be chosen, using MRF-based statistical theory, from a sample, or a modified version of the original sample, which uniquely communicates the values of the vector field at that point. A detailed explanation of the procedure is given below in the implementation section (§ 3.4), where the algorithm is described step by step and the used variables and methods are listed and illustrated in relative tables.

### 3.3.2 Field-driven sample alignment and transformation

The control vector field - input data set - is therefore responsible of various *transformations* over the sample, including re-alignment, resizing, change of color. The essential field attributes and the values assumed by the variables need to be adequately mapped using visual representations, in order to effectively communicate the behavior of the field. I illustrate this in the following and remand to § 2.1 and Chapter 6 for some theory on perceptually motivated correspondence among texture visual dimensions and field attributes, and for basic notions of filtering and information mapping, plus further examples and results.

Let interpret a static standard texture synthesis process as the synthesis of a given input sample along a unitary uniform vector field with zero phase. This means that we can now generate any other texture-based vector field, synthesizing a texture along a given vector field, and this just varying the control action using an arbitrary vector field. The presented visualization approach is achieved in this sense extending and constraining texture synthesis. In this way the vector field will influence the starting sample to follow new directions and it will deform or distort it according to non-unitary magnitude values (Fig. 3.14). Assuming to be then interested in visualizing extra field attributes, it is straightforward to take into consideration further information, such as an additional third dimension in space or time, further field entities as divergence or curl, or special features as critical points, separatrices and regions or points of interest. More details about field features extraction, relative information encoding and more sophisticated mapping options are given in the following related chapters of this thesis (Chapters 5, 6).

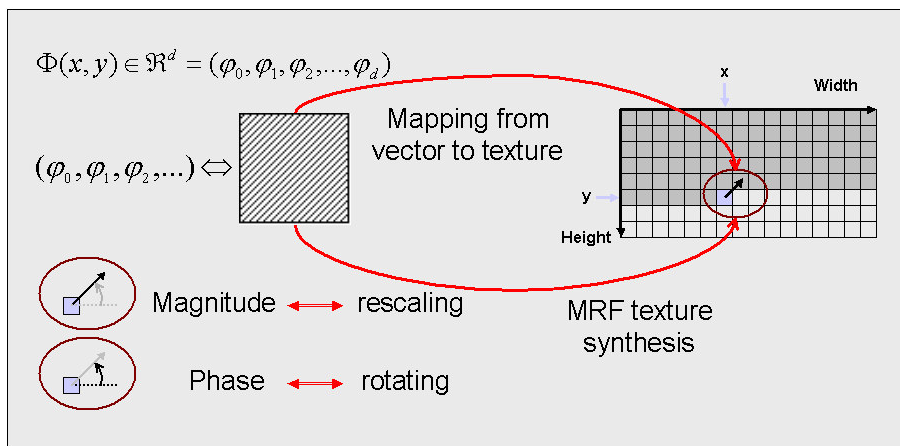


Figure 3.14: Correspondence between samples and vector values: transforming a sample through scaling and rotating operators.

#### Rotating the texture sample

Let consider a two-dimensional vector field and let start just focusing on the representation of the *vector phase* as single field feature of interest.

One mapping solution, may be the most uncomplicated and intuitive one, is to start with a directional input sample and successively *rotate* it by the phases defined by the angles that the vector field builds with the  $x$ -axis of the plane coordinate system (analytical details in 3.4). This leads to the generation of a set of sample instances, which accordingly communicate the different phases of the field at the various output locations (Fig. 3.15). This rotating process can be done on the fly during the synthesis, but it is obviously more efficient to pre-rotate the sample in a pre-processing stage and then use a look-up table to choose, for each output angle value, the correspondent rotated version of the sample. This improves the performances, being the texture synthesis process the most time consuming part of the algorithm.

The rotating process produces *diamond*-structured images; as a portion of the previous window size remains empty, a border area of the resulting images needs to be cut away (Fig. 3.16). In general (using simple Pythagoras theorem), one can use a squared image with side of length  $r' = r\sqrt{2}/2$ , where  $r'$  is the new side length and  $r$  the old one. Anyway, for simplicity, one uses  $r' = r/2$  side, which corresponds to a sample which is the half in size of the original one; for this reason is opportune to chose starting sample images which lead to sufficient resolution after transformation.

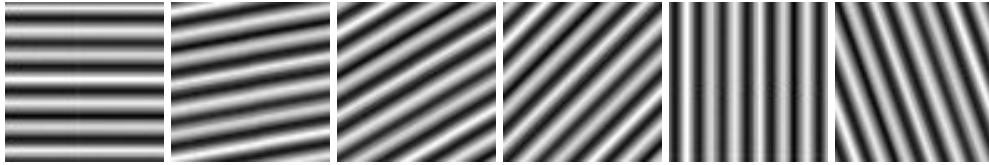


Figure 3.15: Rotated sample input images.

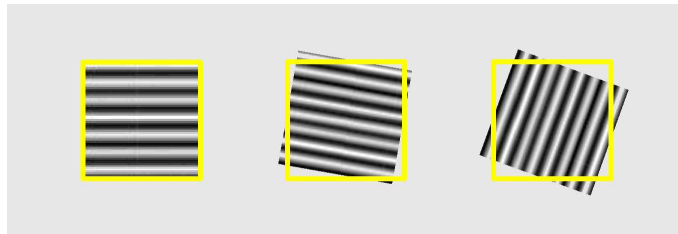


Figure 3.16: The rotating operator over the sample, generates *diamond* structures, whose border windows need to be cut away.

### Re-scaling the texture sample

The second most common vector field feature, which is of interest to visualize, is the *vector magnitude*. Depending on the particular physical meaning of the vector field, it may represent a velocity scalar field, a signal amplitude, and so on. Again, this quantity is calculated at all the output locations of the vector field, and then used to accordingly *resize* the input sample (Fig. 3.17). In this way, the sample resolution will be *magnified* if at that location the field amplitude is larger than one, it will *scaled down* if the magnitude is smaller than one, while the resolution stays unvaried if the amplitude value is unitary.

When using the resizing operator to encode information of magnitude, one needs to have a sufficient large input sample as starting input. In fact, when scaling down the sample, a smaller sample results from this operation and is now available for the synthesis process, so that it needs to contain enough information in order to guarantee the proper functioning of the algorithm. On the contrary, when enlarging the starting sample by a given magnitude factor, a border portion of the resulting sample may be cut away, of course under the condition that the magnified structure of the pattern still stays recognizable and the cut sample still contains enough information after the border cutting (Fig. 3.18).

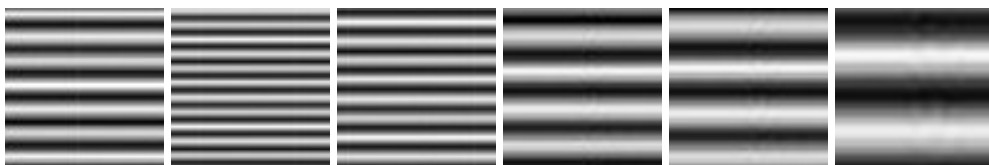


Figure 3.17: Scaled sample input images.

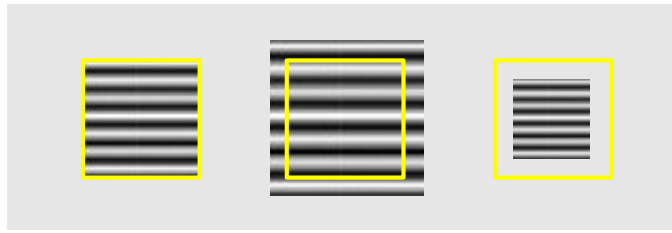


Figure 3.18: The resizing operator over the sample image (left) re-scales it by a factor  $A$ , generating sample versions in different resolution (left:  $A = 1$ , center:  $A > 1$ , right:  $A < 1$ ).

### Arbitrarily transforming the texture sample

The correspondence of the rotation and resizing operators with, respectively, the field values of *phase* and *magnitude* is just one of the possible mapping options. Several further choices are possible and especially combinations of them are available.

In a more general case, and especially in presence of a more numerous set of field parameters to represent, it is possible to map every feature of interest of the vector field using *ad hoc* specified *transfer functions*. Consequently, not exclusively rotated and scaled variants of the original image may be produced, but also filtered - in the sense of image processing operators - versions of it (Fig. 3.19, 3.20). Color, brightening, embossing and other filtering operators may also, however, be applied in a post-processing stage of the synthesis. This, besides maintaining the same performances, also has the advantage of offering the user interactivity and adaptivity in the information mapping stage. More details and relative illustrations of the system block scheme are given in Chapter 6, 7. In conclusion, any reasonable arbitrary mapping from vector values to example images can be defined, leading to a variety of possible effects. The vector values serve as argument for the chosen filtering operator, which transforms (*transfer function*) the sample by that factor.

### Quantizing the encoding of the information

In any case, regardless to the chosen information encoding, an appropriate mapping should establish correspondences between vector values and sample images. If the vector field presents very few variation in its domain, a one-to-one correspondence between each different vector value and an example image best represents the peculiar field characteristics, allowing to discriminate the different values. Otherwise, in case the field features are easily recognizable, a *quantization* of the filtering factor is possible. In practice, the original sample is pre-rotated, pre-scaled, and in general pre-transformed, in a quantized number of orientations, resolutions, and in general new transformed versions. In this way a reduction of the necessary sample transformations is achieved, and hence a reduction of the computation time.

The so defined range of field values (angles of phase, magnitudes, *etc.*) should be sufficient to adequately characterize the vector field. For each field value, the algorithm chooses the transformed sample version that most closely resembles the values locally specified by the vector field. This range in turn determines the amount of sample images necessary for a proper representation of the carried information. Obviously, different quantization can be performed depending on the variance of the considered field feature: the phase of the field may be represented with a one-to-one correspondence between phase value and rotated sample in order to enhance and precisely visualize the direction of the field, while the magnitude may be represented just using a sub-set of the assumed magnitude values if this information is of secondary importance or if this field attribute is simply recognizable even using quantization.

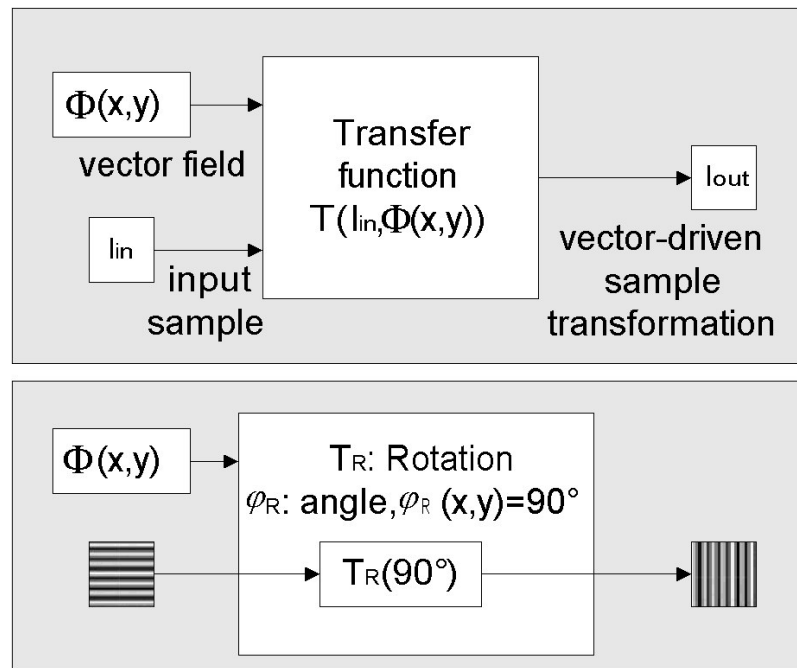


Figure 3.19: Transforming the input sample  $I_{in}$  to  $I_{out}$  using a field-driven transfer function  $T(I_{in}, \Phi(x,y))$  (top), and example of rotation by 90 degrees as particular operator  $T_R$ , where  $\varphi_R(x,y)$  represents the field feature of orientation, or angle of rotation, at a given output position  $(x,y)$  (bottom).

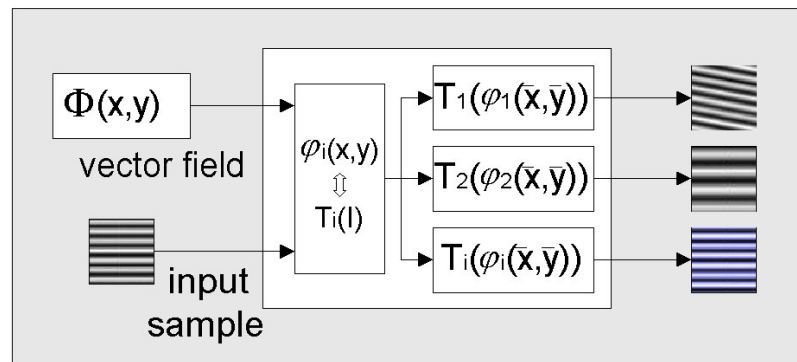


Figure 3.20: Generalization of Figure 3.19 to generic transfer functions. This general scheme illustrates a set of transformation operators (rotation, up-scaling, re-coloring) over the input sample image  $I_{in}$ . Correspondences are set between transfer functions  $T_i(I)$  and the field variables  $\varphi_i(x,y)$  taken as parameters.

### 3.3.3 Building causal neighborhood models

Let now consider the central element of the synthesis algorithm: the *neighborhood model*. As explained in § 3.1.2, MRF-based synthesis models a texture as a Markov Random Field, assigning properties of *stationary* and *locality* to the represented texture (Fig. 3.6). This means that each pixel belonging to the texture can be described by a set of surrounding pixels. The neighborhood model is thus the instrument to specify how neighboring pixels contribute to the estimation of the output pixel values, which is the core step of the synthesis procedure. In fact, the statistical approach consists on predicting each output pixel color value on the base of its neighboring pixels.

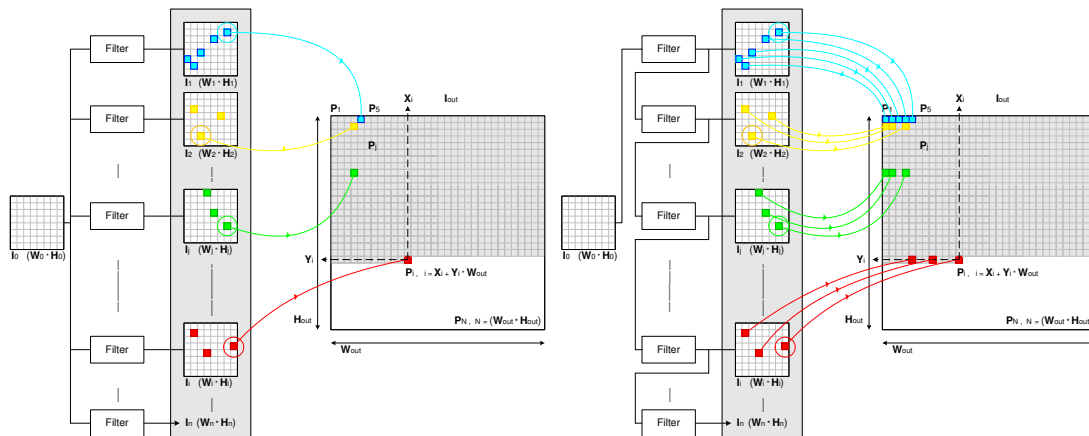


Figure 3.21: Two possible filter bank schemes for independent and iterative sample transformation.

Consequently, for each point in the output image, an appropriate neighborhood region is defined. The pixels that belong to this region contribute with their color values and with their relative distances from the central pixel to estimate and measure the local structure of the image.

### Causal neighborhoods

A standard and broadly accepted way to proceed for the generation of an output image is to synthesize it in *scan line order*; the causal property of the neighborhood accelerates the convergence speed of the algorithm. Consequently, the regularly used neighborhoods are *L-shaped*, that is, the causal portion of a squared window around the central pixel is considered (Fig. 3.22). They are *causal neighborhoods*, since they are defined through already known pixels; the neighborhood will only include pixels that have already been generated during the raster scan. This ensures that every output pixel is updated on the base of previously generated pixels and not from random noise. Only the first few pixels of the output image use white noise as their neighborhoods in the first iteration, while subsequently all pixels will use neighborhoods that have been visited in an earlier pass. As in causal systems, those known pixels represent the observed information that is used to determine the following pixels. This is due to the fact that, while proceeding in raster order, each pixel at location  $(x,y)$  can take advantage of the *a priori* already known values of the pixels located on its left and above itself (Fig. 3.22). The L-shape of the neighborhoods also leads to the fact that, when using symmetric neighborhoods centered on the current pixel, the size  $n$  of such models is always an *odd-defined size*. Starting from the simplest case, one can use three-sized neighborhoods, five-sized neighborhoods and so on. A general  $n$ -sized neighborhood contains then  $(n * n)$  pixels, and its L-version  $(n * n - 1) / 2$  pixels.

In case of *multiresolution* - or multi-pass - approach (see § A.2.1 for the relative explanation and illustrations), the neighborhoods from previously synthesized levels will be *squared* and fully populated instead of L-shaped.

### Setting the neighborhood size

The neighborhood *size*, and in turn the amount of the pixels contained in it, strongly depends on the structure complexity of the given input sample, *i.e.* the pattern resolution plays a fundamental role for the definition of a proper neighborhood size. The size of the neighborhood needs to be adequately specified, since using a  $n$ -sized neighborhood enables to conserve the spatial coherency of texture elements that are up to  $n$  pixels large. The size of the neighborhood is then responsible for the algorithm performance, as it directly determines the number of calculations needed for the choice of the output pixels.



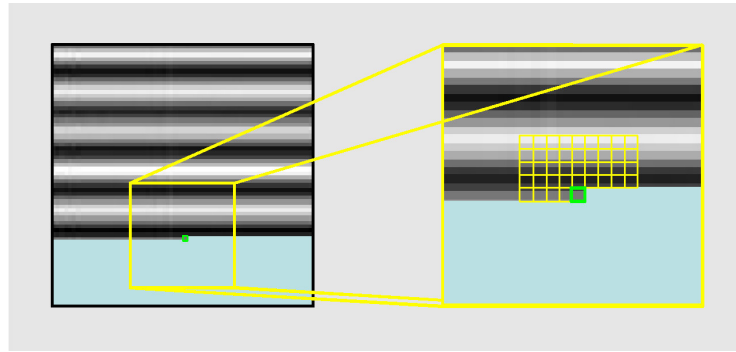


Figure 3.22: Causal or L-shaped neighborhood structure.

In the figure below (Fig. 3.23) I show three simple possible input samples for use in vector field visualization. From left to right they present increasing structure size and thus require larger neighborhoods in order to allow the algorithm procedure to learn the sample statistics and reproduce them in a proper way. Organizing different patterns in a *texture space* (§ 6.4.1) also allows to choose a proper neighborhood size, corresponding to the different pattern structures.



Figure 3.23: Samples characterized by structures with different complexity require different neighborhood sizes.

### Edge and corner treatment: toroidal neighborhoods

When considering pixels that are near to the edges of the image, or directly lie on the margins of the image, their relative neighborhoods result to be cut and incomplete. Consequently, such neighborhoods would be not symmetric around the central pixel anymore. A possible solution to handle them, as already suggested in [83] is to build those neighborhoods toroidally, and thus considering pixels from the corresponding opposite side of the image where pixels are missing, in order to complete the neighborhood. This corresponds to define a circular convolution:

$$I(x, y) = I(x \bmod N, y \bmod N) \quad (3.6)$$

where  $I(x, y)$  is an image with size  $N \times N$ . This works well especially for textures that tile seamlessly (Fig. 3.24).

Alternatively, a reasonable border handler is to pad the image with a reflected or mirrored copy of itself (except for obliquely oriented textures). Otherwise, if the sample is large enough or contains enough information to communicate its intrinsic statistical distribution and apparent structure, the synthesis can be performed disregarding the pixels on the boundaries, hence using a sub-image obtained cutting away the external boarder area (a squared or rectangular frame) of the sample.

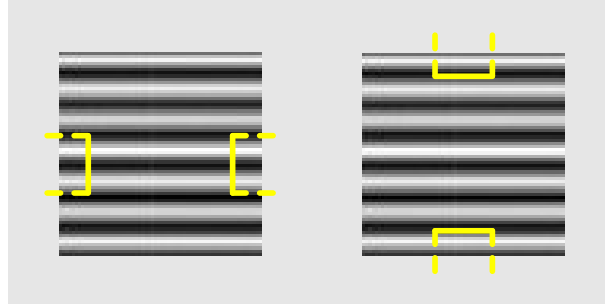


Figure 3.24: Building toroidal neighborhoods (right-left and up-down).

### 3.3.4 Calculating distances and measuring similarity probabilities

The example images in the matrix serve as seed in that pixels are selected and copied from the input to the output image. This is done after *comparing* the neighborhood of the current output pixel with similarly-shaped neighborhoods within the input sample. This consists in a measure of *distance* between pixel values, which, in practice, corresponds to a degree of *similarity* with the output region under consideration. A *similarity function*, commonly based on the *L2-norm*, is computed to evaluate the relative distances. The comparison leads to the selection of the most similar neighborhood, in term of minimum distance; its central pixel is then chosen for the actual output position. This pixel is thus the most probable to fit at the current location. For each step and output location, a best matching pixel from the input sample is chosen and its color is set to be the value in the output.

#### Similarity metrics

The *similarity* between different neighborhoods - arrays of neighboring pixels - can be calculated using a *distance function*. This calculation can be done using different metrics or distance functions. A simple solution is to use the *L2 norm*. The L2 norm is a standard to measure similarity in images, due to its simplicity and efficiency. It is a compromise between computational efficiency and accuracy.

The L2 norm, or *Euclidean norm*, of an  $n$ -dimensional vector  $\mathbf{d}$  is simply given by the square root of the sum of the elements squared:

$$\mathbf{d} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix} \quad \|\mathbf{d}\| = \left( \sum_i |d_i|^2 \right)^{1/2} \quad (3.7)$$

where the vector elements  $d_i$  represent the *Euclidean distances*

$$d_i = d(p_j, p_k) = \|p_k - p_j\| \quad (3.8)$$

between the neighborhood arrays of pixels  $p_j$  and  $p_k$ , where  $p_j, p_k \in N$ ,  $j, k \in [1, (n^2 - 1)/2]$  and  $n$  is the size of the neighborhood  $N$ . More precisely, such distances between arrays correspond to the differences of the color values of the arrays elements. In fact, arrays are used to describe the L-shaped neighborhoods and they are constructed to contain the color values of each pixel of the neighborhood, concatenated to build a long vector. The difference between two neighborhoods is then quantified as the difference between their neighborhood vectors. The quality of a match is

determined by calculating the (possibly weighted) sum of squared differences between the already colored pixels around the current output pixel and the pixels surrounding a candidate pixel in the input image.

Although the L2 norm does not provide a quality measure of perceptual similarity between images, it is very fast and easy to compute; moreover it performs adequately for texture synthesis. Nealen and Alexa [142] also recognize that the use of metrics based on human perception does not noticeably improve the results.

### Weighting the neighboring pixels

The pixels belonging to the neighborhoods are weighted through a Gaussian function, in order to assign different relevance to the pixels of the neighborhoods. The pixels adjacent or near to the center are assigned a higher weight, as the correlation with the central pixel is stronger, while the correlation factor decreases when considering the external pixels.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (3.9)$$

In this way, the local structure of the texture can be emphasized. The circular weighting scheme used in this approach helps preserving continuity when performing the synthesis along the varying curved direction of the vector field. Furthermore, as opposed to standard texture synthesis approaches, where the target is to reproduce an existing starting pattern, here a directional pattern is synthesized along the curved lines of a vector field, and for this reason the currently synthesized pixel constantly conditions changes in the input sample, and strong locality and particular attention on the central pixel and its neighborhood is required to achieve continuity in the visualization. Hence, a relationship between the radius of curvature and the *standard deviation*  $\sigma$  (and thus with the variance  $\sigma^2$  of the Gaussian) is beneficial to achieve smoothness in the resulting visualization.

Another weighting approach is proposed in Chapter 4, where novel anisotropic and non-uniform weighting schemes are investigated and presented, even more sensitive to directionality and change of curvature, in order to further enhance the property of directionality in vector field visualization (Fig. 3.25).

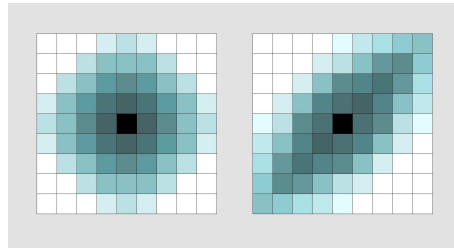


Figure 3.25: Circular and elliptical (§ 4.3) weighting schemes.

### 3.3.5 Finding the best matching pixel

In this way, the neighborhood of the output pixel is compared with all the similarly shaped neighborhoods inside the particular input sample that is under consideration for that specific current output vector value (Fig. 3.26). Such comparison leads to a *minimum*. The neighborhood with minimum distance - in terms of pixel values differences - corresponds to the most similar neighborhood:

$$N_{best_{in}} = \operatorname{argmin}_{N_{in}} d(N_{out}, \{N_{in}\}) \quad (3.10)$$

$$d_0 = d(N_{out}, N_{best_{in}}) \quad (3.11)$$

Consequently, the central pixel of such neighborhood has the highest probability to fit in the output and can be chosen to be set at the current output location. In this way, each pixel in the output is set under best-matching criteria. During the search process, a candidate pixel  $P_{in}$  may come into question when the distance between his neighborhood and the output neighborhood satisfies

$$d(N_{out}, N_{in}) \leq d_{threshold} = (1 + \epsilon) \cdot d_0, \quad \epsilon > 0 \quad (3.12)$$

where  $d_0$  is a minimal distance set as constraint, and  $\epsilon$  is a positive infinitesimal (for instance  $\epsilon = 0.1$  in [55]). Also in [55] a normalized sum of squared differences metric  $d_{SSD}$  is used as suitable distance. This leads to

$$\Omega(P_{in}) = \{P_{in} \subset I_{in} : d(N_{out}, N_{in}) \leq d_{threshold}\} \quad (3.13)$$

where  $\Omega(P_{in})$  is a set of pixels in the input image  $I_{in}$  that are best candidate to fit at the current output location. Randomly, one of such pixels can be chosen to be set in the output. Also the parameter  $\epsilon$  in the definition of the threshold distance  $d_{threshold}$  may contribute with some variance to produce randomness, avoiding not realistic repetitive effects in the output image. The distance or *match value* between two neighborhoods  $d(N_{out}, N_{in})$  is the component-by-component sum of the squared differences. Anyway, repetitive unwanted effects especially occur in standard texture synthesis, while I experienced that using the proposed approach to vector field visualization does not produce such artifacts, also due to the intrinsic variation of the vector field that constantly uses different input seeds. For this reason, and in order to speed up the best pixel search, I adopt a variation in the *searching process*. After having tested and performed the best pixel search using exhaustive and deterministic search, and having then tested the use of threshold plus variance setting, I alternatively execute the search allowing a `break` as soon as a *zero distance* or, since there might not be any perfect match, a user-set *minimum distance*  $\Delta_{min}$  is reached (see pseudo code below, § 3.4.2). This, besides allowing a faster process, also proved not to suffer from additional artifacts or loss of quality. To avoid finding always the same neighborhood with zero distance, it is sufficient to let the search in the input sample regularly start from a different pixel or constantly change the pixel count. In this way, the best fitting pixel can be chosen in a semi-stochastically way from a set of equally best matching pixels. Anyway, the fact that a matrix of samples, instead a single one, is used as seed, alone guarantees that no repetition artifact occurs. Although I did not experienced the occurring of repetitive artifacts, in case it is desired (*e.g.* if appearing too synthetic) to insert more variation for realism (*randomization* of the sample), it is straightforward to non-uniformly modify the input samples, for instance adding rotational, horizontal or vertical jitter in directional seeds.

### 3.4 Algorithm description and implementation

In this section, I more formally describe the algorithm implementation, presenting pseudo code and illustrating variables and methods used in the synthesis process.

Mathematically, I need to define a function  $\mathcal{F}$  that, as introduced above (§ 3.1.1), takes a vectorial data set and an input sample pattern  $I_{in}$  (or sample set  $\{I_{in}\}$ ) to a new output image  $I_{out}$ , which represents the vector field  $\Phi$ . As explained, the input parameters that play a role in the synthesis process are the input sample that determines the output appearance, the MRF probabilistic search, which leads to the definition of pixel neighborhoods, and the vectorial data set; this schematically can be expressed as:

$$I_{out} = \mathcal{F}(I_{in}, N, \Phi) \quad (3.14)$$

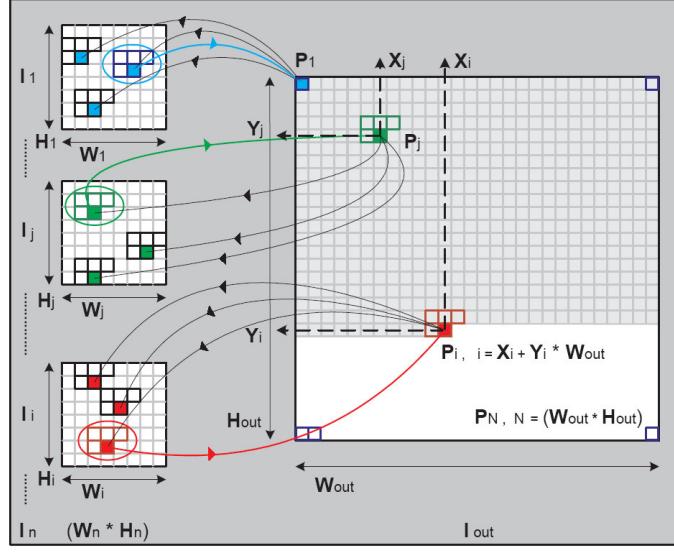


Figure 3.26: Proposed approach: unlike basic texture synthesis approaches (Fig. 3.5), the best matching pixels are here derived from diverse samples that resembles the vectorial information.

In the remainder of this section, I illustrate the required steps in detail.

Let  $\Phi$  be a vector field in  $d$  dimensions over  $\mathbb{R}^2$ :

$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^d \quad (3.15)$$

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

$$\tau(\vec{v}) : [0, 1]^2 \rightarrow [0, 1]^c \quad (3.16)$$

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  $i$ -th output pixel  $P_i$  at  $(x, y)$  is computed using an appropriate neighborhood  $N_{x,y}$  (or neighborhood pyramid  $\{N_{x,y}(l)\}$ , being  $l$  the level of the image pyramid - refer to § A.2.1) of that pixel and the MRF texture synthesis method with example texture  $\tau(\Phi(x, y))$ . The pixels  $\{P(x_i, y_j)\}$  of a general  $n$ -sized squared neighborhood can be defined as:

$$N_{x,y} = \{P(x_i, y_j)\} \quad \begin{cases} x_i \in [x - h, \dots, x + h] \\ y_j \in [y - h, \dots, y + h] \end{cases} \quad (3.17)$$

where  $i = x + y \cdot W$ ,  $h = (n - 1)/2$  defines the offset of the neighboring pixels *w.r.t.* the center,  $W$  is the output width and  $n$  is the neighborhood size.

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 and 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. In the *Euclidean space*,  $\Phi$  can be expressed as

$$\Phi(x, y) = \langle \Phi_x(x, y), \Phi_y(x, y) \rangle = \Phi_x(x, y) \cdot \hat{i} + \Phi_y(x, y) \cdot \hat{j} \quad (3.18)$$

where  $\Phi(x, y) \cdot \hat{i}$  and  $\Phi(x, y) \cdot \hat{j}$  are, respectively, the components of the vector  $\Phi(x, y)$  along the  $x$  and  $y$  axis of the Euclidean plane, and  $\hat{i}, \hat{j}$  the versors, or orthonormal vectors. The *magnitude* can then be easily computed using an appropriate norm of the values, *i.e.*

$$A(x, y) = \|\Phi(x, y)\| = \sqrt{\Phi_x(x, y)^2 + \Phi_y(x, y)^2} \quad (3.19)$$

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

$$\theta(x, y) = \angle\Phi(x, y) = \arctan \frac{\Phi_y(x, y)}{\Phi_x(x, y)} \quad (3.20)$$

is the angle of the tangent in the vector field relative to the  $x$ -axis, and we let the *direction* correspond to the phase of the vector.

As above introduced, a straightforward approach for the mapping from vector values to example images would be to use the information in  $A$  and  $\theta$  to scale and rotate an example image. This is also the approach that I have used to generate the examples presented in this chapter. Typical example images should have a certain directional structure and scale features so that their scale and rotation is easy to perceive (§ 3.2.1). A set of example images I have used is depicted in Figure 3.8. In a first stage of this research, as illustrated in this chapter, I have mostly used simple gray scale images that are constant along one direction and smoothly vary along the other. Figures 3 and 3.30 show the visualization results obtained for the same vector field using different textures (*i.e.* using the first two samples of Figure 3.8. See Figure 3.31 for further examples.

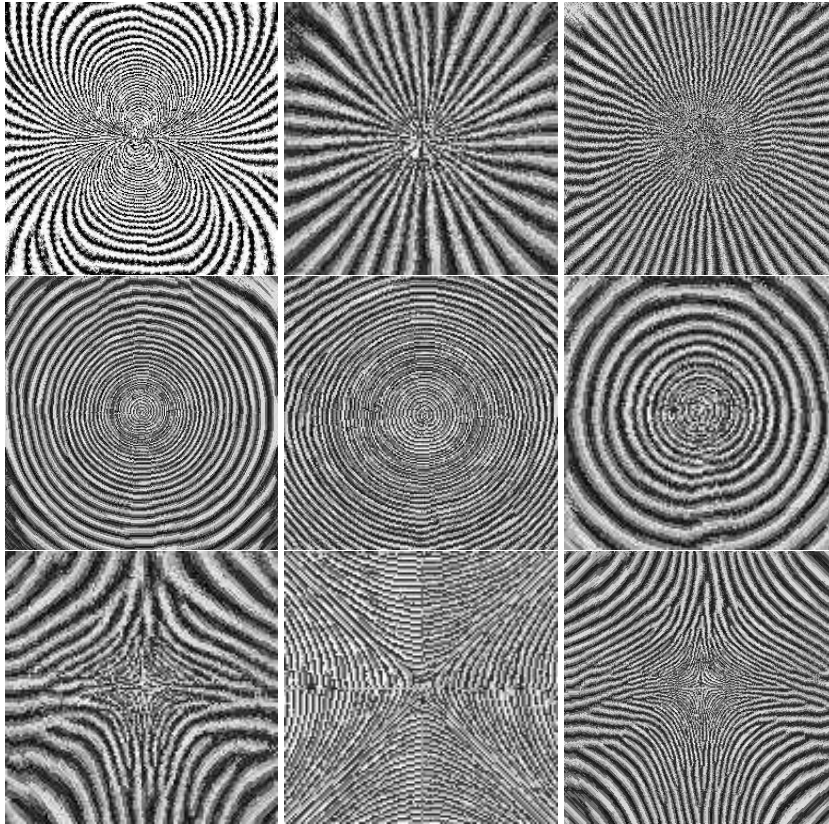


Figure 3.27: Visualization of critical points.

*Critical points* are prominent features in vector fields. I have generated visualizations of isolated critical points to evaluate how prominent those features become in the visualization. The

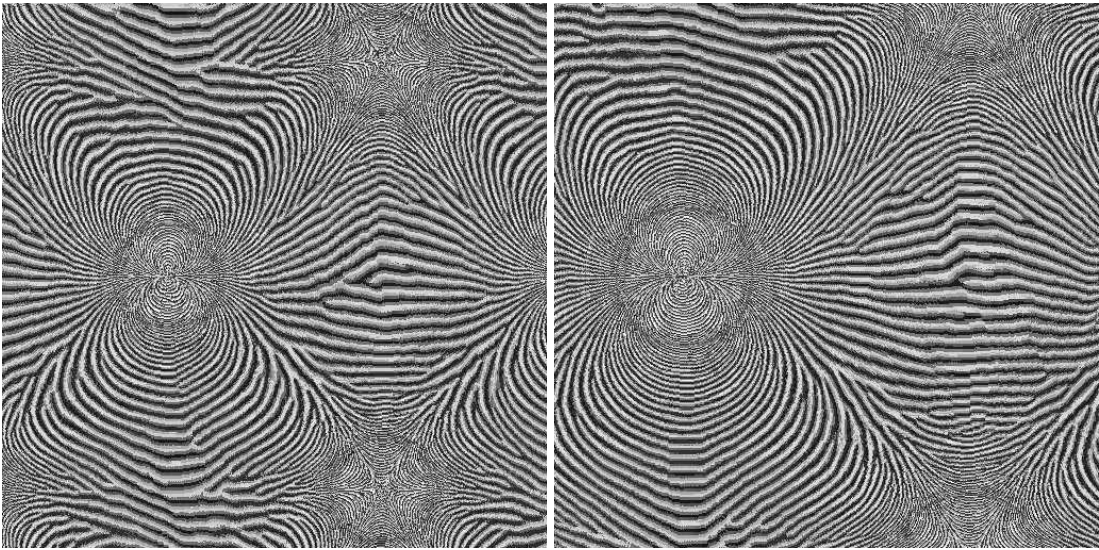


Figure 3.28: Different parts of the vector field visualized using the same output resolution.

result is depicted in Figure 3.27. In addition, it is possible to perform a local analysis and determine critical points using the *eigenvalues* of the *Jacobian*; this is described later in § 6.3. Special textures could then be devoted to the different classes of critical points.

### 3.4.1 Steps of the procedure

To sum up, the structure of the algorithm to realize the proposed MRF-based vector field visualization 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 typically rotated and scaled by these parameters (see Figure 3.15).
- For every pixel, an L-shaped neighborhood is defined, which is constituted by neighboring pixels (the size is user defined).
- Every pixel in the output image is generated in scan order with a routine that searches the most similar pixel in the modified sample image, according to a probability model.
- The probabilities are defined by comparing the distances between the neighboring pixels inside the neighborhoods.

### 3.4.2 Pseudocode

In term of pseudo-code, the synthesis algorithm can be described as in following (Tables 3.1-3.4). The used variables and methods are then listed in Tables 3.5, 3.6.

Function synthesizerPixel-a	
1	for(y = 0; y < H <sub>out</sub> ; y++) {
2	for(x = 0; x < W <sub>out</sub> ; x++) {
3	N <sub>x,y</sub>   <sub>out</sub> = calculateNeighborhood(x, y);
4	A = calculateMagnitude(x, y);
5	θ = calculatePhase(x, y);
6	resizeInput(A);
7	rotateInput(θ);
8	for(j = 0; j < H <sub>in</sub> ; j++) {
9	for(i = 0; i < W <sub>in</sub> ; i++) {
10	N <sub>i,j</sub>   <sub>in</sub> = calculateNeighborhood(i, j);
11	distance <sub>i,j</sub> = compareNeighborhood(N <sub>x,y</sub>   <sub>out</sub> , N <sub>i,j</sub>   <sub>in</sub> );
12	}
13	}
14	minDistance = findMinimumDistance({distance <sub>i,j</sub> });
15	bestMatch = getBestPixelValue(minDistance);
16	I <sub>out</sub> (x, y) = synthesizeOutputPixel(bestMatch);
17	}
18	}
19	return I <sub>out</sub> ;

Table 3.1: Synthesis procedure (specially designed for rotation and scaling transformations).

## 3.5 Results and discussion

### 3.5.1 Comments: limitation and benefits

The size of a sample image plays a critical role in the rotating and 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 3.29). Large structures allow to use larger neighborhoods for comparison and are likely to yield smoother results. Yet, this comes at the price of locality.

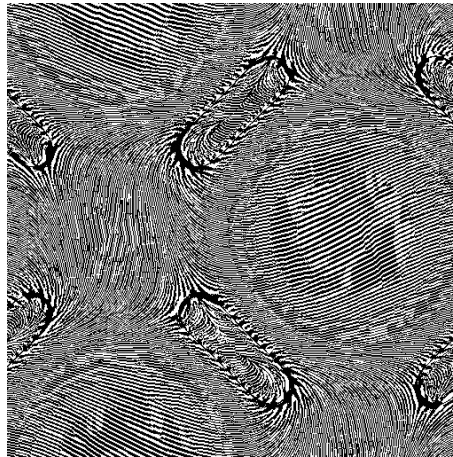


Figure 3.29: Using a small scale example texture might lead to aliasing artifacts in the synthesized visualization.

The core of the synthesis algorithm computes the similarity functions and probabilities for the choice of the best matching pixels in run time. Due to the fact that the input samples vary with respect to the vector values at the various output locations, it is not possible to pre-compute this part in order to speed up the calculation. Computation time is the main drawback; this is an



intrinsic characteristic of pixel-based algorithms. However, the time needed for generating the visualization is identical to the synthesis method used for standard textures. Rotated and scaled versions of the examples are precomputed and fetched from look-up tables.

The method attempts combining the intuitivity of iconic mapping from *direct visualization* techniques with the locality and powerful encoding typical of *texture-based techniques*, also allowing the extraction of physically meaningful features of interest, such as in *geometric* and *feature-based visualization* for a better communication of the field behavior. 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, especially thanks to the statistical search procedure for the best matching pixel. This method can faithfully represent the structure of arbitrary complicated vector fields. Although there is no need for interpolation methods or robust numerical integration algorithms, this approach can simply but precisely visualize vectorial data sets.

**Locality of calculation, controllability** Depending on the smoothness and continuity to achieve, but also on the possibility to offer sufficient distinguishable visual representation and encoding, the input matrix may consist of (at most) as many samples as the different vector values assumed by the vector parameter under consideration (parameter range), or it may consist of a sampled version of this range, which permits to reduce the complexity and number of operations, still achieving satisfying continuous outputs with enough variation in the information mapping. Locality of calculation thus yields a precise and reliable representation of the vector field information. Generating the output image on a per-pixel basis not only allows to change the direction and resolution of the texture patterns, but also several texture characteristics according to any arbitrary transformation function (§ 6.4) that reflects variation in the parameter values of the vector field.

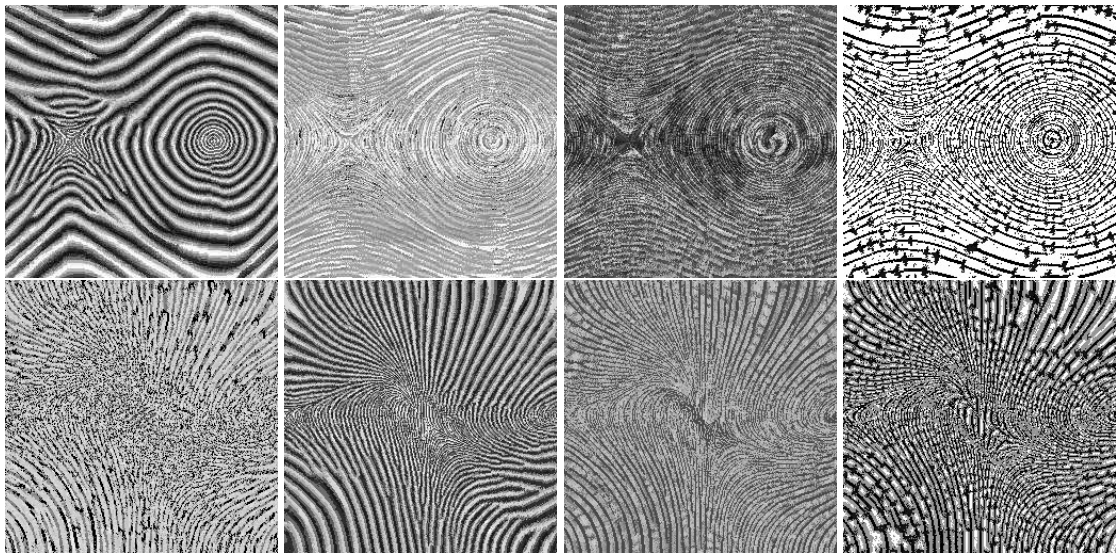


Figure 3.30: Synthesized outputs: two vector fields are obtained both using two sets of different input sample images chosen from Figure 3.8

**Versatility** The sample-based visualization provides a wide variety of appearances and visual attributes, to improve the quality of the visualization and to contribute to an intuitive and meaningful data representation. Various kinds of textures can be generated using a few parameters and can be used as seed for the resulting appearance, producing various types of images. The presented method can depict the data information in an intuitive and easy to understand way, which is desirable in every visualization approach. In fact, a severe drawback of many established existing techniques is that they are often even too sophisticated

and require a significant amount of interpretation to understand the relevant structures of a vector field from its visual representation.

**Generality** The approach is fully general: every arbitrary 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. High vorticity, as well as strong curvature areas and singularities can be represented without limitations. The simplicity and versatility of this algorithm allow its application in several different fields, representing a confluence of signal and image processing, computer graphics and vision, scientific visualization and engineering applications.

**Ease of use** The method is easy to use and implement. Meaningful mapping between field parameters and image attributes is easy to define or select. For instance, a meaningful mapping from vectors to examples can be defined by using phase and amplitude of the field to rotate and scale a single sample texture. Once the parameters are set, the image generation is automatic, but the user is still allowed to personalize in part the visualization in a post-processing stage. The basic concepts can be mathematically described using concise formalism and a few simple equations.

**Customizability** Process customization is easy thanks to a set of control commands. In a *preprocessing* and *initialization* stage, it is necessary to set the inputs of the system, and eventually select some visualization options. At a post-processing stage, again mapping options can be applied to the generated image, offering the possibility to analyze the output under different perspectives, testing different available encoding operators, which is particularly advantageous, especially to optimize the perception of multiple co-existent scalar distributions (see also Chapter 6).

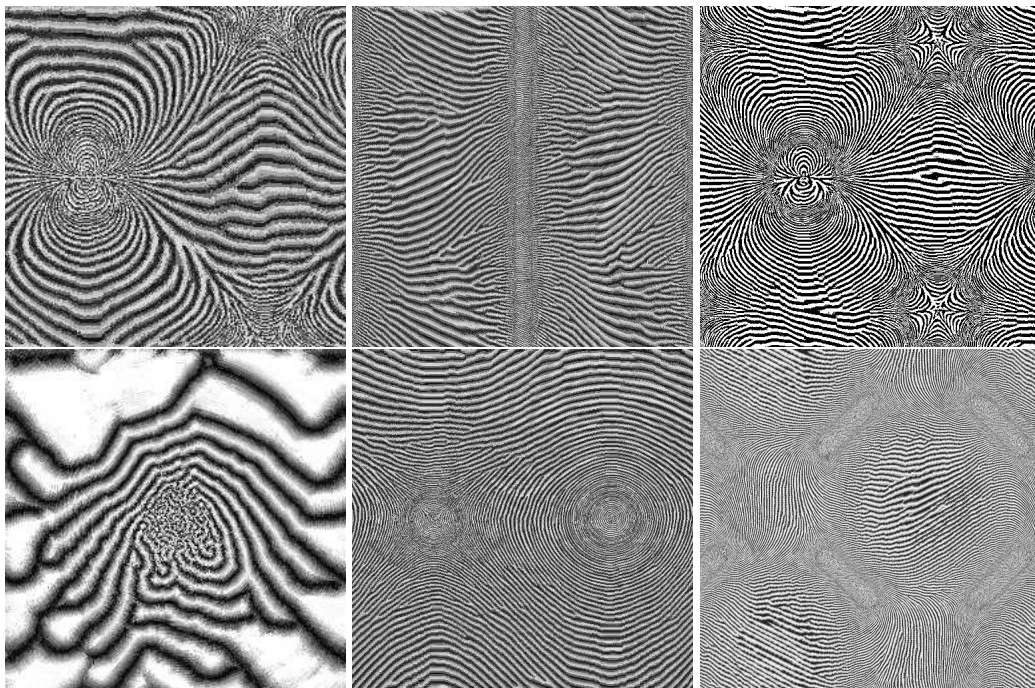


Figure 3.31: Examples of synthesized vector fields.

### 3.5.2 Contribution

The approach to vector field visualization presented in this chapter 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.

It is fairly straightforward to use special example textures for critical points or special application specific values in the vector field; furthermore, incorporating this feature should lead to stronger visual results. For this reason it is important to exploit the idea of using special example textures for critical points or special application specific values in the vector field.

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, I feel that more investigation for suitable mappings from data values to example textures is needed, and I go on in this direction in Chapter 5, 6. Finally, the current implementation would benefit from using the latest possibilities in speeding up the texture synthesis computation.

To test the robustness of the algorithm under various conditions, I analyzed a number of different synthetic distributions, specially designed to characterize different vector field classes, also characterized by singularities, areas of strong curvature or rapid change of direction. The final images effectively capture the input sample appearance and represent the vector field information.

In summary, the proposed algorithm allows a direct mapping of the information contained in the data onto textural visual representations. This results in a straightforward approach to vectorial data visualization and controlled texture synthesis, regardless to the data set complexity.

Function <code>synthesizePixel-b</code>	
1	<code>for(y = 0; y &lt; H<sub>out</sub>; y++) {</code>
2	<code>for(x = 0; x &lt; W<sub>out</sub>; x++) {</code>
3	<code>{f<sub>(x,y)</sub>} = calculateFeatures(x, y);</code>
4	<code>I<sub>in</sub> <sub>(x,y)</sub> = transformSample({f(x, y)});</code>
5	<code>}</code>
6	<code>}</code>
1	<code>for(y = 0; y &lt; H<sub>out</sub>; y++) {</code>
2	<code>for(x = 0; x &lt; W<sub>out</sub>; x++) {</code>
3	<code>N<sub>x,y</sub> <sub>out</sub> = calculateNeighborhood(x, y);</code>
6	<code>{N<sub>i,j</sub>} <sub>in</sub> = calculateNeighborhoods(I<sub>in</sub>);</code>
7	<code>bestMatch = compareNeighborhoods(N<sub>x,y</sub> <sub>out</sub>, {N<sub>i,j</sub>} <sub>in</sub>);</code>
8	<code>I<sub>out</sub>(x, y) = synthesizeOutputPixel(bestMatch);</code>
9	<code>}</code>
10	<code>}</code>
11	<code>return I<sub>out</sub>;</code>

Table 3.2: Pre-computation of field features and input samples for generalized synthesis procedure. Note that in this case the input samples  $I_{in}(x, y)$  ( $W_{in} \times H_{in}$ ) are pre-computed transforming  $I_{in}$  in dependance of the features of the vector field calculated at the output locations  $(x, y)$ .

Function compareNeighborhoods-a	
1	for each pixel $(i, j)$ in $I_{in}$ {
2	newDistance = $D(N_{x,y} _{out}, N_{i,j} _{in})$
3	if(newDistance < $\Delta_0$ ) $\Delta_0 =$ newDistance;
4	}
5	bestMatch = getBestPixelValue( $P(i, j)  _{\Delta_0}$ );

Table 3.3: This function exhaustively compares the current output neighborhood with all neighborhoods within the corresponding input sample.

Function compareNeighborhoods-b	
1	for each pixel $(i, j)$ in $I_{in}$ {
2	newDistance = $D(N_{x,y} _{out}, N_{i,j} _{in})$
3	if( $\Delta_\delta < \text{newDistance} < \Delta_0$ ) {
4	$\Delta_0 =$ newDistance;
5	if(newDistance $\leq \Delta_\delta$ ) break;
6	}
7	}
8	bestMatch = getBestPixelValue( $P(i, j)  _{\Delta_0}$ );

Table 3.4: This function compares the current output neighborhood with those within the corresponding input sample until a specified threshold distance is reached.

Variable	Meaning
$I_{in}$	input sample image
$I_{out}$	output sample image
$I_{out}(x, y)$	value of output pixel $(x, y)$
$W_{out}$	horizontal size - width - of $I_{out}$
$H_{out}$	vertical size - height - of $I_{out}$
$N_{x,y} _{out}$	L-shaped neighborhood of pixel $P(x, y)  _{out}$
$A$	vector field magnitude at current output position $(x, y)$
$\theta$	vector field angle of phase at $(x, y)$
$W_{in}$	horizontal size of $I_{in}$
$H_{in}$	vertical size of $I_{in}$
$N_{i,j} _{in}$	L-shaped neighborhood of pixel $P(i, j)  _{in}$
distance $_{i,j}$	difference between the L-shaped neighborhoods
minDistance	minimum distance between $N_{x,y} _{out}$ and the neighborhoods within $\{N_{i,j}\} _{in}$
bestMatch	best match chosen after distance comparison
$\{f_{(x,y)}\}$	array of vector field features calculated at $(x, y)$
$I_{in} _{(x,y)}$	instance of $I_{in}$ transformed according to $\{f_{(x,y)}\}$
$\{N_{i,j}\} _{in}$	array of neighborhoods of the input image pixels
$\Delta_0$	starting value for minimum distance
$\Delta_\delta$	threshold distance

Table 3.5: Table of symbols

Method	Meaning
calculateNeighborhood	calculates the neighborhood of the given pixel
calculateMagnitude	calculates $A$ at output location $(x, y)$
calculatePhase	calculates $\theta$ at output location $(x, y)$
resizeInput	scales the resolution of the input sample
rotateInput	rotates the input sample by the argument
compareNeighborhoods	calculates the distance between the neighborhood of the current output pixel with that of one input pixel
findMinimumDistance	compares all distances and finds the minimum one
getBestPixelValue	within the input image, chooses the pixel, whose neighborhood leads to $minDistance$
synthesizeOutputPixel	sets the current output pixel $(x, y)$ to the value of $bestMatch$
calculateFeatures	calculates features of interest of the vector field at $(x, y)$
transformSample	modifies the original sample $I_{in}$ according to the field features taken as parameters for the transformation
compareNeighborhoods	calculates an array of neighborhoods for the pixels of the considered input sample

Table 3.6: Table of functions



## Chapter 4

# Directional enhancement in texture-based vector field visualization

In the context of this research (Chapter 3), texture synthesis algorithms have been applied for the special application of vector field visualization. As explained later in this thesis this technique can be successfully applied for the production of oriented and controlled textures, offering a variety of effects. Recent related works in Computer Graphics (refer to § 2.2) also demonstrate increasing interest in this sector. The use of textures, in fact, can provide in general a rich and diverse set of possibilities for the visualization of flow data. Nevertheless, textures are often characterized by more complex and structured patterns than those used in the previous chapter.

In this chapter, I introduce new classes of neighborhood models and weighting functions for enhancing vector field direction in a synthesized, texture-based visualization. This investigation provides new insights based on the specification and classification of neighborhood structures for synthesizing a texture that accurately depicts a vector field; it results to be particularly effective for textures applied to vector field visualizations, oriented textures and nonhomogeneous textures in general. Suggestions for neighborhood shape, weighting functions, and similarity metrics are presented in order to improve the synthesis of the resulting image, enhancing its property of directionality and resulted in [204]. This is especially motivated by the need for preserving continuity and smoothness in the resulting visualization while using complex and structured patterns. Textures, in fact, are a traditional and natural instrument 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.

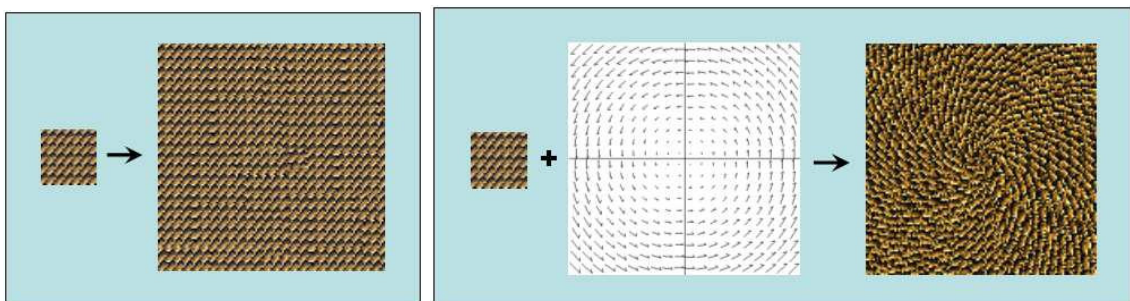


Figure 4.1: Standard texture synthesis vs. controlled texture synthesis: the controlled, field-driven synthesis algorithm requires the input pattern to be synthesized along new directions.

## 4.1 Motivation and contribution

The need for innovative neighborhood specifications also arises from new applications that deal with the synthesis of curved patterns, and also refer to Chapter 7). In order to visualize vector fields and in order to generate controlled textures, patterns can be deformed, curved, modified and controlled to follow given directions and orientations (*cf.* figures 4.1 and 4.2. Consequently, a variety of deformation operations must be used to align a given pattern along a newly specified direction, in addition to modifying the texture color, resolution and appearance in general. Current texture synthesis techniques, however, do not adequately highlight the anisotropy of a texture when synthesized in this fashion. In such cases, the output textures are directional outputs and the synthesis procedure requires to be specially designed and improved in order to highlight and stress the attribute of anisotropy. Although recent approaches (§ 2.2) has arisen, which consider the idea of controlling texture generation, there have been surprisingly almost no investigation and attempts to improve standard neighborhood models and weighting schemes.

Taking into consideration the texture-based visualization of vector fields introduced in the previous chapter, the neighborhood search used to find the best fitting pixels at the output locations requires a dynamic procedure. This procedure adapts to varying conditions by using different input samples (Fig. 4.2) at different output orientations at each step in the algorithm. For this purpose, I build a new class of neighborhoods, based on different pixel weighting functions. The main contribution of this investigation is to provide an approach for accurate synthesis of controlled textures along vector fields, using novel specifications to preserve the directionality of the samples and maintain good resolution while changing the pattern alignment.

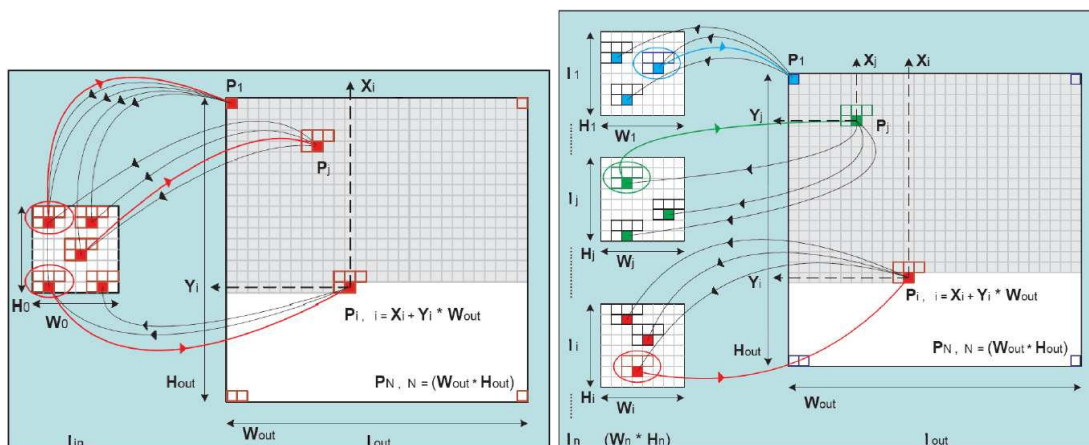


Figure 4.2: Standard texture synthesis *vs.* controlled texture synthesis: the pixels  $P_i$  of the output  $I_{out}$  are set checking the most probable pixels in the input  $I_{in}$  (left); the best matching pixels are derived from diverse samples (right).

## 4.2 Synthesis neighborhoods

### 4.2.1 Standard squared and L-shaped neighborhoods

As introduced in the previous chapters, in standard pixel-based texture synthesis [55, 241], 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 4.4. 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 (Fig. 4.3).



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  $L_2$ -norm, is used to find the neighborhood that is most similar for the selection of the best matching pixel.



Figure 4.3: Different complexity in the structure resolution of the example textures.

### Neighborhood size

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.

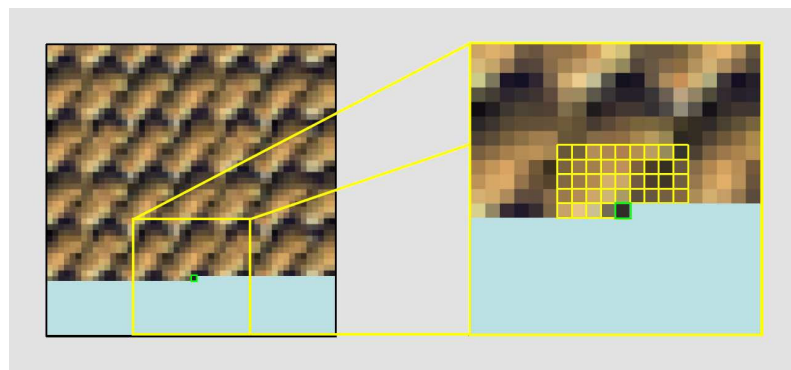


Figure 4.4: Standard texture synthesis: the enlarged picture shows the L-shaped “causal neighborhood” structure (yellow) around the current pixel to synthesize (green). The light-blue portion of the image still needs to be completed.

#### 4.2.2 Different neighborhood shapes

Using the standard texture synthesis approach, however, does not adequately highlight the anisotropy that may be displayed in the sample input texture. For this reason, I build a new class of neighborhoods, based on different pixel weighting functions. The goal is to provide an accurate synthesis of controlled textures along vector fields, using novel specifications to preserve the directionality of the samples and maintain good resolution while changing the pattern alignment.

In the course of this research, I 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 led 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 (especially a *trapezoidal*) weighting scheme is able to account for the two existing pattern directions.

Although it is straightforward to model and use irregular structures, I mainly prefer here, 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.

## 4.3 Non-uniform neighborhood filtering

### 4.3.1 Anisotropic neighborhood model specifications

Using a MRF-based approach to vector field visualization, the input samples are weighted heavily or oriented in a principal direction. I especially use *anisotropic directional patterns* that present a major direction and smoothly vary along the other one. 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.

### 4.3.2 Bilateral filtering

Bilateral filtering has been introduced by Tomasi and Manduchi [211] and later applied in several applications (refer for instance to [52]). 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, possibly in a perceptual meaningful fashion. Closeness refers to vicinity in the *domain* (geometric closeness), similarity to vicinity in the *range* (photometric similarity). A possibility for measuring pixel similarity is to use the CIE-Lab color space, 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 [211]. 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 count 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 count value differences from the center pixel, instead of geometric distances. Hence, the larger pixel count difference, the smaller the pixels' contribution during filtering, resulting in a measure of similarity.

Under these considerations, concepts from bilateral filtering can be used in the weighting schemes approach as illustrated below. The feature of edge-preserving smoothing is particularly advantageous to enhance directionality in texture-based vector field visualization. I use the measures of Euclidean distance (*domain*) and intensity differences (*range*) as similarity metrics. The bilateral filter kernels take a form that depends on the weighting function. I use for instance the Gaussian case; consequently, 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.

### 4.3.3 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



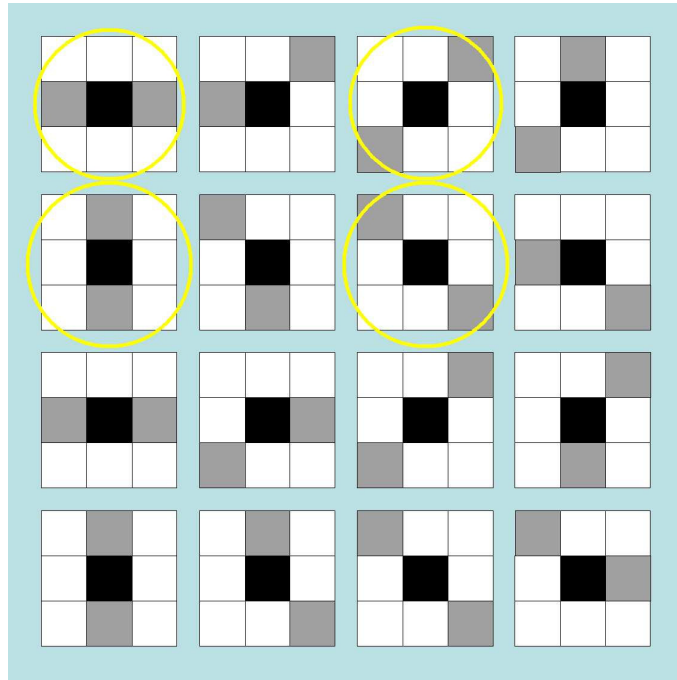


Figure 4.6: Anisotropic weighting of a 3-sized neighborhood, where grey pixels are assigned stronger weights to stress the direction of the vector field. The 4 orientation instances circled in yellow are usually sufficient to produce good results.

Unlike standard texture synthesis, this approach, seen as field-driven texture synthesis, constantly uses different input samples. 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, a pixel-centered approach can be used, 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 the 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 that the synthesis algorithm to not operate in scanline order, as is traditionally done, but in the direction given by the vector field [73]. This ensures that 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 fall-off Gaussian functions or decreasing exponential functions, where the decay factor can be specified in dependence of the field velocity and the curvature radius. Again, there exists a trade-off between computation efficiency and image accuracy. A simple radially symmetric weighting function using a 2d Gaussian can be described by

$$G_C(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \quad (4.4)$$

and leads to the weighting scheme illustrated in figure 4.5-left.

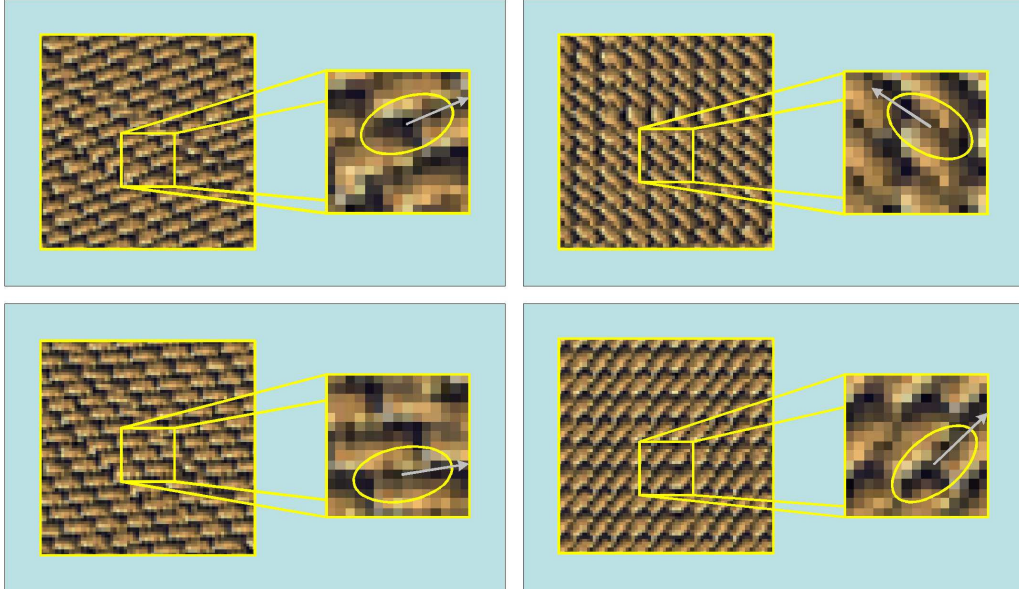


Figure 4.7: Directional enhancement using the elliptical weighting scheme.

#### 4.4.2 Elliptical neighborhood

Next, I adapt the circular neighborhood structure to a more directionality sensitive one: the elliptical structure. I specify this family of neighborhood models for use in controlled oriented texture synthesis. The relative structure is elliptically shaped (Fig. 4.5-right). In other words, the weights of the neighboring pixels are weighted using an *elliptical scheme*.

I use the following expression

$$G_E(x, y) = \frac{1}{2\pi\sigma_x\sigma_y} \exp\left(-\frac{x^2}{2\sigma_x^2}\right) \exp\left(-\frac{y^2}{2\sigma_y^2}\right) \quad (4.5)$$

to obtain an *elliptical Gaussian* for the weighting scheme. In this formula, two standard deviations  $\sigma_x$  and  $\sigma_y$  are used to differently control the fall-off of the Gaussians along the  $x$  and  $y$  directions.

An *oriented elliptical Gaussian* can be then expressed by

$$G_{E,\vartheta}(x, y, \vartheta) = \frac{1}{2\pi\sigma_u\sigma_v} \exp\left(-\frac{u^2}{2\sigma_u^2}\right) \exp\left(-\frac{v^2}{2\sigma_v^2}\right) \quad (4.6)$$

with the change of coordinates:

$$\begin{cases} u = x \cdot \cos\vartheta - y \cdot \sin\vartheta \\ v = x \cdot \sin\vartheta + y \cdot \cos\vartheta \end{cases} \quad (4.7)$$

that controls the rotation by the angle  $\vartheta$ . Examples of possible masks, where intensity values are used for the weighted pixels, are shown in Fig. 4.8-top. 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 (Fig. 4.7). Orientating the major direction

of the elliptical neighborhood along the deformation or the vector field allows for a better control of 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 4.6, I show for simplicity the case of a 3-sized neighborhood, where possible orientation spanning in the range  $[0^\circ, 360^\circ]$  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), one can also distinguish between opposite aligned cases.


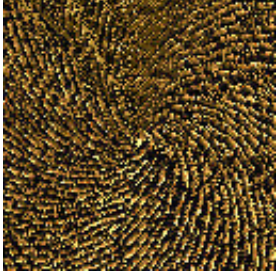

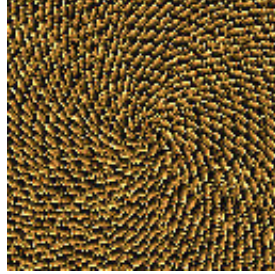

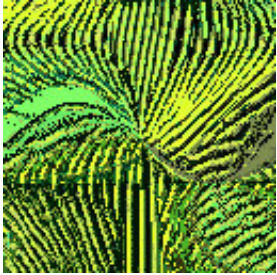
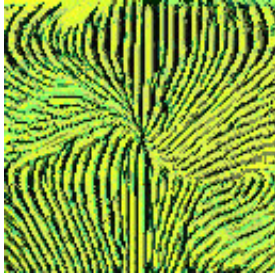
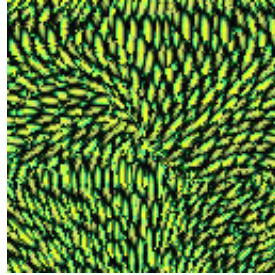
In	squared	circular	elliptical
			
			

Table 4.1: Controlled texture synthesis (fabric and green-scale patterns) along curved vector fields. Using a five-sized kernel, the proposed elliptical weighting scheme (right) shows significant improvements with respect to the circular and the standard squared neighborhood.

Table 4.1 shows a comparison of results obtained using the different neighborhood structures. For each example, I 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). It results that elliptical and circular schemes typically produce more detailed images with fewer artifacts than the squared neighborhood. Additionally, it can be said that the more directional the pattern, the more significant improvements are achieved with the elliptical scheme.

It is necessary to note that good results can be obtained using standard neighborhood structures as well: examples, in part obtained without the enhancement improvement, can be seen in Chapter 7; nevertheless, 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.

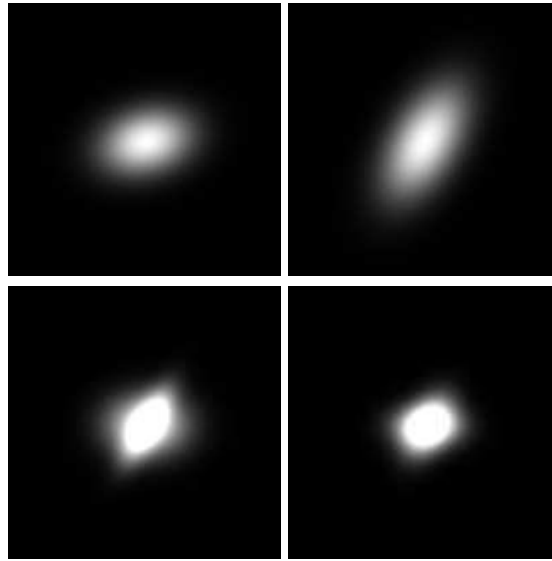


Figure 4.8: Elliptical weighting schemes, with different eccentricity and orientation (top); weighting schemes designed by composition of oriented elliptical Gaussians (bottom).

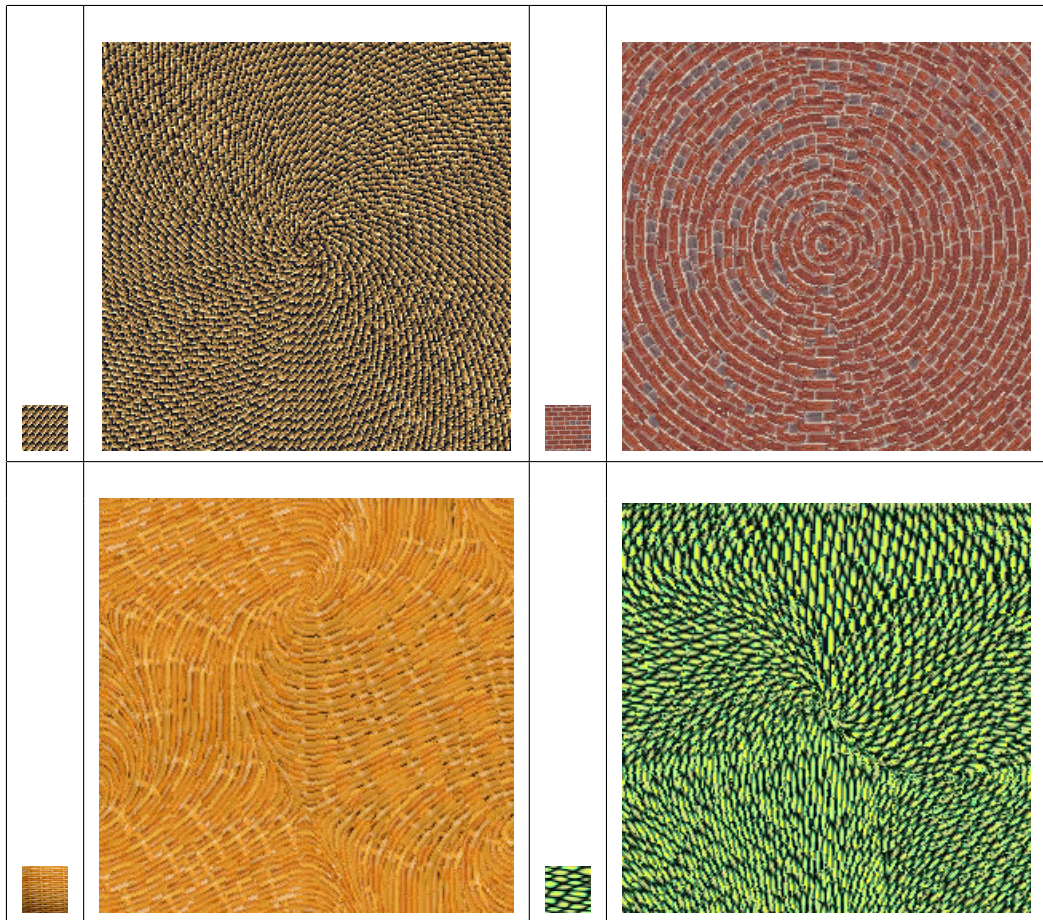


Table 4.2: Obtained results for fabric, bricks, tissue, straw and green-scale patterns patterns.

### 4.4.3 Further weighting schemes: blobs

*Blobs*, first introduced by Blinn [25] are procedural surfaces, which can be generated by summation of several functionals. Simplifying these concepts, summation of 2D Gaussians can be applied here to specify arbitrary weighting schemes, to optimally fit the pattern of structured textures. Examples of possible weighting masks are shown in Fig. 4.8-bottom.

For instance, the sum of the elliptical Gaussians  $G_E(x, y)$  and  $G_{E,\vartheta}(x, y, \vartheta)$  leads to

$$G_{B,\vartheta}(x, y, \vartheta) = \frac{1}{2\pi\sigma_x\sigma_y} \exp\left(-\frac{x^2}{2\sigma_x^2}\right) \exp\left(-\frac{y^2}{2\sigma_y^2}\right) + \frac{1}{2\pi\sigma_u\sigma_v} \exp\left(-\frac{u^2}{2\sigma_u^2}\right) \exp\left(-\frac{v^2}{2\sigma_v^2}\right) \quad (4.8)$$

where the same rotation of coordinates (from equation 4.7) applies. This expression corresponds to an envelope of the two 2d Gaussians, the one oriented by an angle  $\vartheta$ , with respect to the other. This roughly corresponds to a trapezoidal neighborhood scheme, as described in § 4.2.2, which is beneficial for weighting complex textures that present two (orthogonal or not) or more principal directions.

## 4.5 Extensions

### 4.5.1 Spherical and ellipsoidal neighborhoods

MRF-based vector field visualization can be extended to the temporal domain, permitting the visualization of unsteady vector fields. Additional fields of applications can be found in the visualization of tensors, as well as in generating solid textures. To achieve this, a three-dimensional cubic neighborhood is used (see Chapters 5, 7).

Extending the proposed neighborhood specifications to the three-dimensional domain, the weighting functions can be usefully adapted adding a third extra dimension. *Circular* neighborhoods become *spherical* neighborhoods and *elliptical* neighborhoods become *ellipsoidal* neighborhoods. The concepts explained above for the two-dimensional case apply here in the same way, with the only addition of the third dimension. The weighting functions are now defined to operate in the space  $(x, y, z)$  instead of simply in the plane  $(x, y)$ .

### 4.5.2 Sample textures bending

A further improvement can be achieved through specification of optimized input samples. An example is given by patterns containing *multi-frequency* characteristics, as introduced in the previous chapter. This guaranties more continuity and better matching chances between pixels, especially when synthesizing an output using several resized samples.

A further notable improvement can be achieved curving anisotropic samples and using them to compose a broader set of input seeds, characterized by different degrees of *eccentricity* to match different values of curvature for the vector field visualization. In fact, an intrinsic structural limitation of using static example patterns as primitive for vectorial outputs is given by the fact that, although anisotropic inputs still can be arbitrarily modified through rotation and further operators, they do not present anyway curved features. It could occur, in areas of strong curvature of the vector field, that discontinuities arise, since the best matching search within the input pattern is not able to find the optimal pixel corresponding to the output field feature, but only a sub-optimal one.

However, since the algorithm works at a per-pixel level, and since rotation or transformation over the sample can be performed in an accurate continuous way, the presented results show that the proposed steerable texture synthesis still works pretty well and better than previous techniques, especially in critical areas of the vector field, where it presents strong curvature. Although the method produces satisfying results, preliminary investigation, which considers bending input texture examples to build a more complete matrix of input seed providing a more complete input



set, provides promising improvements and interesting extensions that require further research.

## 4.6 Comments

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. Flow textures have traditionally been limited to synthesized renderings where additional attributes are displayed with color, differences in spatial frequency, or contrast enhancements and are added in an artificial manner. The presented approach utilizes the qualities and attributes of textures to visualize scalar distributions and vector fields related to a planar velocity field.

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. Novel model specifications and weighting schemes to model the neighborhoods are introduced. They produce promising results to enhance directionality in texture-based vector field visualization and guarantee continuity also in areas of strong curvature. The use of anisotropic filters for texture synthesis, unlike conventional algorithms, results in fact 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.



## Chapter 5

# Multi-valued visualization

Experiments and numerical simulations yield nowadays high-resolution multivariate data. Data sets can be incredibly rich and complex, consisting of several quantities, defined at numerous points on a two- or three-dimensional grid. The size of these data sets increases rapidly as the dimensionality grows when the data are dependent on time or on one or more parameters [85]. Although the inherent 2d layout of printed paper, the display of the computer screen and its limited spatial resolution restrict the number of graphical icons that can be displayed at one time, techniques can be developed to enhance visualization effectiveness and perception of data dimensionality. In the following, strategies for multi-valued visualization applications are proposed to solve this problem. This research has been in part conducted in cooperation with Professor Victoria Interrante and Timothy Urness, from the department of Computer Science and Engineering, during my stay at the University of Minnesota, in Minneapolis, and has resulted in some related publications [192, 195, 204], and some ongoing work [205]. Beside proposing a solution for the popular task of vector field visualization, the aim of this research is to provide a general approach for the visual representation of multi-valued data sets. As recognized in the STAR of Post *et al.* [155], comparative visualization and multisource comparative data analysis, as well as visualization of multivariate fields with scalar, vector and tensor data are particular important and need additional work. The behavior of complex phenomena in the real world is often described by a combination of many quantities and physical laws: several parameters and data sets have to be taken in account in order to generate a suitable visualization of the carried information, which has to be good perceived and understood. Many *multi-parameter fields* are variable over time and their features may need to be tracked along such temporal variation. *Higher order fields* as well, such as tensors, are present in numerous scientific fields and play a fundamental role, nevertheless they still often remain difficult to investigate and understand, due to their high dimensionality. In addition, from one side, a given field may comprise several variables and need an independent visualization of such information at every point; from another point of view, one may want to visualize more than one vector field at the same time. Such a simultaneous visualization of different fields - *multi-field visualization* - on the same grid is significant for instance for comparison or for the analysis of possible relative influence or interaction. The techniques presented in this chapter attempt enabling users, both experienced or not, to obtain an effective and meaningful visualization of multi-valued data and allow the simultaneous or independent visualization of several scalar and vector distributions, permitting an easy understanding and user-centered analysis of the quantities and features of interest.

With respect to the basic approach described in Chapter 3, I focus here on a more general and broad approach to visualization of vector fields, regardless to their amount of parameters, dimensionality and temporal dependence. To achieve this more complex target, the proposed algorithm is here extended to allow multi-parameter visualization. The extensions go in several directions: in the following I describe the general context of *multi-valued visualization*, together with notions and relative required algorithm adaptations for

- *multivariate multi-parameter fields* and especially *temporal fields*

- *higher-dimensional fields*
- *multi-fields*

Solutions and examples are proposed for each of these cases.

## 5.1 Multivariate fields

### Context and fields of application

Multivariate analysis is important in many areas of scientific inquiry. In the simple case of basic measurements of an element, the description of its condition requires at least the coordinate of its 3d physical position, its temperature, pressure and density at a given time. Hence, in such example, already a plenty of variables are present simultaneously. If the subject matter to be studied is more complicated, it will involve much more dimensions. In addition, some variables often interact with each other and it could be desired to visualize multivariate relationships. Vector variables are for instance velocity, vorticity, magnetic or electric field, a force or the gradient of some scalar field, etc. Tensor variables might correspond to stress, strain or rate of deformation, *et cetera*.

Multivariate visualization comes to the fore when researchers have difficulties in comprehending many dimensions at one time. Researchers of many different scientific and engineering communities (but concepts are also extensible to statistics or human sciences) have always been interested in a deeper understanding of the key mechanism and meaning of complex data sets. Visualization techniques are often considered valuable to meet the demands of multivariate data because of their ability to portray numerous aspects of the data simultaneously.

In this section, I consider *multivariate vector fields*, distinguishing between significant sub-classes:

- vector fields characterized by multiple variables: *multi-parameter fields*
- three-dimensional fields, such as terrains: *height fields*
- fields that vary over time: *temporal fields*

Briefly explained, general attributes can be visualized through data encoding, elevation variable can be represented through bump mapping (related to the  $z$  cartesian coordinate), and temporal evolution can be achieved generating successive field frames, depending on the additional  $t$  temporal coordinate. It can still be considered that, with theory translation and limited effort, the combined visualization of different multi-valued fields is possible. While the first two classes (multi-parameter and height fields) can be treated as done for simpler fields, taking advantage of intuitive encoding strategies, on the other side visualizing time-variant vector fields, depicting their evolution over time, requires a more consistent algorithm extension. For this reason, I briefly describe applications for the first two classes of multi-valued fields (§ 5.1.1, 5.1.2) and spend then more time in explaining temporal fields (§ 5.1.3), which represent a very common and important case of multi-parameter fields.

### 5.1.1 Multi-parameter fields

In Chapter 3, I limited for simplicity the representation of a vector field  $\Phi$  to the representation of its phase and magnitude. The vector field  $\Phi$ , as in equation 3.15, is in general in  $d$  dimensions:

$$\Phi(x,y) \in \mathbb{R}^d = \{\varphi_0 \varphi_1 \varphi_2 \dots, \varphi_{d-1}\} \quad (5.1)$$

Hence, when additional parameters such as general scalar and vectorial distributions are of interest to be visualized, this information can be simply mapped using *encoding strategies*. For

this target, in the next chapter solutions are presented to map data onto visual representations, according to intuitive criteria. This can be achieved also using layers and transparency, to allow data investigation, focusing on one or more variables at the same time, and using several visual representations to depict multiple data attributes.

Using encoding strategies, it is possible to augment the visualization approach described in Chapter 3. This allows simple 2d images to display several variables; it is in fact possible to map additional variables representing them by different shapes, sizes, colors, and locations of complex symbols (*glyph*). In this sense, this leads to a sort of *hybrid visualization technique*, where the proposed dense texture-based algorithm developed using MRF statistical theory is integrated with *global imaging techniques* making use of direct, geometric and feature-based visualization concepts. Symbol size and placement (angle, orientation) can contribute to map an additional variable, or icons can be added to indicate parameters as well. In the next chapter, I provide a more detailed explanation and present an approach to multi-valued visualization based both on *multiple-symbols* (§ 6.4) and on *multiple-views* (§ 6.6).

### 5.1.2 Height fields

Most two-dimensional visualization techniques do not allow the representation of three-dimensional data. Using special encoding methodologies, it is possible to visualize this additional variable. In the presented approach, *height* or *elevation fields* are basically treated like other vector fields, with the difference that their peculiar feature of elevation needs now to be visualized with particular focus. Information encoding using colormaps or glyphs can be used to visualize and highlight the height variable, as well as small-scale geometric texture characteristics.

A particularly effective and perceptually intuitive way to represent such additional information is given by the *bump mapping* technique. When visualizing three-dimensional data sets in the plane, the third dimension along the  $z$  axis can be encoded using bump mapping. Such filter provides a sense of depth in a very effective way. Perception of bump and valley is automatic and a simple luminance map provides an impression of a three-dimensional visualization, still furnishing simple 2d output image, where it is still possible to map further attributes. The intensity values of the bump map image can be interpreted as height displacements.

The *bump mapping* technique was invented by Jim Blinn [24], and was later used for a variety of applications, including BLIC (*Bumped LIC*) [171].

Bump mapping can be easily integrated in the proposed approach, to map the additional scalar  $z$ , augmenting the output image with bumps and depressions (Fig. 5.1). In some cases, a redundant visualization of some parameters may help better data analysis<sup>1</sup>. This method is able to simulate bumps and wrinkles like in a surface, without the need for geometric modifications<sup>2</sup>. In the case examined in this work, the scalar  $z$  is used to obtain a bump map (or *altitude map*) that leads to bumps or depressions according to the sign of the scalar  $z$ . This can be used over the output image, or also to modify the input samples, letting correspond this scalar value only to the lines characterizing the lines of the field flow, and thus leaving the background of the samples unvaried.

### 5.1.3 Temporal fields

Besides being characterized by a set of diverse parameters, vectorial fields also may vary over time. *Time-dependent vector fields* represent a special class of multivariate vector fields. Additionally to other variables, the time domain needs thus to be represented and leads to the generation of successive temporal frames, which depict the evolution of the vector field over time, and which can successively be animated.

<sup>1</sup>By using a redundant representation, different perceptual channels can simultaneously process features of the data distribution [162].

<sup>2</sup>In bump mapping, the surface normal of a given surface is perturbed according to a *bump map*, and the perturbed normal is used instead of the original one when shadows are computed using the *Lambertian* technique. I remand to [24] for a more detailed explanation of the technique.

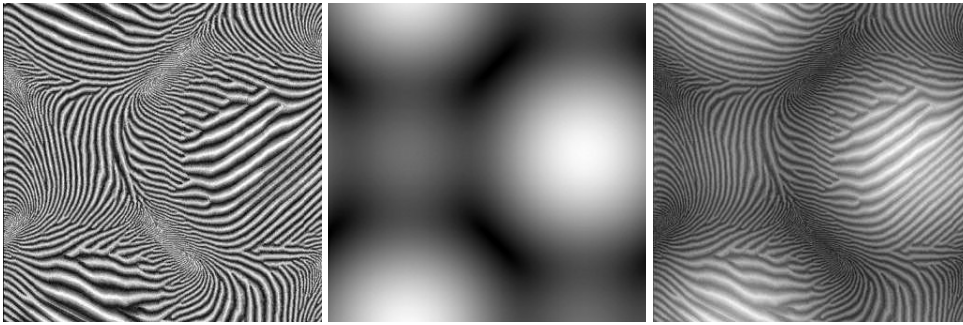


Figure 5.1: Vector field visualization enhanced by a brightness map.

The analysis and the visualization of time-dependent vector fields, in general unsteady fields, is fundamental in scientific visualization and computer graphics. Relevant applications can be found for instance in kinematics, solid mechanics, fluid- and aerodynamics, magnetism and electromagnetism, meteorology, physics, and many others. Despite intensive research efforts and many interesting methods that have been developed, the representation of complex flow fields and multivariate vector fields remains challenging, and the need for a general and adaptive method still exists, especially when visualizing unsteady vector fields.

In this section, a flexible approach to time-variant vector fields is proposed. It incorporates the temporal domain as additional variable as extension of the previously explained method (Chapter 3). This method uses a simple approach to generate a smooth and continuous visualization with correlation in the spatial and temporal domains.

### Handling not-stationary vector fields

Usually, visualizing the evolution of vector fields and the variation of their parameters is difficult and most visualization approaches encounter problems or have difficulties when the fields are not stationary or unsteady. *Unsteady flows* originate from fields that vary over space and time. Temporal vector field visualization is a very productive area of research and numerous techniques exist. For this reason, I remand to [155, 156] for good surveys and more complete state of the art on temporal vector fields and flow visualization.

The most prominent dense and texture-based techniques for flow visualization are adaptation or extensions from existing methods valid for steady field visualization. Many methods base on the Line Integral Convolution (LIC) technique [35] and achieve good correlation. The basic primitive is a noise texture that is smeared in the direction of the vector field. UFLIC (*Unsteady Flow LIC*) is based on the Line Integral Convolution technique [177] and achieves good spatial and temporal correlation. However [99], the images are difficult to interpret: the paths are blurred in regions of rapid change of direction and are thickest where the flow is almost uniform. Spot noise [230, 206] uses a spot as basic primitive: a spot is an ellipse or another shape that is warped and distributed visualizing the vector field. The method has later been adapted [45] to flow visualization and extended [43]. The spot noise technique also has been extended to the visualization of unsteady flows [41], but, as in the steady case, a large collection of spots is created to cover the image, their positions need then to be integrated along the flow, bent along the local pathline, and finally blended into the animation frame. Further methods, still based on particle position integration, have been proposed [138, 13, 98]. Max & Becker [138], van Wijk [231] visualize flows animations respectively by warping textures on triangles, and advecting and motion blurring particles by the flow field. Successively, Crawis and Max [39] extend the method in 3d using texture splats. Traditionally, unsteady flows are represented via a collection of pathlines that originate from user-defined seed points. These trajectories are often visualized in experimental laboratories through the injection of dye into the fluid [99]. As for steady fields, though, using user-defined seed points could lead to problems in the visualization, in case the chosen points miss regions of interest. Ap-

proaches for dense visualization are possible, the major challenge, however, is to guarantee good spatial and temporal correlation and to generate smooth animations. Combining the advantages of Lagrangian and Eulerian formalisms, the LEA method [98, 99] provides a good solution for unsteady flow visualization at interactive rates. In advection, pixel or texels are advected backward (backward coordinate integration) in the direction of the vector field. Accelerated version of the method have been proposed [246]. An interesting extension to LIC, also using concepts from advection is the *image based flow visualization (ibfv)* by van Wijk [232]. He simulates advection and decay of dye by defining each frame of a flow animation as a blend between a warped version of the previous image and a number of background images.

These methods help providing good visualization, but do not always result to be general: as described in the vector field visualization survey (§ 2.3), also their extensions to unsteady field and flow visualization still suffer the same limitations. Direct visualization is intuitive but may miss density and accuracy, geometric visualization directly communicates information but is sometimes too simplified; feature-based visualization may be too specialistic for inexperienced users.

Unlike the existing techniques, the method presented in this thesis allows a straightforward extension for the visualization of unsteady fields as done in the simpler case of steady fields. This technique provides dense representation of time-dependent vector fields; the visualization approach is still based on the proposed MRF algorithm, which makes use of texture synthesis and statistical theory, so that it does not encounter additional difficulties when the field is variable in the temporal domain. This approach does not need critical variations under changed conditions. No physical equation needs to be solved and no time integration is necessary. The key point is still the setting of the pixel color on the base of best matching neighborhood search.

### Steady vs. unsteady fields

*Steady fields*  $\Phi(\mathbf{x})$  are  $n$ -dimensional ( $nD$ ) fields whose lines of flow do not vary over time:  $\mathbf{x}$  represents here the  $n \cdot D$  locations of the vectors within the flow domain  $\Omega$ . The placement of the lines of flow do not change, and if it is desired to visualize the variation of a given parameter over time, color coding or similar direct techniques (for instance using intuitive color table map or advection) can be used to represent particle motion

$$\Phi(\mathbf{x}) = \dot{\mathbf{x}} = \frac{d\mathbf{x}}{dt}, \quad \mathbf{x} \in \Omega \subseteq R^n, \Phi \in R^n, t \in R \quad (5.2)$$

along the flow lines. This ordinary differential equation determines the integral curves of the vector field. A simple algorithm extension can thus communicate dynamics and variation of variables, ideas of relationship and sense of movement in a straightforward manner. In case only a single scalar parameter of the vectorial data set is changing, color shift and changes in a given color range can easily produce a very effective visual communication of the varying phenomenon. In effect, in case of steady fields, the texture pattern representing the streamlines, once computed, remains valid for all frames, being in time-independent fields the topology and magnitude distribution of the flow constant over time.

In the case of more complicated *unsteady fields*, *i.e.* time-dependent,  $\Phi(\mathbf{x}, t)$ :

$$\begin{aligned} \Phi(\mathbf{x}, t) &= \Phi(\mathbf{x}(t), t) = \dot{\mathbf{x}}(t) = \frac{d\mathbf{x}(t)}{dt} \\ \Phi(\mathbf{x}, t) : \Omega \times \Pi &\rightarrow R^n, \mathbf{x} \in \Omega \subset R^n, t \in \Pi \subset R \end{aligned} \quad (5.3)$$

the lines of the flow field vary their trajectories over time, and pathlines are solution of the differential equation for a given starting position  $\mathbf{x}(0)$ . Here, potential visual artifacts, such as pulsation, synchronization, flickering effects must be avoided when animating the sequence of the MRF-based images, which represent the temporal instances of the unsteady vector field. In the formula above  $\mathbf{x}$  is again the spatial reference of the flow,  $\Omega$  is the  $n - D$  Euclidean space, and  $t$  represents the system time, consequently, if  $t$  is considered to be constant, *i.e.* for steady flow data, the more simple case of  $\Phi(\mathbf{x}) : \Omega \rightarrow R^n$  is given.

### Tracking of features

An appropriate mapping of features to visual patterns, symbols or colors highlights the variation of time-dependent parameters and permits to observe them moving or changing during the animation. This allows to follow and track a special attribute of the field during its temporal evolution, or to analyze the behavior of singularities or special areas of interest while the lines of flow vary over time (refer to § 6.3 for topological extraction of features). This is of relevance to describe how features evolve in time. During the evolution, certain events can occur, such as interactions between two or more features, or significant changes in the features shape.

### Achieving temporal and spatial coherence

Let  $\Phi$  be now a time-dependent vector field  $\Phi(\mathbf{x}, t)$ , or here  $\Phi(x, y; t)$ , we aim at visualizing its variation  $\partial\Phi/\partial t$  in time. In the algorithm, each vector  $\vec{v} = \Phi(x, y; t)$  defines an example texture image  $\tau(\vec{v})$ . In the visualization, every frame of the vector field temporal evolution is generated similarly as done in Chapter 3; each pixel at  $(x, y)$  and at time step  $t$  is computed using an appropriate (see detail below) neighborhood or neighborhood pyramid of that pixel and a MRF-based texture synthesis method with example texture  $\tau(\Phi(x, y))$ .

Adapting the proposed technique to accept time as extra dimension allows producing dense smooth visualization of unsteady vector fields in a very uncomplicated way. The suggested algorithm remains texture-based: the field, which may depend on various quantities and on time, still determines how a chosen basic pattern of a texture has to be transformed and adapted to locally represent the features and variation of the field. The texture synthesis method aims at generating globally smooth textures and yields dense visualizations of the vector field. Extending this synthesis in time leads to texture-based animations with frame-to-frame coherence. Smoothness in spatial and temporal domain is achieved. This approach is general and produces unsteady field visualizations in an intuitive and straightforward way.

Briefly, I adapt the algorithm and the basic technique from Chapter 3 to accept time as extra dimension. A set of frames is generated, where each one represents a temporal instant of the vector field evolution; this sequence of frames can be then animated to visualize the field variation. An enlarged 3d neighborhood is introduced and illustrated below as novel structure to specify inter-pixel correlation; in this way it is possible to keep trace of previously synthesized frames and use this information to maintain coherence and continuity along the temporal evolution besides the spatial domain. According to which one of the past frames is being considered to contribute to the next step of the temporal sequence, an adequate weighting of the neighborhood pixels is used. In this way, a sequence of frame-coherent textured images produces the animation.

### One-frame synthesis for multi-valued visualization

For the generation of the starting frame of the temporal sequence, I use the basic algorithm (§ 3.4). As previously done, a given array of vector values or an arbitrary functional expression of the field are accepted as input to generate a dense visualization of variable vector fields. The target is again to produce a vector field visualization, where every point in the output is appropriately colored. This color value should represent the combined information of scalar values, direction, orientation, carried by the field at that specific point and time. The user may choose sample patterns to control the appearance and certain properties of the vector field. In this sense, the visualization remains sample-based, as a space of example images is used to create an appropriate appearance of the resulting field.

The technique is implemented using a multi-resolution representation (§ A.2.1). In this way it is possible to generate static vector fields combining the technique with the controlling action of a deformation field. By adequately modifying the input samples through operators (typically scale  $A(x, y; t) = \|\Phi(x, y; t)\|$ , and rotation  $\theta = \arctan \frac{\Phi_y(x, y; t)}{\Phi_x(x, y; t)}$ ) the vector field is turned into a visual representation. The basic technique described for static vector fields is used to visualize the starting



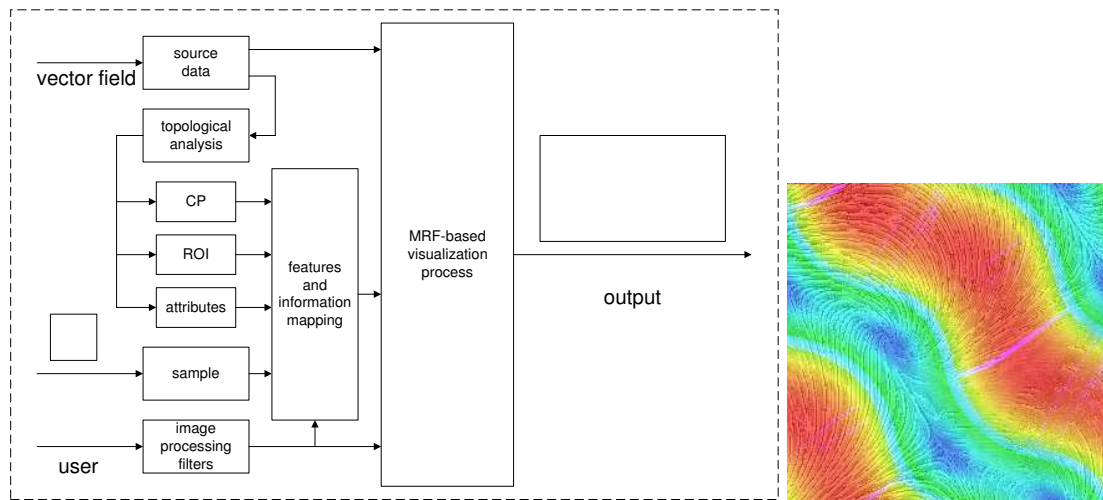


Figure 5.2: Block diagram for the generation of one frame (left) and resulting example (right).

step of a field temporal evolution, and it is then extended to generate the succeeding frames. Figure 5.2-left sketches the pipeline to initialize the visualization process. In the scheme, the blocks on the left side are the inputs of the system. The source file may be in form of raw data or functional description. Through topological analysis prominent features can be calculated while they vary over time: this includes identifying critical points (CP), choosing regions of interest (ROI) and calculating further flow attributes, as direction, orientation, magnitude, divergence, curl, etc. At this step, the user can directly contribute to personalize the visualization process. He may decide how to map those field features in order to get an intuitive visualization and observe how some peculiar attributes change and move over time. For instance one can highlight the field velocity through brightness intensity, or map a scalar temperature or pressure field to a given color range. He can also use image processing filters (*e.g.* blurring, sharpening, embossing) to progressively modify the input example in an adaptive way, by mapping features variations to gradual change of filter parameters. Some of the described blocks in Figure 5.2 are optional: in the simplest case just the example texture and the source data are available as input for the system. The user still may adaptively map the field attributes in a post-processing phase.

### Synthesizing consecutive frames

When using the basic approach to static vector field visualization, the method achieves continuity in the spatial  $(x, y)$  domain through the per-pixel synthesis that takes advantage of spatial locality. On the other hand, in order to preserve continuity in the temporal domain  $t$  as well, I take now into account information from the previous steps. The MRF model helps to combine and organize spatial and temporal information by introducing strong generic knowledge about the features to be estimated, namely, the pixel values of our visualization. Consider to synthesize a still frame representing the structure of the flow at a fixed time  $t$ : an appropriate value is assigned to each point of the output frame after taking into consideration information from the *spatial* and the *temporal domain*. This information is namely derived using the pixel neighboring pixels plus a neighborhood, or set of neighborhoods (§ A.2.1), at the corresponding position from the previously synthesized step, or steps, at time  $(t-1)$ , or  $\{(t-1), (t-2), \dots\}$ .

The block scheme in Figure 5.3 illustrates the generation of the output sequence. Note that the scheme of Figure 5.2 has to be inserted here in the pipeline and it represents the dashed block. Using the previous iterations as feedback, the set of subsequent frames is synthesized in a recursive way. The iteration loop is performed  $T$  times, where  $T$  is the desired number of frames, determining the length of the output sequence. The animation of the whole set shows then the vector field in motion.

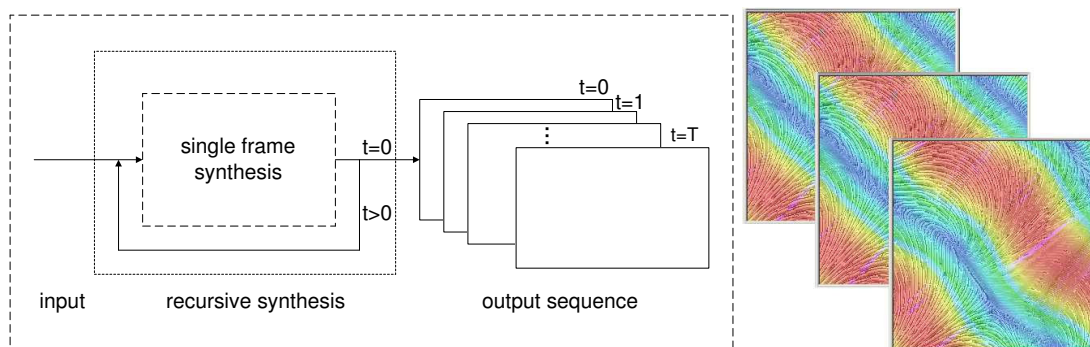


Figure 5.3: Block diagram (left) for visualization of unsteady fields frames (right).

### Three-dimensional neighborhood - cubic neighborhood structure

In order to accurately visualize unsteady fields, a special neighborhood  $\mathcal{N}^3$  is here introduced, which is constructed using a cubic structure (Figure 5.4). The use of a three-dimensional model confers continuity to the temporal visualization. This novel structure can take into account coherency from the surrounding pixels in the current planar image, and, in addition, it includes information contained in the latter already generated frame.

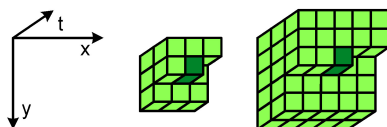


Figure 5.4: Spatio-temporal coordinate axis system and 3d neighborhoods for  $n=3$ ,  $n=5$ .

### Pixel weighting

The pixels that build the extended neighborhood have to be appropriately considered: in the spatial domain they should be radially (or elliptically: § 4.3) weighted around the current one. As explained in Chapter 4, a non-uniform weighting of the pixels within the planar spatial neighborhood can help achieving better results in static field visualization. Accordingly, this also applies to the temporal cubic neighborhood: the pixels in a 2d planar neighborhood should be considered with decreasing weighting factor when far away from the neighborhood center, and this should happen in the 3d neighborhood as well. The weighting function should decrease from the neighborhood midpoint both in the planar L-neighborhood and in the previous temporal squared neighborhoods. The preceding temporal frames have to contribute with non-uniform weights, since pixels in the actual frame present less correlation with pixels belonging to previous frames. The number of the underlying layers, which have to be taken into consideration at each step, depends on the chosen neighborhood size, which in turn depends on the complexity of the example pattern. In the simplest case of a regularly and fine structured sample (such as those used in Chapter 3), usually a 3-sized neighborhood suffices, so that just one previous frame is required to visualize a new one. In case of 5-sized neighborhood, the synthesis needs to acquire information from two previous frames, and so on. In general, for a  $n$ -sized neighborhood,  $(n-1)/2$  previous frames are taken into consideration. In this way, the neighborhood structure is able to correlate pixels in space and time along the spatial and temporal evolution of the vector field. Consequently, as anticipated in Chapter 4, and especially in § 4.5.1, this correspond to build *spherical* or *ellipsoidal* weighting schemes for the neighborhood models (see Figure 5.5).

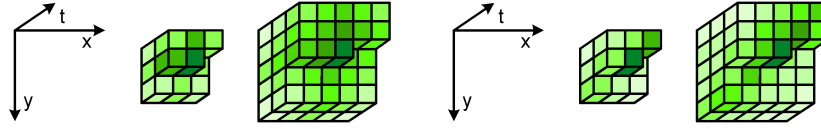


Figure 5.5: Spherical (left) and ellipsoidal (right) weighting schemes for 3D neighborhoods, with respect to the uniform scheme of Fig. 5.4. for  $N=3$ ,  $N=5$ .

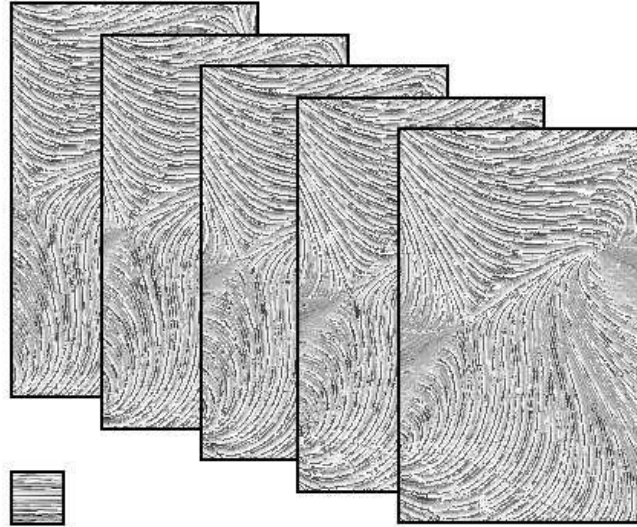


Figure 5.6: Example of temporal frames generation. A grey-scale sample is adapted locally to produce a sequence of frames for the time-varying vector field.

### Generation and animation of successive temporal frames

The whole process is implemented to work in an automatic way, so that, starting from a given frame, a sequence of its evolution in time is generated without the need of further intervention. In this manner, the algorithm computes temporal series - in arbitrary length - of correlated images. Every still frame depicts the instantaneous structure of the flow, and the sequence of such frames can be successively animated to reveal the flow evolution.

### Temporal frames animation

In order to generate an animation of  $\Phi$  over an arbitrary period of time  $T$ , successive frames are generated. Let  $F_t = \{F_0, F_1, \dots, F_i, \dots, F_T\}$  be those frames,  $F_i = \Phi|_{t=i}$  represents the still vector field at time  $t = i$ . The desired set of frames  $F_t$  is then generated as follows:

$$F_t(x, y) = \begin{cases} \text{synthesis2D}(\tau(x, y), \Phi(x, y; 0)) & t = 0 \\ \text{synthesis3D}(\tau(x, y), \Phi(x, y; t), F_{t-1}(x, y)) & t > 0 \end{cases} \quad (5.4)$$

where `synthesis2D` represents the visualization process for the vector field at time  $t = 0$  (see the sub-section *One-frame synthesis* above, and refer to the method `synthesizePixel` of Table 3.1 and 3.2). The method `synthesis3D` (refer to the sub-section *Consecutive frames* above, and see table 5.1) performs the temporal visualization of successive frames and gets previously visualized instantaneous vector fields as input. Note that in table 5.1 the case of a 3-sized neighborhood is considered; consequently, only the previous frame is taken into consideration for the synthesis of the next one. The relative considerations for larger  $n$ -sized neighborhoods

Function synthesis3D(t)	
1	for(x = 0; x < W <sub>out</sub> ; x++) {
2	for(y = 0; y < H <sub>out</sub> ; y++) {
3	N <sub>x,y,t</sub> <sup>3</sup> <sub>out</sub> = calculateNeighborhood3D(x, y, I <sub>out</sub> (x, y, t-1));
4	A = calculateMagnitude(x, y, t);
5	θ = calculatePhase(x, y, t);
6	resizeInput(A);
7	rotateInput(θ);
8	for(i = 0; i < W <sub>in</sub> ; i++) {
9	for(j = 0; j < H <sub>in</sub> ; j++) {
10	N <sub>i,j,t</sub> <sup>3</sup> <sub>in</sub> = calculateNeighborhood3D(i, j, I <sub>in</sub> (x, y, t-1));
11	distance <sub>i,j,t</sub> = compareNeighborhood(N <sub>x,y,t</sub> <sup>3</sup> <sub>out</sub> , N <sub>i,j,t</sub> <sup>3</sup> <sub>in</sub> );
12	}
13	}
14	minDistance = findMinimumDistance({distance <sub>i,j,t</sub> });
15	bestMatch = getBestPixelValue(minDistance);
16	I <sub>out</sub> (x, y, t) = synthesizeOutputPixel(bestMatch);
17	}
18	}
19	return I <sub>out</sub> (t);

Table 5.1: Temporal synthesis procedure (consistent with synthesizePixel-a of Table 3.1).

Function animateFrames	
1	I <sub>out</sub> (0) = synthesis2D();
2	for(t = 1; t < T; t++) {
3	I <sub>out</sub> (t) = synthesis3D(I <sub>out</sub> (t-1));
4	}
5	loop({I <sub>out</sub> (t)});

Table 5.2: Temporal frames generation and succession

are reported above, and  $(n - 1)/2$  previous frames are considered in a general case. Both methods are explained for clarity in single resolution; straightforward extensions to multiresolution are achieved using the concepts explained in § A.2.1.



Figure 5.7: A set of generated temporal frames.

## Results and Discussion

The results in Figures 5.7, 5.8 show some frames extracted from longer generated sequences. Different patterns are used as seed to generate unsteady fields evolution. Figure 5.9-right shows two sets of successive frames; here, each one shows the underlying time-dependent vector field structure together with color coding, which is used to map the varying intensity of the velocity

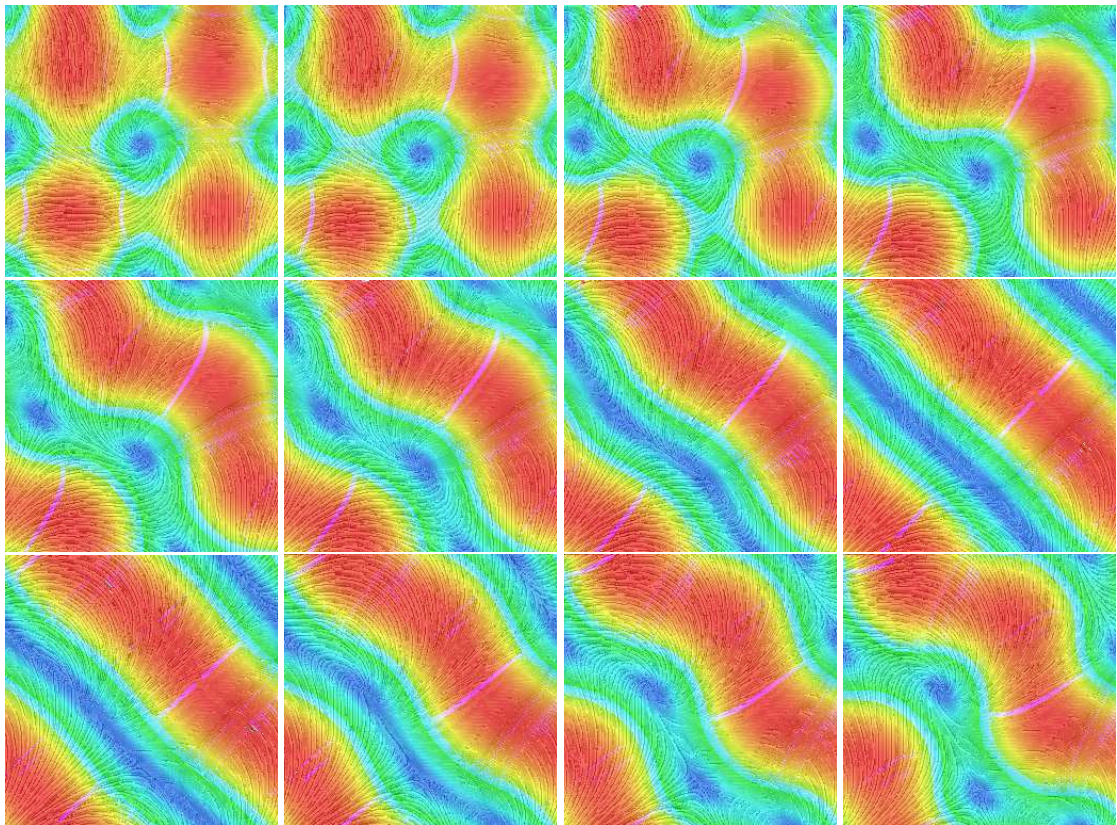


Figure 5.8: A set of generated temporal frames.

field. Since the visualization of variable vector fields is a kind of multi-variate visualization, it is possible to use multiple information mapping. Figure 5.9-left shows an application implemented to superimpose and blend layers that carry different information (see more in Chapter 6). This supports an easy understanding of the features of a particular flow. Starting from the same input data, a user may generate different frame sets, and then decide to analyze the one that better communicates the field information. Figure 5.9 illustrates how the use of transparency provides stronger or weaker evidence to the mapped features.

### Limitation and optimization

The ideas reported here are to provide a general and flexible technique for the visualization of unsteady fields, to generate an arbitrary number of frames for later off-line animation. Once the flow is structurally visualized, it is possible to observe it in motion and the user may interactively edit or analyze it modifying parameters and mapping features using masks and filters in a post-processing phase. Anyway, as the approach requires a couple of minutes to visualize  $256^2$ - and  $512^2$ -sized frames, it cannot be used at interactive rates.

### Discussion

This extension outlines a novel technique for generating continuous visualizations of unsteady and multi-variate fields. The algorithm automatically generates a set of frames, which in turn represent the structure of the flow field at several time steps. Besides providing a methodology for precise smooth visualization of dense flow fields, one of the main targets was to allow user intervention for local and global control in the visualization process. This method combines the intuitivism of icon-based methods and the benefits of geometric and feature-based methods, offering at the

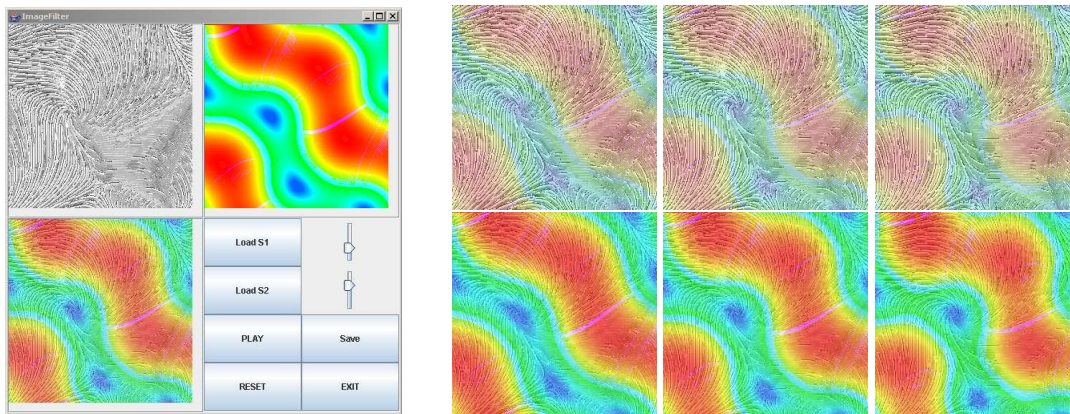


Figure 5.9: Screenshot of the blender application, the sliders are responsible for controlling the intensity of the encoding in the information masks (left). Generated frame sets with different rate of transparency (right).

same time a dense representation of the field, avoiding to miss important details and generating a smooth visualization. Another advantage of this technique with respect to others is that this approach is independent of a particular discretization of the data. Consequently, this permits an adaptive sampling for use in non-structured or non uniform grids, which is often required in industrial applications. These features enrich the visualization process with more flexibility; this is an important step to cover fundamental perception issues in visualization, and, further, it is possible to combine scientific visualization together with information visualization to get benefits in data analysis. As the pattern representing the field may be chosen arbitrarily, illustrative concepts can be taken into account as well: depending on the kind of flow to visualize, it is straightforward to select a wave- or a wind-like pattern, which could be better suited to depict the stream data set. Similarly, it is possible to design a specific pattern, in order to individually condition the resulting field appearance through traditional design principles. This can be of relevance for engineering and other scientific applications.

## 5.2 Higher order vector fields

### 5.2.1 Tensors

In scientific visualization, tensorial data are almost ubiquitous, they are useful in many medical, mechanical and physical applications. In general, a tensor field is considered as a force field that deforms an object placed inside it.

*Tensors* provide a general notation to include scalars, vectors, and tensors in general. The most common tensors are *scalar* fields, or 0–rank tensors, *vector* fields, or 1–rank tensors, and *tensor* fields, or 2–rank tensors, which can be represented in matricial notation ( $n \times n$  matrix). For simplicity, I consider here *planar* tensors. A tensor is said to be *planar* when defined in 2D space  $(x,y)$ . In this case, a 2–rank tensor is a map that associates at each point in space a  $2 \times 2$  matrix. A general 2D planar tensor can be expressed by the matrix:

$$T(x,y) = \begin{bmatrix} T_{11}(x,y) & T_{12}(x,y) \\ T_{21}(x,y) & T_{22}(x,y) \end{bmatrix} \quad (5.5)$$

that contains four unique quantities, or three for a real symmetric tensor. Common 2–rank

planar tensors are in fact *symmetric*, which means that they can be expressed as:

$$T(x,y) = \begin{bmatrix} \alpha(x,y) & \beta(x,y) \\ \beta(x,y) & \gamma(x,y) \end{bmatrix} \quad (5.6)$$

this guarantees that they are diagonalized, and the analysis leads to two eigenvalues  $\lambda_1 \geq \lambda_2$  with corresponding eigenvectors  $v_1 \perp v_2$ .

In matricial analysis, we need to calculate the eigenvalues  $\lambda$  and the corresponding eigenvectors  $v$ ; this means solving the  $2 \times 2$  *eigensystem*  $\{(v_1, \lambda_1), (v_2, \lambda_2); \lambda_1 > \lambda_2\}$ , associated to the tensor  $[T]$ , for each pixel (see more about eigenanalysis later in § 6.3.3). In particular, in the 2D case, 2 eigenvalues  $\lambda_1, \lambda_2$  can be found, which are in general a major eigenvalue  $\lambda_+$  and a minor eigenvalue  $\lambda_-$  ( $\lambda_1 > \lambda_2$ ), with corresponding principal and secondary eigenvectors  $v_1$  and  $v_2$ .

In this way, the visual representation of tensors can be treated in a vectorial way as done with the MRF-based approach, visualizing,  $\forall$  pixel  $(x,y)$  in space, the principal eigenvector, opportunely mapping its information, *e.g.* phase (corresponding to the value of its angle of orientation) and magnitude (corresponding to the value of its eigenvalue). The information related to the minor eigenvector can be mapped using color coding or further encoding cues. A possibility to completely display the tensorial information is to use iconic mapping [122, 113], for instance using ellipsoids in 3D (if the tensor is a  $3 \times 3$  matrix), the axis of which have lengths equal to the 3 eigenvalues (major, medium, minor) of the tensor (Fig. 5.10).

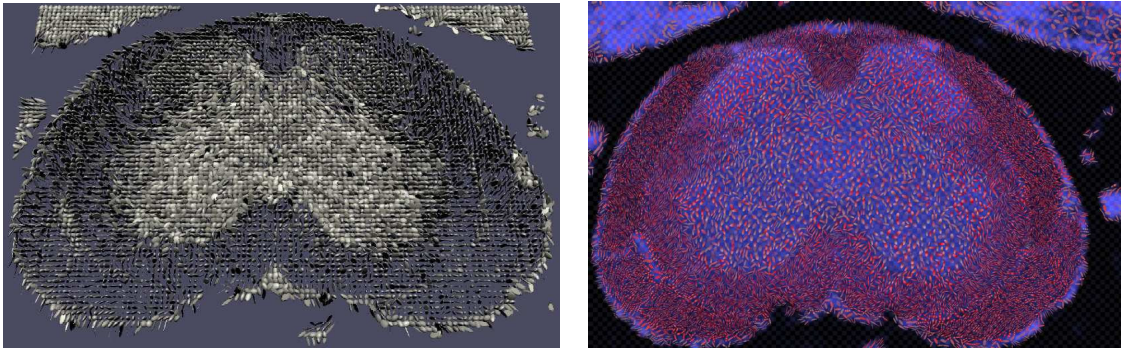


Figure 5.10: Diffusion tensor image (DTI) visualized using ellipsoids (left) and concepts from painting (right) (image courtesy of David H. Laidlaw).

In the 2D case, ellipses can serve the same function; they can be displayed at sampled positions in the output, representing a geometrical equivalence with the tensor data. Such ellipses are placed aligning their major axis (whose length is proportional to the corresponding major eigenvalue) in the direction of the principal eigenvector, and the minor axis (proportional to the corresponding minor eigenvalue) in the direction of the secondary eigenvector, which is perpendicular to the major one ( $v_1 \perp v_2$ ). Hence, eigenvectors give the ellipse orientation and eigenvalues the ellipse size (defining the size of the two major axis). Especially when dealing with *stress* and *strain* tensors<sup>3</sup>, the sign of the eigenvalues indicated regions of expansion and compression [89]. Often, approaches concentrate on the representation of the eigendirections and neglect the importance of the eigenvalues.

There are several possibilities to depict tensors. A tensor field can be decomposed into its eigenvector fields. One approach is thus to represent the major eigenvector of the tensor, as done for vector fields, and to encode the minor eigenvector, for instance using color coding or as the change in the cross section along streamlines. In this way, it is possible to simply extend the approach proposed in Chapter 3 to visualize tensors, treating them as vectors. Another possibility

<sup>3</sup>In continuum mechanics, deformation is measured by the *strain tensor*  $\varepsilon = \frac{1}{2}(J + J^t)$  where  $J$  is the velocity gradient matrix  $(\frac{\partial v_i}{\partial x_j})_{i,j}$ .

is to represent both eigenvectors and visualize them simultaneously, conferring more relevance or more significant value to the major one. In this case, more details about possible solutions to visualize multiple vectorial distributions are provided in the following section.

### 5.3 Multiple scalar and vector fields visualization

The simultaneous visualization of multiple fields on the same display finds important applications in scientific visualization. The comparison, as well as the detection of potential interference or correlation between data sets can lead to relevant observations. In science and engineering, it is often fundamental to visualize multiple scalar or vectorial distributions at the same time and at the same place. Special interest arises when different distributions are copresent and need to be conjunctly analyzed, or when two or more different processes may occur simultaneously and could create interference. Finding potential correlations or being able to compare several distributions is important to detect behaviors, to estimate possible intercorrelations and understand whether and how some variables could affect the behavior of other ones.

#### 5.3.1 Interweaving vector fields

A common way to visualize multiple distributions is to use layers, superimposing the different information. Anyway, this approach could suffer from the same problems of standard visualization, and in addition stronger occlusion could occur. The problem is that, in the absence of indications to the contrary, objects are generally perceived to lie in the background over which they are superimposed [71]. When layering two surfaces, in fact, visual confounding between the images may occur [11].

Layering can be anyway improved by means of transparency rates to separately control the influence and relevance of distinct layers (a possible solution is proposed later in § 6.6). In general, a too sparse, *glyph-based*, or too dense, *noise-based* visualization algorithm fails to provide effective representations of multiple copresent multivariate vectorial data sets. In this context, we are investigating, in cooperation with Tim Urness and Victoria Interrante, a hybrid streamline- and texture-based approach as solution to multi-field visualization.

In this section, we demonstrate how directional example-based textured visualization algorithm and streamlines are better suited for the visualization of multiple vector fields. The target [228] is to obtain a reliable, integrated understanding of the multiple fields. The goal of this research is to explore strategies for developing effective methods for the visual representation of multiple co-located vector and scalar fields to allow each field to be understood and analyzed both individually and in the context of the other. It is sometimes challenging to distinguish two co-located data sets, especially in regions where the orientations are aligned, as it could be difficult to determine which field is being represented by which texture at any given location. It is necessary to maintain continuity, as the accidental patterns of intersections between streamlines in different vector fields can lead to visual artifacts that may cause an inaccurate perception of the data.

The main idea is based on the concept of *interweaving vector fields*. This is an innovative approach to the problem of multi-fields visualization. The main benefit is given by a resulting representation of the fields, that are presented at the same level, thus avoiding occlusion and enhancing mutual correlation.

#### 5.3.2 Integrating streamline-based and texture-based visualization

Besides *layering* (§ 6.6), we investigated two different methodologies, based on *streamlines* and *textures*, for multi-fields visualization. I explain in the following both these approaches and relative two solution proposals in more detail. The methods are designed to produce oriented and controlled textures that accurately reflect the complex patterns that occur in vector field visualizations. A goal of this research is to better understand how the properties of textures can effectively



be used to represent various components of flow data. Textures have the ability to provide a richly diverse set of possibilities that allow various aspects of the underlying flow field to be visualized. 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 exhibit such as regularity, directionality, contrast, and spatial frequency.

*Streamlines* are a successful approach to visualize vector fields. Accurately choosing seed points is fundamental to avoid missing relevant areas of interest or singularities when visualizing the vectorial distribution [227]. Urness *et al.* recognized that textures can be used to enhance standard LIC. Texture mapping streamlines can supply to the lack of mapping option that is probably the major lack of LIC techniques; it is a computationally efficient method that utilizes outlining textures to depict flow orientation. An additional advantage of this techniques is that, through the mapping of textures onto streamlines, also images and fields visualization that are inherently 2d acquire a 3d sense of depth and the streamlines are given width to. Successively, standard methods are applied to control streamline density and seed placement [226, 101]. For instance sparse equally spaced streamlines are used in order to let the overlaying and underlying fields be visible. The resulting image is composed then on a pixel-by-pixel basis.

On the other side, *texture-based approaches* (and in this way an adapted version of the MRF technique) also can be successfully applied for multi-field visualization. Texture synthesis algorithms allow a variety of information encoding that is useful and particularly suited to intuitively portray complex multi-parameter data sets. Another advantage of textures and especially of the proposed MRF-based visualization approach, is that it guarantees more continuity, being pixel-based. When considering dense interweaved fields in fact, a particular attention needs to be paid to the matching of different field lines, and a best fitting search typical of the proposed method can contribute to this point. Additionally, when interweaving streamlines, it is sometimes difficult to control their density in every area of the display, and when the lines of the different fields overlap in different and non-regular way, this could lead to visual artifacts, while pre-specified samples are more easily controllable.

### Texture mapping streamlines

I briefly report here some more detailed concepts on texture mapping streamlines, which is part of our work in progress (see [205] for a pre-printed version and a complete description of this work), and which is also used for § 5.3.3.

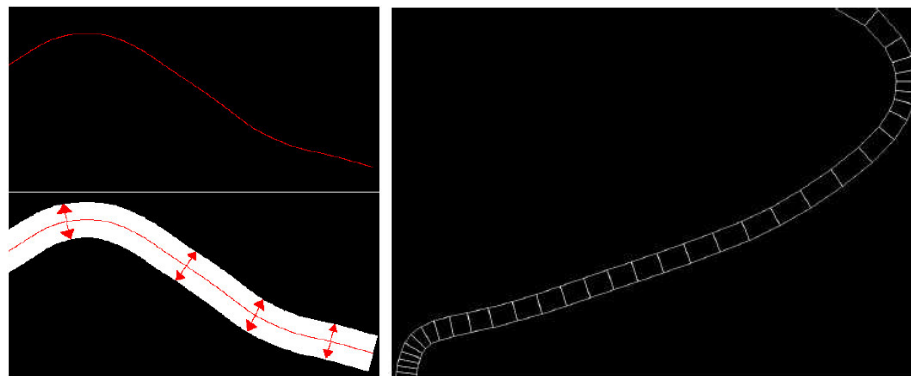


Figure 5.11: Left: a thick streamline (bottom) is constructed from a 1D streamline (top). Right: polygons are generated according to an adaptive step-size algorithm that allows for smaller polygons to be generated around areas of high curvature.

Texture mapping can be seen as an alternative to texture synthesis to display textures for the representation of a vector field. The capabilities of textures and texture attributes can be examined to represent vector fields and correlated data, such as vector magnitude.

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 [45, 233]. A *thick streamline* is constructed [205] by first calculating a 1D streamline given the vector field that defines the flow to be visualized (Fig. 5.11-left, top). 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 (Fig. 5.11-left, bottom). 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. 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 [188]. 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 (Fig. 5.11-right). Controlling streamline density allows an entire field of thick streamlines to be created and equally spaced so that applied textures can be perceived [100].



Figure 5.12: Left: using texture-mapped thick streamlines to visualize a flow field. Right: an illustration of texture outlining used to disambiguate streamline orientation.

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 (Fig. 5.12-left).

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. Fig. 5.12-right (top) shows the result of applying an isotropic texture to streamlines and the ambiguous orientation of 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 stream-

lines is displayed in Fig. 5.12-right (middle). Finally, the two images can be overlaid allowing the orientation of the flow field to be displayed (Fig. 5.12-right, bottom).

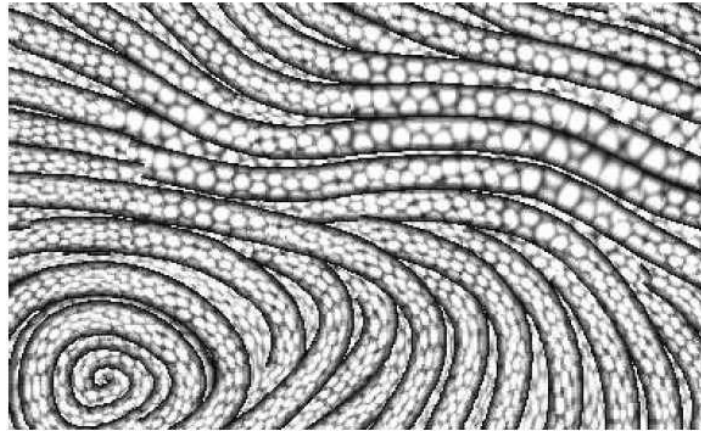


Figure 5.13: Illustration of using texture attributes to represent a scalar distribution. The scale of the texture is here related to Reynolds shear stress - a scalar field used to characterize regions where drag is generated in turbulent boundary layers.

Texture mapping streamlines gives great flexibility in changing texture parameters (Fig. 5.13) and the number of different appearances that a vector field representation can have (Fig. 5.14).

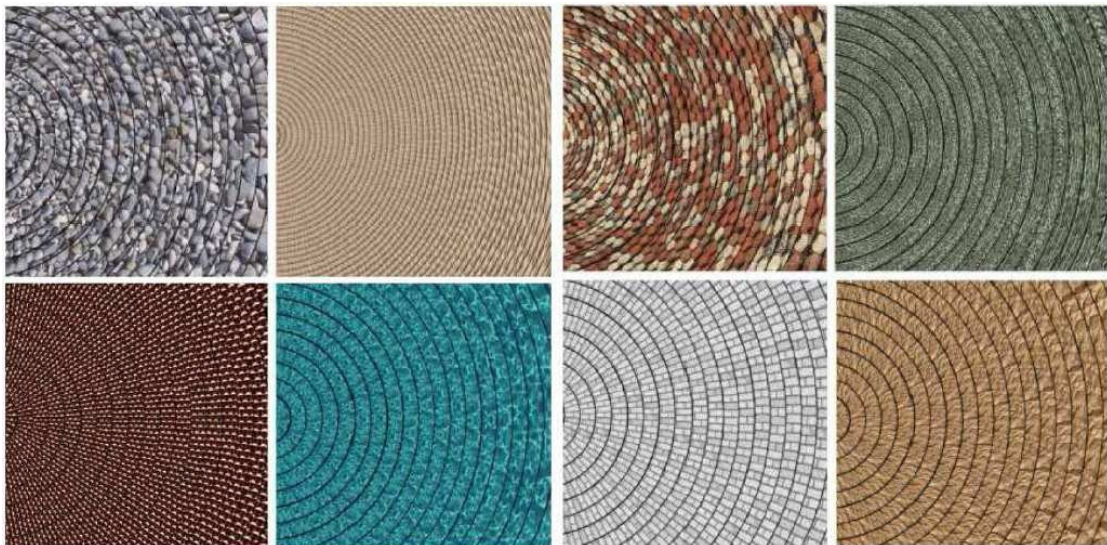


Figure 5.14: Examples of the diversity of natural textures that can be applied to a vector field. A circular flow is used demonstrate each example.

### 5.3.3 Dual vector fields

Using the theory on texture mapping streamlines described above, we attempt now visualizing vector fields interweaving them.

The term interweaving means displaying two vector fields ( $\Phi_1$  and  $\Phi_2$ ) in an inter-woven fashion, and is derived from cloth material or tissue terminology. This art to represent copresent distributions depicts two fields that are interlaced together. This has the particular advantage of simultaneously presenting two or more fields visualizing them at the same level and thus showing

them with the same relevance, while most layered approaches could give the impression of different level of importance due to the superimposition of the last layers on top of the previous ones, and occlusion, or color mixture and confusion could occur.

Texture mapping streamlines is used to *ad hoc* generate a set of input samples, where two different textures are mapped onto two co-present and misaligned sets of streamlines. The MRF-based method proposed in Chapter 3 can be effectively extended to visualize such copresent interlaced vector fields.

Basically, the MRF approach to vector field visualization works adapting a chosen directional texture pattern to the vector field to visualize. The example pattern is chosen to determine the appearance of the resulting generated output field, and it is optimally chosen to be anisotropic to better locally align its major direction to the direction of the vector field, conveying the vectorial information. In the case of dual fields visualization, every point in the output image should yield at the same time the information related to both the two vector fields, which are usually misaligned. An inter-woven representation of the fields is achieved using a collection of input samples that are *ad hoc* specified by the set of the *relative directions* between the two vector fields at the various output locations (Fig. 5.15). In this way it is possible to deal with two distinct vectorial distributions at the same time. This set of samples is produced as described above, generating interweaved streamlines of the two fields, with a respective phase drift derived by the reciprocal misalignment of the two fields at each given point. Figure 5.17 shows a subset of possible input samples and Figures 5.18, 5.19 show a couple of results.

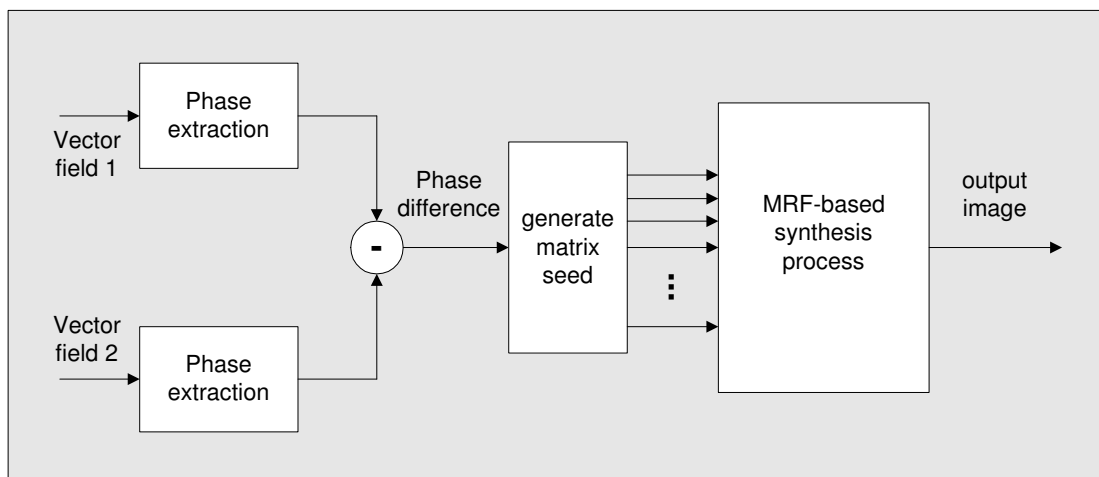


Figure 5.15: Scheme block for simultaneous dual field representation.

More precisely, for each location  $(x, y)$  in the output image  $I_{out}$ , the adapted algorithm measures the angular direction of both vector fields  $\Phi_1$  and  $\Phi_2$ . In this way, it is easy to extract (equation 3.20) the two angles of phase  $\theta_1$  and  $\theta_2$  and to calculate  $\delta\theta$  as the difference of such values ( $\delta\theta = \theta_1 - \theta_2$ ), leading to the reciprocal phase difference between the two vector fields at that point. This is done for every output point and it results in an array  $\Delta_\theta = \{\delta\theta_0, \delta\theta_1, \dots, \delta\theta_n\}$  of phase differences, where  $n$  corresponds in the most general case to the dimension of the output image, and which are used to generate or chose correspondent sample textures  $\{I_{in}\}$ .

Consequently, if the two fields at a given point are not aligned, there exists between them a rotational offset. If this angle measures for instance  $45^\circ$  in degrees, or equivalently  $\pi/4$  in radians, an example pattern is generated, which is characterized by two linear interwoven patterns: one is horizontally oriented (the reference one), and the other is oriented at  $\pi/4$  with respect to the first one. The lines characterizing the second field are interwoven with respect to those of the first ones, using a generation lines algorithm based on texture mapping described above [205]. In this way, this example pattern is used to find the best matching pixel for the texture-based synthesis algorithm at a location where the two fields have a relative orientation of  $\pi/4$ .

Function <code>synthesizeInterwoven</code>	
1	<code>for(y = 0; y &lt; H<sub>out</sub>; y++) {</code>
2	<code>  for(x = 0; x &lt; W<sub>out</sub>; x++) {</code>
3	<code>    N<sub>x,y</sub> <sub>out</sub> = calculateNeighborhood(x, y);</code>
4	<code>    <math>\theta_1</math> = calculatePhase(x, y, <math>\Phi_1</math>);</code>
5	<code>    <math>\theta_2</math> = calculatePhase(x, y, <math>\Phi_2</math>);</code>
6	<code>    <math>\delta\theta</math> = <math>\theta_1 - \theta_2</math>;</code>
7	<code>    selectInput(<math>\delta\theta</math>);</code>
8	<code>    rotateInput(<math>\theta_1</math>);</code>
9	<code>    for(j = 0; j &lt; H<sub>in</sub>; j++) {</code>
10	<code>      for(i = 0; i &lt; W<sub>in</sub>; i++) {</code>
11	<code>        N<sub>i,j</sub> <sub>in</sub> = calculateNeighborhood(i, j);</code>
12	<code>        distance<sub>i,j</sub> = compareNeighborhood(N<sub>x,y</sub> <sub>out</sub>, N<sub>i,j</sub> <sub>in</sub>);</code>
13	<code>      }</code>
14	<code>    }</code>
15	<code>    minDistance = findMinimumDistance({distance<sub>i,j</sub>});</code>
16	<code>    bestMatch = getBestPixelValue(minDistance);</code>
17	<code>    I<sub>out</sub>(x, y) = synthesizeOutputPixel(bestMatch);</code>
18	<code>  }</code>
19	<code>}</code>
20	<code>return I<sub>out</sub>;</code>

Table 5.3: Interwoven synthesis of two co-located vector fields. The pseudo-code is consistent with a simplified version of that from `synthesizePixel-a` of Table 3.1.

Similarly, a set of further samples is generated, in dependence of every different occurring phase shift between the angles of the two fields. The rest of the synthesis algorithm does not require additional modification (see also Tables 5.3, 5.4).

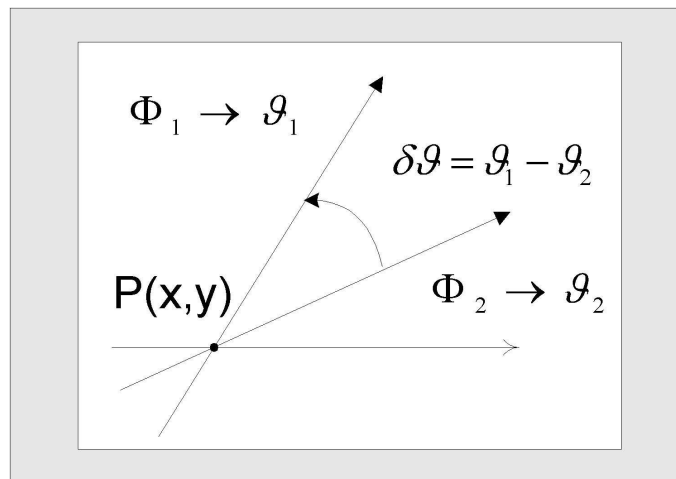


Figure 5.16: Relative phase difference between the two vector fields at a given point.

The calculations needed to measure the fields phases, to calculate the relative angle differences, to generate corresponding interweaved example textures and to store such data in corresponding arrays, are performed in an automatic way and can be completed in a pre-synthesis stage, before the visualization algorithm starts. Consequently, the method does not add complexity in the essential part of the synthesis algorithm and neither computational time. Similarly, a comprehensive set of samples can be produced, specified by a uniformly varying difference of phase and thus leading to a complete array of samples describing phase-displaced streamlines with a  $\Delta_\vartheta$  spanning in the range  $[0, \pi]$ , or  $[0, 2\pi]$  in case of samples that presents also attributes of orientation. The algorithm

Variable / method	Meaning
$\Phi_1$	vector field # 1
$\Phi_2$	vector field # 2
selectInput	selects the input sample that exhibits the given phase drift between $\Phi_1$ and $\Phi_2$

Table 5.4: Table of symbols and methods

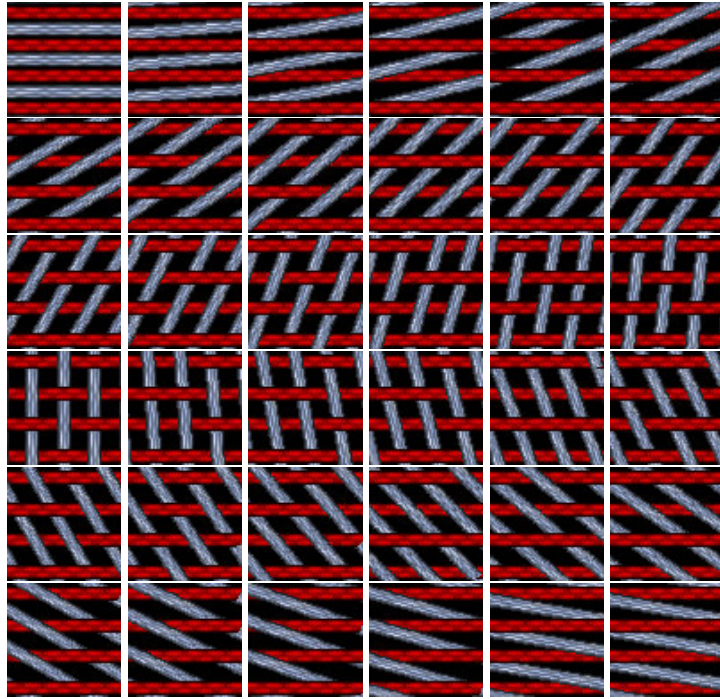


Figure 5.17: Samples generation for dual fields representation.

then goes on picking the required samples corresponding to the shift occurring between the angles of the two vector fields in the output image. In this way, the array of samples is not specially computed for a given data set and can more generally be used for any different pair of vector fields. When the fields allow it and no loss of accuracy can be noted, the array can be eventually reduced to a quantized version of phase differences.

### Further applications

The concepts used to visualize to co-located vector fields can be used for samples that exhibit more major directions. A further reason why the study of textures being characterized by two principal directions is interesting, is that studies have proven [112] that, for a large class of application cases, "two directions seem better than one" in showing shape with texture. Using textures with two principal directions as stimuli has proven how observers can make more accurate surface shape judgement. Furthermore, when texturing arbitrary double curved surfaces, textures that contain elongated elements that can be interpreted to follow both of the principal directions simultaneously can contribute to shape perception more than using textures in which the elongated elements solely in one of the two principal directions. This reserves applications in texturing surfaces, providing better perception of 3d objects form.

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

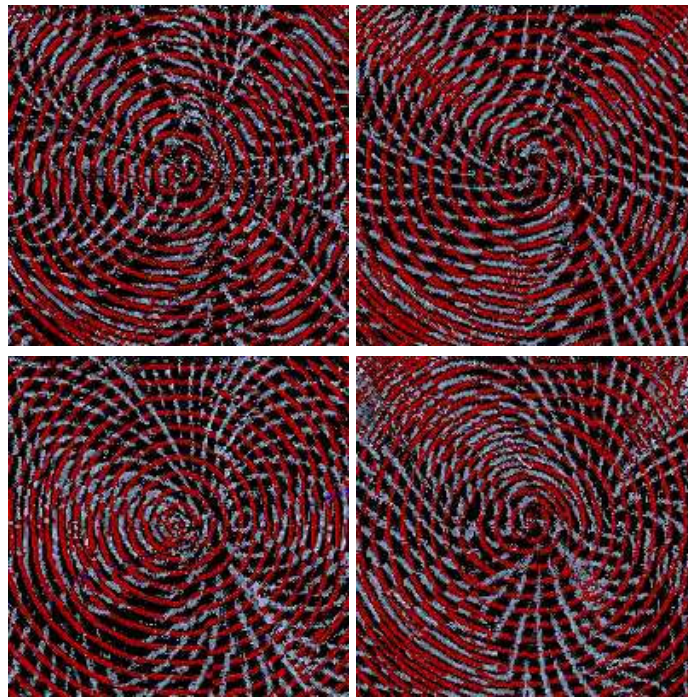


Figure 5.18: Dual fields representation.

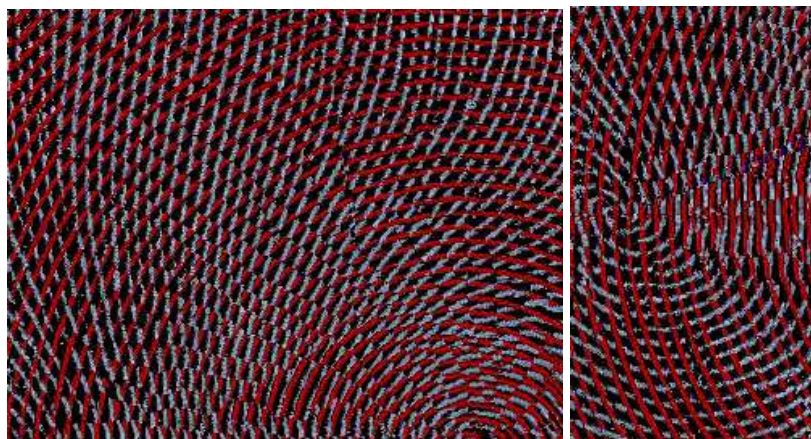


Figure 5.19: Dual fields representation.

the perceptual dimensions of textures. Furthermore, we are interested in visualizing multi-fields within the same image; for this we are also using (as introduced in § 4.2.2) a pseudo-trapezoidal weighting scheme, which can take into account two principal axis of orientation (*e.g.* for the simultaneous visualization of two vector fields) at several different phase differences.





## Chapter 6

# Extracting and encoding data content

“The world is complex, dynamic, multidimensional; the paper is static, flat”. There exists a need for representing the rich visual world of experience and measurements; in his book [221], Tufte expresses the need for designing strategies or enhancing the dimensionality and density of portrayals of information.

Appropriately encoding the information carried by the data set into effective visual representations is still an extremely actual and fundamental challenge to produce effective visualization. Adequate information mapping should facilitate the rapid, intuitive detection and analysis of the essential features and main content of data sets. The solution to this problem, *translating data to images*, is not always obvious and is often on the contrary a very demanding task; analytical data need to be transformed into clear and concise representations.

This chapter seeks to provide some description and a summary about extraction and encoding of information. Here, as more detailed compendium to concepts and explanations done in the previous chapters, a discussion on vector field parameters and an explication of relative *topological analysis* to extract relevant field attributes are presented. In particular, I describe a set of fundamental field variables, explaining how to derive them and proposing some possible solutions for expressive mapping and visual communication. This also has led to the publications of [195, 197]. A survey on important encoding possibilities is also presented. An approach based on *filtering* and *transfer operators* is then introduced, which can contribute adding a great amount of encoded information and leading to a variety of different effects and applications. Findings from perception and resulting implications validate the choices done. In particular, a generalized flexible approach is proposed to allow adaptive data encoding for user- and task-driven applications. It leads to promising preliminary benefits in visualization, and it is open to further extensions. The approach is based on *layers* constrained by *transparency rates* to vary the intensity of the attribute mapping.

### 6.1 Motivation and challenges

Achieving effective visualization is not trivial, especially when considering multivariate multi-dimensional data sets. The visualization of such data mandates the development of techniques for information mapping, which have to be intuitive to understand, in order to allow fast and easy data analysis and interpretation, at the same time avoiding confusion that could arise due to the numerous parameters and due to the potential interference deriving from the chosen representations.

Another main reason that motivates deeper investigation in information encoding is given by the fact that the visualization approach proposed in this thesis is based on textured images. Textures can be employed as powerful visual primitives; their visual characteristics can be exploited, specifying a paradigm for effective data representation. The approach is thus particularly suited for data encoding. As better described below (§ 6.4, 6.6), in fact, textures offers a very flexible way to map a broad set of variables, thanks to their rich visual dimensions.

## 6.2 Vector field parameters

### 6.2.1 User selected vs. field intrinsic features of interest

The basic algorithm explained in Chapter 3 visualizes vectorial data sets representing their lines of flow mainly in terms of *direction* and *magnitude*. Extending this approach to accept an arbitrary amount of parameters, as for the cases discussed in Chapter 5, means that further fields attributes, either *user selected* or *field intrinsic*, can be extracted and adequately mapped (Fig. 6.1).

*User selected features* are points or areas that are of interest for the user (regions of interest, ROI); they can be selected to focus on relevant attributes of the field and on zones of the domain. Besides simple definitions of information encoding discussed here to highlight particular areas of the field, in general arbitrary region selection and more complicated specifications to visualize user selected features could be implemented. Special image processing filtering operators can be here applied in a straightforward manner to the resulting output vector field, for instance specifying the coordinates of the window of interest. In this way, a specific or restricted region of interest of the field can be highlighted, or it is possible to limit the visualization to pre-defined disjoint areas.

*Intrinsic field features* inherently characterize the vector field, and are directly taken into account for visualization. It is for instance possible to simultaneously show the information related to the *magnitude* and the *phase* of the field, as well as further field attributes, such as *curl* and *divergence*, which can be encoded in the texture appearance. Further *topological information* provides a meaningful characterization of the field and can be extracted from the data set (see § 6.3). Especially the *singularities* strongly characterize the behavior of a vector field; hence, it is often interesting to visualize them, also employing a classification of critical points, attained through *eigen-analysis*. In a starting step, field parameters can be calculated, extracted or selected, and can then be mapped in a pre- or post-processing stage of the visualization process, or either in run time.

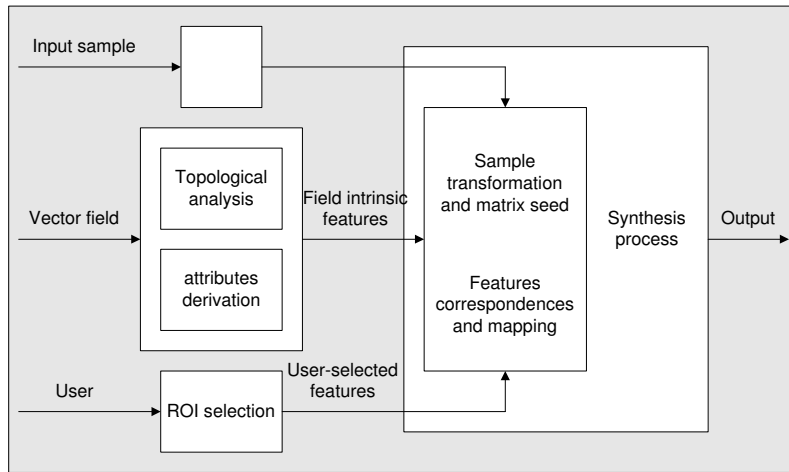


Figure 6.1: Field features extraction and selection for data mapping during the synthesis stages.

### 6.2.2 Deriving vector field attributes

Relevant vector field attributes are parameters that communicate meaningful information and, if adequately mapped, lead to an intuitive representation of the nature of the vector field. As introduced in § 3.4 (equations 3.19, 3.20), the magnitude  $A$  and phase  $\theta$  of a vector field are respectively derived as:

$$A = \|\Phi\| = \sqrt{\Phi_x^2 + \Phi_y^2} \quad \theta = \angle\Phi = \arctan \frac{\Phi_y}{\Phi_x} \quad (6.1)$$

These attributes, as explained in Chapter 3, provide a first global representation of the field; further attributes as vorticity, shear stresses, *etc.* may be potentially mapped in a similar way to contribute to a more complete field representation (see future extension in Chapter 8). The Jacobian matrix  $J$ , or matrix of the first derivatives of the vector field (see § 6.3 for a detailed introduction), can be used to compute a number of derived fields, such as the divergence, curl, helicity, acceleration, curvature.

### Divergence

In terms of fluid flow, *divergence* can be interpreted as the flux over an infinitely small loop. Divergence is a measure of the difference between the amount of flow leaving and approaching the measurement point. Hence, positive divergence is found at source points, negative divergence is found at sinks, while zero divergence occurs in case of saddle points. In the two-dimensional case, divergence is defined as:

$$\text{div}\Phi = \nabla \cdot \Phi = \frac{\partial\Phi_x}{\partial x} + \frac{\partial\Phi_y}{\partial y} \quad (6.2)$$

This derives from the general expression for the vector field in the space  $\Phi(x, y, z) = \langle \Phi_x(x, y, z), \Phi_y(x, y, z), \Phi_z(x, y, z) \rangle$ ,  $\Phi(x, y, z) = \Phi_x(x, y, z) \cdot \hat{i} + \Phi_y(x, y, z) \cdot \hat{j} + \Phi_z(x, y, z) \cdot \hat{k}$ :

$$\begin{aligned} \text{div}\Phi &= \nabla \cdot \Phi, \quad \nabla = \frac{\partial}{\partial x} \cdot \hat{i} + \frac{\partial}{\partial y} \cdot \hat{j} + \frac{\partial}{\partial z} \cdot \hat{k} \\ \text{div}\Phi &= \frac{\partial\Phi_x}{\partial x} \cdot \hat{i} + \frac{\partial\Phi_y}{\partial y} \cdot \hat{j} + \frac{\partial\Phi_z}{\partial z} \cdot \hat{k} \end{aligned} \quad (6.3)$$

### Curl

*Curl* is equivalent to the circulation in an infinitely small loop. Curl is a measure of how twisted or non-linear the flow lines are around a particular point (axis of rotation); it is a measure of the amount of flow that circles around the measurement point. The curl of a velocity field is called the *vorticity*. This parameter represents the local flow rotation, both in speed and direction.

In the plane case, the curl of a vector field is defined as:

$$\text{curl}\Phi = \nabla \times \Phi = \frac{\partial\Phi_y}{\partial x} - \frac{\partial\Phi_x}{\partial y} \quad (6.4)$$

which is derived from the more general expression of a vector field in the space:

$$\begin{aligned} \text{curl}\Phi &= \nabla \times \Phi = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ \Phi_x & \Phi_y & \Phi_z \end{vmatrix} \\ \text{curl}\Phi &= \left(\frac{\partial\Phi_z}{\partial y} - \frac{\partial\Phi_y}{\partial z}\right) \cdot \hat{i} + \left(\frac{\partial\Phi_x}{\partial z} - \frac{\partial\Phi_z}{\partial x}\right) \cdot \hat{j} + \left(\frac{\partial\Phi_y}{\partial x} - \frac{\partial\Phi_x}{\partial y}\right) \cdot \hat{k} \end{aligned} \quad (6.5)$$

Divergence and curl are two useful analytic characterizations of a vector field<sup>1</sup>. Mapping divergence and curl information strongly contributes for compelling, meaningful visualization. Curl is quite difficult to visualize along with other metrics, but it does give information that is not quite easy to see using just streamlines and hedgehogs.

<sup>1</sup>Depending on the application, a divergence-free vector field is also known as *Hamiltonian*, *solenoidal*, or *incompressible*; a curl-free vector field as *gradient*, *conservative*, or *irrotational*.

## 6.3 Topological analysis

Performing a *topological analysis* of the data set allows integrating new features in the presented approach and augments the algorithm with the benefits of feature-based visualization. Topological approaches are based on the mathematics underlying the physical phenomenon. Observing the vector field, or a frame of reference in a flow field, it is possible to determine critical points that, connected by principal lines or planes, determine the *topology* of the field. Then, *eigen-analysis* is used to classify the singularities. The extraction of features of interest or high-level information from a data set leads to selection and simplification of the set based on content (*selective visualization*), and consents to visualize the data from a problem-oriented point of view. In fact, topological features exhibit the qualitative properties of a field in a synthetic way. Topological analysis was initially inspired by the qualitative theory of *dynamical systems* [151, 9] and *differential geometry* [187]. The basic idea is that of focusing the visualization of a vector field on its singularities and spatial integral curves that connect them, partitioning the domain into subregions of uniform qualitative behavior.

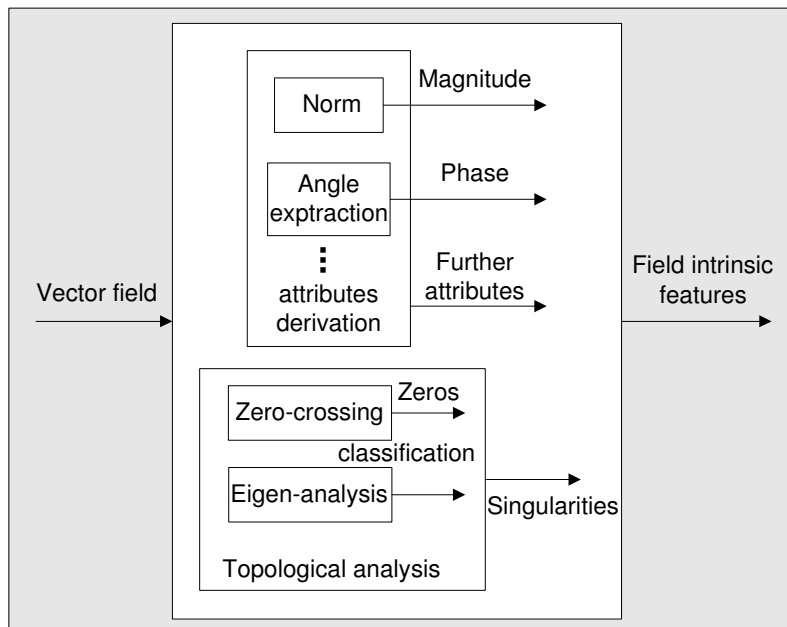


Figure 6.2: Block scheme illustrating field features (including zero crossing, eigen-analysis, feature extraction and classification).

### 6.3.1 Basic notions

#### Tangent curves

A *tangent curve* is a curve for which the tangent vector at any point along the curve is parallel to the vector field at that point. The *tangent vector* at each point along the curve is the derivative of the position vector along the curve with respect to the arc length. Given the critical points and their principal tangent curves, an observer can visually infer the shape of other tangent curves and hence the structure of the whole vector field. Consequently, topology contributes to simplify the visualization of a vector field, communicating its essential information in a condensed form.

### Phase portrait, topological skeleton

The *phase portrait* of a vector field is defined as the family of all its paths over the plane [216]. It depends on the number, type and arrangement of the critical (or equilibrium) points.

The *topological skeleton* consists of critical points and connecting separatrices.

### Separatrices, periodic orbits

A *separatrix*  $\Gamma$  is a trajectory  $\mathbf{x}(t)$  such that  $\lim_{t \rightarrow \infty} \mathbf{x}(t)$  or  $\lim_{t \rightarrow -\infty} \mathbf{x}(t)$  is a saddle. They are curves connected with saddle points along the eigenvectors, they typically start at a source or end at a sink.  $\Gamma$  is *homoclinic* if and only if  $\lim_{t \rightarrow \infty} \mathbf{x}(t) = \lim_{t \rightarrow -\infty} \mathbf{x}(t)$ . Otherwise it is *heteroclinic*. Basically, a *heteroclinic* separatrix connects a saddle with another singularity, typically an attractor or a repeller, or a periodic orbit. A *homoclinic* separatrix leaves a saddle from one of its outgoing directions and comes back to the saddle in one of its incoming directions.

A *periodic orbit*  $\Gamma$  is a trajectory  $\mathbf{x}(t)$  such that there is a positive number  $t_0$  and  $\mathbf{x}(t+t_0) = \mathbf{x}(t)$  for any  $t \in \mathbb{R}$ . the minimal positive value for  $t_0$  is the *period* of  $\Gamma$ . Furthermore, if there exists a neighborhood  $U$  of  $\Gamma$  such that  $\omega(U) = \Gamma$ , then it is an *attracting limit cycle*. Similarly, if  $\alpha(U) = \Gamma$ , then  $\Gamma$  is a *repelling limit cycle*.

### Critical points

*Critical* or *singular points* are those points at which the magnitude of the vector field (velocity vector) is zero. Such points are also called *equilibrium states*, *zeros*, *fixed* or *stationary points*, while non singular points are called *regular*. Their specificity is that they are the only locations where stream lines (the integral curves) can meet or intersect. These points can be characterized according to the behavior of nearby tangent curves.

Given a vector field, its topology, or topological description, mainly consists of its critical points. It can also be said that the critical points are the *nodes* of the *topological graph*. *Separatrices* represent the *edges* of the *topological graph*. Critical points of different order and separatrix lines strongly characterize a vector field and help determining its nature. Many of the interesting field features are associated with singularities, hence their visualization is integral to communicate the main sense and behavior of the data set. Such singularities can be automatically visualized or emphasized using the proposed approach and are here used for additional features mapping.

Geometrically, the topology of a vector field is given by the structure of its integral curves. From this point of view, thus, critical points can be classified as following:

**Center type:** Singularities are of *center type* if they are approached by no integral curve. In this case, one can find a neighborhood of the point where all paths are closed, inside one another, and contain the singular point in their interior (Fig. 6.3-a).

**Non center type:** In this case, one has at least two paths converging to (or leaving) the singular point. Such two semi-paths tend to the singular point and the *curvilinear sector* is the open region bounded by the two integral curves and the respective arc (Fig. 6.3-b). There exist three different types of curvilinear sectors:

- Hyperbolic or saddle sector, which is defined by so called  $\alpha$ - and  $\omega$ - separatrices (Fig. 6.3-c).
- Parabolic sector, also called  $\alpha$ - or  $\omega$ - parabolic sector if the integral curves respectively tend to the singular point for  $t \rightarrow -\infty$  or for  $t \rightarrow \infty$  (Fig. 6.3-d).
- Elliptic sector, if the bounding paths defining the sector are two semi-paths on the same path through the singular point. These curves form nested loops tending to the singular point for both  $t \rightarrow \infty$  and  $t \rightarrow -\infty$  (Fig. 6.3-e).

After having considered singular points from a general and geometrically motivated viewpoint, they can be analytically considered using common piecewise linear vector fields. This approach confers flexibility in modelling the topological structure for the singular points [88].

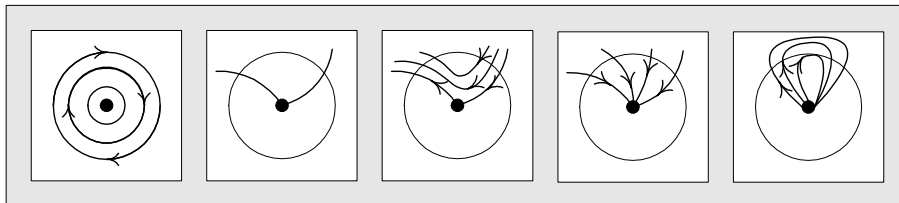


Figure 6.3: Center (a) vs. non-center singularities: general curvilinear sector (b), hyperbolic (c), parabolic (d) and elliptic (e) sectors (illustrations derived from [216]).

### 6.3.2 Piecewise linear interpolation

#### Finding critical points

Usually, classification of singularities based on the *local linearization* of the vector field is used. Note that, for consistency with literature classical notation, the vector field  $\Phi$ , as referred to in the previous chapters:

$$\begin{aligned} \Phi : E_2 &\rightarrow \mathbb{R}^2 \\ (x, y) &\rightarrow \begin{pmatrix} \Phi_x(x, y) \\ \Phi_y(x, y) \end{pmatrix} \end{aligned} \quad (6.6)$$

with the planar domain  $E_2$  being a closed and compact subset of the Euclidean space  $\mathbb{E}^2$ , is here referred to as a velocity field  $\mathbf{v}$ .

As introduced, critical points  $(x_0, y_0)$  can be identified as those points at which the components of  $\mathbf{v}$  all simultaneously vanish:

$$(x_0, y_0) \in E_2 : \mathbf{v}(x_0, y_0) = (0, 0)^T = \mathbf{0} \quad (6.7)$$

and  $\mathbf{v}(x, y) \neq \mathbf{0} \forall (x, y) \neq (x_0, y_0)$  in a certain neighborhood of  $(x_0, y_0)$ .

#### Linear interpolation

The field in the neighborhood of each critical point is approximated by the *Taylor series expansion*<sup>2</sup> of  $\mathbf{v}$ , which, about a point  $\mathbf{x}_0$ , is expressed as follows:

$$v_i(\mathbf{x}) = v_i(\mathbf{x}_0) + \frac{\partial v_i(\mathbf{x}_0)}{\partial x_j} (\mathbf{x}_j - \mathbf{x}_{0j}) + \mathcal{O}(\Delta \mathbf{x}_k \Delta \mathbf{x}_l) \quad (6.8)$$

assuming  $\mathbf{v}$  to be sufficiently smooth and differentiable for the Taylor expansion to exist<sup>3</sup>. In the equation, and considering a first order expansion, the term  $\frac{\partial v_i(\mathbf{x}_0)}{\partial x_j} (\mathbf{x}_j - \mathbf{x}_{0j})$  is the first derivative

<sup>2</sup>The Taylor expansion of an infinitely differentiable real (or complex) function  $f$  defined in an open interval  $(a - r, a + r)$  is the power series:  $T(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x - a)^n$ , where  $n!$  is the *factorial* of  $n$ , and  $f^{(n)}(a)$  is the  $n$ -th derivative of the function  $f$  evaluated at the point  $a$ .

<sup>3</sup>A function is differentiable if it is possible to approximate it, in the neighborhood of each point, using a linear function, with rest of infinitesimal order higher than the first in the distance of the point. The differential is the linear approximating operator, which is expressed by a matrix called the Jacobian matrix.

$v'_i(x_0)$  of the function  $v_i$  evaluated in  $x_0$ ; the terms of order greater than the first term of the series expansion are  $\mathcal{O}(\Delta x_k \Delta x_l)$  and are disregarded. At a critical point, the first term of the expansion vanishes by definition, and considering only the second term, each equation has two terms, one for each coordinate direction. Consequently, the coefficients

$$(\nabla \mathbf{v})_{ij} = \frac{\partial v_i}{\partial x_j} \quad (6.9)$$

of the first non-zero term of such Taylor expansion around a critical point build, in the two-dimensional case, a  $2 \times 2$  matrix  $\nabla \mathbf{v}$  called *Jacobian matrix*. Around the critical point, the eigenvalues and eigenvectors of this matrix determine the local behavior of  $\mathbf{v}$ . In matrix notation, this corresponds to consider an affine linear vector field

$$\mathbf{v}(\mathbf{x}) = A(\mathbf{x}) + b \quad (6.10)$$

where

$$\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix} \quad A = \begin{pmatrix} \alpha_x & \beta_x \\ \alpha_y & \beta_y \end{pmatrix} \quad b = \begin{pmatrix} \gamma_x \\ \gamma_y \end{pmatrix} \quad (6.11)$$

consequently, if  $\mathbf{v}$  has a zero, one takes its location as new coordinate origin and thus, as simplification of the vector field, considers the linear field

$$\mathbf{v}'(\mathbf{x}) = J(\mathbf{x}) \quad (6.12)$$

so, an affine linear vector field is uniquely determined by its Jacobian at the location of its possible zeros [216].

### 6.3.3 Eigen-analysis

Once the singularities have been found, it is possible to classify them performing an *eigen-analysis* and, specifically, observing the sign of the eigenvalues of the *Jacobian matrix*  $J$ . The determinant of the *Jacobian matrix* is called *the Jacobian* of the vector field<sup>4</sup>. I illustrate here the necessary steps. In *eigenanalysis*, an *eigenvalue* of a matrix  $J$  is a (possibly complex) scalar  $\lambda$  which solves the *eigenvector equation*:

$$J\mathbf{x} = \lambda\mathbf{x} \quad (6.13)$$

The corresponding non-zero vector  $\mathbf{x}$  is called *eigenvector* of  $J$ . The eigenvectors and eigenvalues of a Jacobian matrix determine the local behavior of  $\mathbf{v}$ ; they indicate the direction of tangent curves of the flow used for a topological description of the field. In general, the eigenvectors of  $\nabla \mathbf{v}$  span the invariant manifolds of the linearized field around a critical point. Curves integrated from initial points on the eigenvectors at a small distance from a critical point connect with other critical points (or the boundary) to complete the topology.

Hence, observing equation 6.9, and calling the vector field  $\Phi$  again, the Jacobian matrix, or *gradient matrix*<sup>5</sup>, for the vector field  $\Phi$ , with respect to the position at a given critical point  $(x_0, y_0)$ ,

<sup>4</sup>The Jacobian  $J$  can be also decomposed for analysis into its symmetric and antisymmetric components. Another possibility of analyzing the Jacobian is to transform  $J$  into the local *Frenet* frame at some points of the trajectory [44].

<sup>5</sup> $\nabla$  is a *gradient*, that means that for a scalar field  $p$ , in two dimensions, it is the vector of its partial derivatives:  $grad p = \nabla p = [\partial p / \partial x \quad \partial p / \partial y]$ , while for a vector  $\Phi$ , it is the  $2 \times 2$  Jacobian matrix of its first derivatives as shown in equation 6.14.

is given by:

$$J_{\Phi}(x, y)|_{x_0, y_0} = \left[ \begin{array}{c} \nabla\Phi_x(x, y) \\ \nabla\Phi_y(x, y) \end{array} \right] \Big|_{x_0, y_0} = \left[ \begin{array}{cc} \frac{\partial\Phi_x}{\partial x} & \frac{\partial\Phi_x}{\partial y} \\ \frac{\partial\Phi_y}{\partial x} & \frac{\partial\Phi_y}{\partial y} \end{array} \right] \Big|_{x_0, y_0} \quad (6.14)$$

Starting from equation 6.13, the eigenvalues of the Jacobian matrix can be calculated by solving the following equation:

$$\det(J - \lambda I) = |J - \lambda I| = 0 \quad (6.15)$$

where  $I$  is a unit matrix (*identity matrix*) and  $\det()$  represents a determinant:

$$\left| \begin{array}{cc} \frac{\partial\Phi_x(x_0, y_0)}{\partial x} - \lambda & \frac{\partial\Phi_x(x_0, y_0)}{\partial y} \\ \frac{\partial\Phi_y(x_0, y_0)}{\partial x} & \frac{\partial\Phi_y(x_0, y_0)}{\partial y} - \lambda \end{array} \right| = 0 \quad (6.16)$$

The equation is a quadric (in two dimensions) equation for  $\lambda$ . Using linear algebra, the solution of the *eigensystem* is a scalar  $\lambda$ , possibly complex, which may assume two forms:

- Two real eigenvalues:  $\lambda_1 = R_1, \lambda_2 = R_2$
- A pair of complex conjugate eigenvalues:  $\lambda_1 = R_1 + i \cdot I_1, \lambda_2 = R_2 + i \cdot I_2$ , with  $R_2 = R_1, I_2 = -I_1$

### Critical points classification

In practice, thus, the flow in the neighborhood of critical points is characterized by eigenanalysis of the velocity gradient tensor, or Jacobian of the vector field. The eigenvectors indicate the directions in which the flow approaches or leaves the critical point. The critical points are classified according to the *sign* of the real  $R$  and imaginary  $I$  parts of the eigenvalues  $\lambda$ . Positive eigenvalues indicate that  $\Phi$  is directed away from the critical point (*repelling eigendirection*) and negative values the opposite (*attracting eigendirection*): they correspond to velocities towards the critical points. A complex conjugate pair of eigenvalues indicate that  $\Phi$  spirals-in or -out, depending on the sign of the real part of the eigenvalues.

Critical points can be therefore classified as *nodis*, *foci* or *saddles* on the basis of the eigenvalues of  $\nabla\Phi$ . Nodes and foci can be further classified as attracting or repelling (sources and sinks). Saddles have one positive and one negative eigenvalue: near a saddle,  $\Phi$  approaches the critical point along negative eigendirections and recedes along positive eigendirections. A saddle has thus two incoming and two outgoing trajectories. Around foci,  $\Phi$  spirals toward or away from the focus. The eigenvalues are a complex conjugant pair with a positive real part indicating a repellor and a negative part indicating an attractor. The magnitude of the imaginary part indicates the strength of the spiraling motion. If the real part is zero, concentric ellipses occur.

Hence, using such categorization, it is straightforward to treat different classes of singularities in separate manners. Furthermore, as mentioned above, we can for instance enhance critical points via an embossing filter, and areas of interest around singularities via a brightening mask with increasingly varying intensity toward the center of the ROI. In this way, feature extraction is easy and a topological analysis is intuitive. Using such mapping it is also possible to observe how those features vary and move over time.

Here, I summarize the classification of the different critical points (Fig. 6.4) in a schematic way:

- Repelling Focus or source:  $R_1, R_2 > 0; I_1, I_2 << 0$ . All eigenvalues have positive real parts. The zero is called source, because any integral curve tend to it for  $t \rightarrow -\infty$  (Fig. 6.5).
- Node source:  $J$  is diagonalizable and its eigenvalues are different.



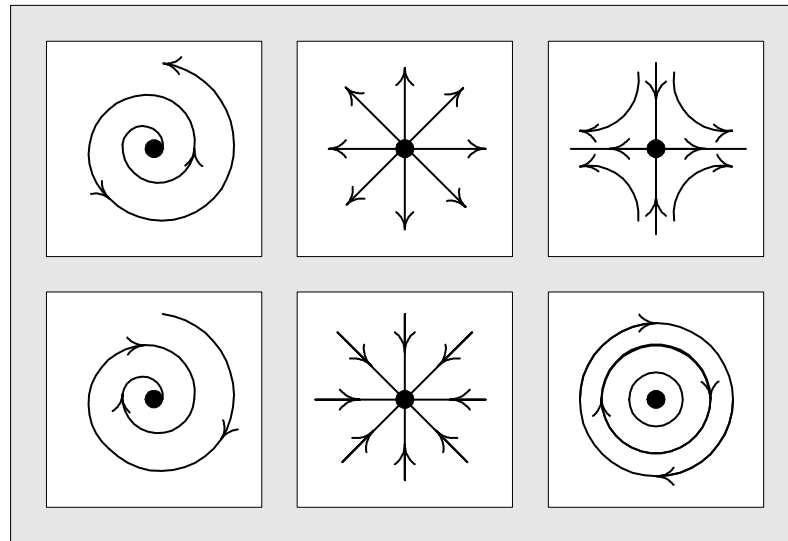


Figure 6.4: Critical point classification: repelling focus, repelling node, saddle point, center, attracting node, attracting focus (clockwise from top-left corner).

- Focus source:  $J$  is diagonalizable and its eigenvalues are equal.
- Improper node source:  $J$  is not diagonalizable but has one real positive eigenvalue.
- Spiral source:  $J$  has two complex conjugate eigenvalues with positive real parts.

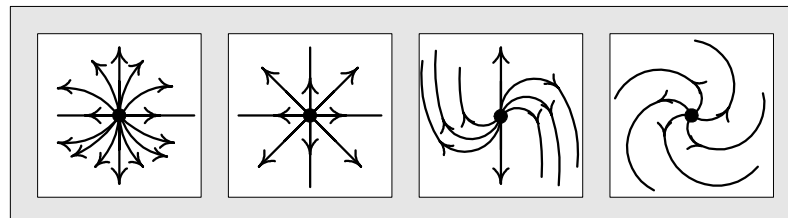


Figure 6.5: Repelling critical points: node source, focus source, improper node source, spiral source.

- Repelling star or repelling node:  $R_1, R_2 > 0$ ;  $I_1, I_2 = 0$ . All eigenvalues are real with positive real parts.
- Saddle Point:  $R_1 * R_2 < 0$ ;  $I_1, I_2 = 0$ .  $J$  has real eigenvalues of opposite signs.
- Attracting Focus or sink:  $R_1, R_2 < 0$ ;  $I_1, I_2 <> 0$ . All eigenvalues have negative real parts. The zero is called sink, because any integral curve tend to it for  $t \rightarrow \infty$  (Fig. 6.6).
  - Node sink:  $J$  is diagonalizable and its eigenvalues are different.
  - Focus sink:  $J$  is diagonalizable and its eigenvalues are equal.
  - Improper node sink:  $J$  is not diagonalizable but has one real negative eigenvalue.
  - Spiral sink:  $J$  has two complex conjugate eigenvalues with negative real parts.
- Attracting star or attracting node:  $R_1, R_2 < 0$ ;  $I_1, I_2 = 0$ . All eigenvalues are real with negative real parts.

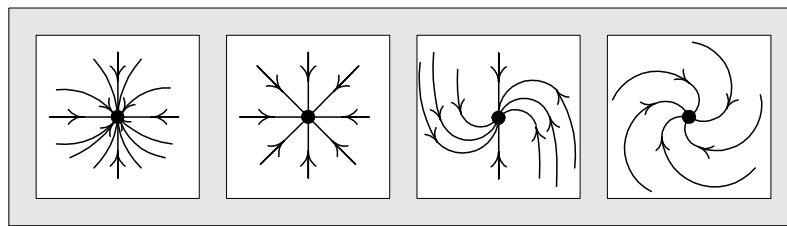


Figure 6.6: Attracting critical points: node sink, focus sink, improper node sink, spiral sink.

- Center:  $R_1, R_2 = 0$ ;  $I_1, I_2 <> 0$ .  $J$  has pure imaginary eigenvalues.

Thus, if the real part of an eigenvalue is less than zero, convergence occurs along the corresponding eigenvector; if it is greater than zero, divergence occurs. If the imaginary part of an eigenvalue is not equal to zero additionally a rotational movement around the fixed point is given.

Depending on their order, singularities can be further classified as follows:

### First order singularities

First order singularities are also called *linear*, *generic* or *hyperbolic* critical points, *i.e.* they are structurally stable and easy to analyze. In linear vector fields, it is only possible to find *simple* or *first order* singularities. A critical point is a first order critical point *iff* the *Jacobian* does not vanish in that point, otherwise the critical point is of higher order. In order to detect more complex topological features, it is necessary to perform further tests.

### Higher order singularities

*Non-hyperbolic* or *degenerate* critical points occur when the matrix  $J$  has not full rank. In this case, the Jacobian determinant is zero: the real part of one or both eigenvalues is equal to zero, the eigenvectors are not uniquely defined, and more complex topological patterns (than those illustrated above) around the critical point can appear. Other exceptional cases occur when defective matrices are encountered and hence eigenvectors coincide. These degenerate cases, though unstable, do occur in flows with imposed constraints such as symmetry or incompressibility.

One example is represented by the *dipole* (Fig. 6.7-left):

$$v(x, y) = (x^2 - y^2, 2xy) \quad (6.17)$$

Another example is given by the so called *monkey saddle* (Fig. 6.7-right):

$$v(x, y) = (x^2 - y^2, -2xy) \quad (6.18)$$

the name derives from the observation that a saddle for a monkey, seen as surface, requires three depressions: two for the legs, and one for the tail. The term horse saddle is used, by contrast with monkey saddle, to designate a saddle point that is a minimax, that is to say a local minimum or maximum depending on the intersecting plane used. The monkey saddle will have a local maximum along certain planes, but it won't be a local minimum along others, just a point of inflection.

### Tensor topology

For the popular case of two-dimensional symmetric second order tensor fields (§ 5.2.1),  $2 \times 2$  symmetric matrices have to be considered. As explained in § 5.2.1, a real two-dimensional symmetric matrix  $M$  has always two real eigenvalues  $\lambda_1 \leq \lambda_2$  with associated orthogonal eigenvectors.

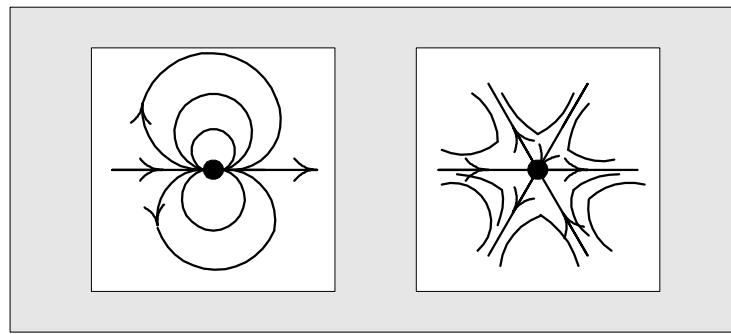


Figure 6.7: Higher order singularities: dipole and monkey saddle.

In this way, a major eigenvector corresponding to the major eigenvalue can be identified for the visualization. In points where the two eigenvalues are equal, the eigenvectors cannot be uniquely determined; this corresponds to a zero value of the deviator and singularities, *trisectors or wedge points*, exist [216].

## 6.4 Augmenting the MRF algorithm with data encoding

As introduced in the survey of § 2.3, some of the methods referred in the state-of-the-art-report cannot be generalized or fail in visualizing some scientific data sets appropriately; anyway they often result to be complementary one to another, since some of them are specially designed to solve a given problem case or to enhance a specific aspect of the data. Direct visualization is particularly suited to rapidly convey a global impression of the data abstracting in part the information and simplifying it. LIC-based methods, on the other hand, densely represent the data information, but they offer less opportunity for information encoding. The methodology I propose is mainly based on the adapted MRF algorithm, but it also can integrate and combine some features from existing visualization techniques, taking advantage of their complementarity. Vector fields attributes, as those described above in § 6.2, need to be appropriately mapped using effective visual representation, and I introduce some solutions in the remainder of this section (Fig. 6.8). In the following, I start with a discussion on texture properties and visual dimensions. Defining a sort of *texture space* based on the rich *texture visual dimensionality* provides a powerful and flexible encoding instrument for the specification of features *vs.* correspondences of representations. Such texture properties also motivate the use of textures in scientific visualization.

### 6.4.1 Texture visual dimensions and texture space

#### Texture as visual paradigm: characteristic visual attributes

When using visual properties that are preattentively processed (see § 2.1.4), viewers do not have to focus their attention on a specific region in an image to determine whether elements with the given property are present or not. Such features result for the human eye of better and faster perception. Besides improving in general the effectiveness of a visualization, the use of preattentive features results of particular interest when it is required to perform a side-by-side examination of similar images (data sets comparison), for instance when visualizing slice data and when observing variations side-by-side [81]. Commonly used preattentive features include hue, curvature, size, intensity, orientation, length, motion, and depth of field. Textures (see *Texture parameter space* following subsection) are characterized by a variety of visual dimensions, and the most prominent of them are indeed preattentive features. They can incorporate all the significant attributes described above and use this potentiality to map relevant information extracted from the field onto such features. For this reason, textures are particularly tailored for information encoding and are a

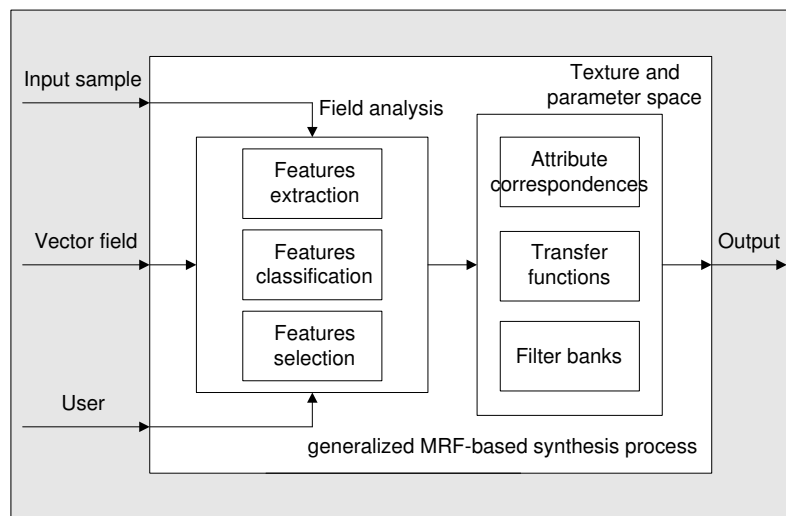


Figure 6.8: Generalized block scheme: augmenting the MRF-based approach.

valid powerful instrument as visual primitives for use in visualization applications. Textures offer strong flexibility and are for this reason a valid tool to depict information and a good solution for scientific visualization. They can be hand-designed, allowing stylistic and illustration-based visualization, which is of particular interest, *e.g.* for educational purposes.

### Parametrization

Textures can be arbitrarily complex and it can take ten to twenty parameters to define a texture with a reasonably complex set of texture elements and color components [11]. For a basic overlap of two surfaces, in [11] each texture is made up of four layers; a background and three layers of features are composited over the background. Two of the feature layers are lines, and the third layer consists of dots. Lines provide the ability to create crosshatching and linear structure, while dots provide the ability to create a high frequency, mottled look. This is a quite complete parametrization of the texture space, allowing the generation of a number of different textures. As also recognized by Interrante [92], giving overlapping surfaces partially transparent textures can help to define and distinguish them.

### Texture parameter space

In order to create or construct a perceptually meaningful multidimensional texture space that can be indexed in the same fashion as a color space, it is necessary to proceed from a rigorous and experimentally supported understanding of how human observers perceive and interpret texture patterns, under the conditions in which it is intended for these patterns to ultimately be viewed [91]. A number of researchers have conducted studies to try to elucidate the most significant perceptual dimensions of textures [161]. In the most experiments, using simple texture samples as *stimuli*, subjects are generally asked to cluster them into groups, estimating the salience of differences in various properties of texture patterns. Textures comprise of several compositional visual elements such as contrast, lighting, directionality. There appears to be general agreement that a small number (about three) of characteristic dimensions seem sufficient to describe most of the structure underlying this classification. Then, classified patterns may remain good candidates for further similarity grouping, according to other characteristic dimensions. The interpretation of the dimensions varies from study to study [91], but most often, as discussed also in the survey (§ 2.2.3), it includes aspects of the following:

- periodic (consisting of repeated discrete elements) vs. non-periodic

- strongly directional vs. rotationally invariant
- coarse vs. fine (spatial frequency of the dominant detail)
- regular (deterministic) vs. random
- high contrast vs. low contrast
- homogeneous (spatially invariant) vs. heterogeneous

Clearly, there is some overlap in these categorizations. Also, it is not evident that we can determine an orthogonal basis that encompasses all members of the texture pattern set [91]. However, the apparent low perceptual dimensionality of the space and the strong agreement between the studies bodes well for application of textures to scientific visualization.

On the one hand, Ware and Knight [237] considered *orientation, size, and contrast* as the primary orderable dimensions of texture. As pre-attentive features of individual elements, size, contrast, and orientation differences are undisputedly important in facilitating pop-out (§ 2.1.4). In addition to determining which textures tend to cluster, it is important for creating a perceptually linear texture space to quantify the perceptual distances between individual texture patterns. It is also necessary to estimate the extent of a given texture characteristic dimension and the magnitude of the perceived distance due to the differences along each of the feature dimensions<sup>6</sup>. Studies using individual element arrays have found that salience (or the tendency to *pop-out*) tends to increase when the targets are characterized by redundant, unique properties such as *luminance* and *hue* or *color* and *orientation* [145]. Similarly, the salience of the target tends to decrease as the heterogeneity of the distracter elements increases, even when the heterogeneity occurs along a different perceptual dimension. It may additionally be of interest to determine how many different texture types people can simultaneously discriminate, using a methodology similar to what Healey employed for studying color [80]. In [11], the color variables *hue, saturation* and *value* are parameters of interest, and in all cases the hue and saturation variables have more variation than value. On the other hand, often variables can be simply used to change the visualization to aesthetic taste. An important role is also given by the *four opacities* in covering a surface: the size, randomness, separation and probability of being drawn of each of the features. Bertin [22] classifies *size, value, texture, color, orientation, shape*, as "retinal properties", according to their value of organization<sup>7</sup>. Recent evaluation studies [4] also use the following design factors to map data: data resolution<sup>8</sup>, feature resolution<sup>9</sup>, linearity<sup>10</sup>, visual bandwidth<sup>11</sup>, dominance<sup>12</sup>, time to read<sup>13</sup> and intuitive association<sup>14</sup>.

Although no standard definition actually exists to define in a unique way the visual dimensions that characterize textures, most scientists agree in recognizing particular importance to the features of *orientation, scale* and *contrast*, as well as *periodicity, directionality* and *randomness*. Thus,

---

<sup>6</sup>One possible approach is to estimate the magnitude of the change required to enable a just noticeable difference between images along individually selected texture dimensions such as scale, contrast, orientation, regularity, and so on using psychophysical methods. Another possibility, which may be more appropriate for judging the kinds of differences that cannot be easily brought down to threshold levels, is to measure the pre-attentive discriminability or salience of differences in features of individual texture patterns randomly embedded in homogeneous and heterogeneous fields of distracters. The objective in this case is to determine how large of a difference is required to allow the effortless identification of the *odd man out*, masked stimulus presentations [91].

<sup>7</sup>whether they could be used to represent quantitative, qualitative, or ordered information, and the number of steps they could take.

<sup>8</sup>the number of different levels of a data variable that can be distinguished by a viewer.

<sup>9</sup>the minimum spatial feature size that can be reliably represented.

<sup>10</sup>the perceptual linearity of the mapping from the data value to visual property.

<sup>11</sup>the percentage of a method that can be covered when combined with other methods, but still remains readable.

<sup>12</sup>the forcefulness or punchiness of the data mapping.

<sup>13</sup>average time to comprehend the data.

<sup>14</sup>to measure whether any associative readings of a method might interfere with the desired numerical reading.

textures are well suited to incorporate many factors and use them for the data mapping. While encoding information, it is crucial to consider potential interference, as this could strengthen or confuse the information in the visualized image. Furthermore, understanding how distinguishable are different features is of particular importance to better design the visualization.

**Creating a texture palette:** in the approach proposed in this thesis, a matrix consisting of texture samples to be used as seeds for scientific visualization has been introduced (§ 3.2.3) and proposal for extension and improvement have been proposed (§ 4.5.2). In the previous chapters, such matrix bases on a few simple transformations over an original input sample. More in general, in order to create a possibly consistent and complete texture palette to span the defined texture space, it is possible to begin with a collection of well-chosen input images, which are potentially uncorrelated, and which exhibit characteristic visual dimensions such as those listed above (Fig. 6.9 shows an example, derived from the Brodatz textures). Intermediate textures can be derived through a sort of *texture interpolation* [91]. They can be either adequately manually designed or obtained through blending and morphing operators [137], deterministically filling the space at equal perceptual distances along each of the dimensions to complete the space. Some of the difficulties in constructing this palette are that there may be interaction among certain dimensions (such as contrast and spatial frequency), which can lead to distracting or confusing effects; moreover some texture type mixtures may not be meaningful, since the co-presence of diverse visual dimensions may produce effects, where two distinct pieces of information are no more separately recognizable and discernable. This could inhibit the perception, and for this reason I use in this thesis a simple texture space, making use of a limited set of parameters to parameterize the texture appearance, as suggested by various vision studies.

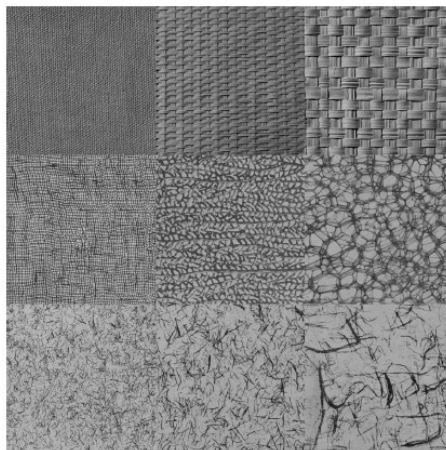


Figure 6.9: A small potential texture palette. Scale increases along the horizontal axis, regularity increases along the vertical axis, and intensity increases along the left-to-right descending diagonal.

Interrante [91] uses texture images for information visualization (and in particular to represent the distribution of agricultural products within countries of the USA). With respect to the applications to scientific visualization and the representation of vectorial data sets analyzed here, information visualization applications do not need to pay special attention to the issue of continuity and smoothness in the definition of a texture space. Nevertheless, her considerations about feasibility issues are common to my approach: some of her open questions are:

- What does a reasonable partitioning of a natural texture space look like?
- Would it be feasible to try to choose exemplars at the endpoints of each perceptually relevant texture dimension, characterize them statistically, then interpolate to obtain intermediate textures that fill out the space?

- To what extent do we need to guarantee that different textures will meld continuously into each other at the transitions between level set regions?
- How can we most effectively combine color with texture to convey yet more information in a meaningful way?

Regarding these open questions, in this thesis, and specifically for the case of vector field visualization, I partition the texture space using a set of dimensions as motivated in § 2.1.4 and illustrated for instance in Fig. 6.9. For the applications shown in this work, such dimensions, together with the relative correspondences between visual representations and vector field parameters, proved to produce good and perceptually meaningful results for an intuitive visualization. The interpolation between different images in the texture space and the continuity between different adjacent samples was driven depending on the particular dimension under consideration. For very significant dimensions, such as angle of phase, a particularly precise set of intermediate images (as described more extensively in the *quantization* subsection of § 3.3.2) must exist to guarantee smoothness in the vector field visualization. Other dimensions allow *vice versa* different quantization, *i.e.* larger distances or steps in the interpolation. In general, in order to guarantee continuity in the results, in most cases I generate the sample matrix that spans the texture space by means of cascades of filters that iteratively and progressively operate transformations between the samples.

The visualization method must balance a trade-off between the need of representing many variables simultaneously and the consideration about the human optical limitations to produce effective visualization, allowing the user to visually separate relevant information and different features in the data sets. Scientists may want to maximize the amount of comprehensible data presented in one visualization, examining a vector field and several derived quantities.

#### 6.4.2 Visual representations and correspondences of features

The amount of *variables* (§ 6.2), which characterize the multivariate field, represents the information that needs to be visualized, in order to provide an effective representation of the vector field and in order to convey its behavior. This information can be visually displayed taking advantage of the various *visual dimensions* of the texture examples, and in addition also using *iconic mapping* over the texture seeds. Such way to encode information contributes to a more holistic understanding of the data content in the visualization output.

A critical issue is to specify appropriate meaningful mapping functions, establishing a relationship among field features, either field-intrinsic or user-selected, and adequate visual representations. In § 3.4, I specify a correspondence between the phase and the magnitude of the vector field with, respectively, a rotation and a scaling operator. Such attributes and transformations are tightly coupled and result thus in intuitive visual representations. Similarly (§ 6.2), further field attributes can be associated to further transformation operators. These operators are in general *transfer functions*, which are responsible of modifying the starting sample through an action and by a factor corresponding to the desired transformation and the considered field attribute value.

Additionally, as done in direct visualization approaches, simple visual representations as *arrow plots*, or their extension to *tensor probes*<sup>15</sup>, can be used. The same visual representation could serve more than one graphical purpose (graphical element). *Ad hoc* designed *glyphs*, for instance, can simultaneously convey different pieces of information, although they use a little portion of the image; they describe a graphical entity whose shape and appearance is modified by mapping data values to some of its graphical attributes. Color can be used to report additional variables. Such multifunctional graphical elements, if designed with care, can effectively display complex, multivariate data. An early example of multi-varied visualization through multiple-symbol format

<sup>15</sup>A vector field probe can indicate, beyond direction and magnitude, properties obtained from a first order field expansion that can be decomposed into characteristic components such as curvature and torsion, which are representable by a set of geometric symbols.

and geometric coding is represented by the *Chernoff faces* [37]. These are popular in information visualization and are also applied to other types of graphs under the multiple-symbol paradigm [7]. In a Chernoff face, multiple variables are represented by different facial features. However, as in direct visualization, the display could become too crowded and tend to overload the viewer, also considering that mapping multiple variables in a graphical element can require space and several pixels, avoiding a continuous visualization.

In general, data encoding can take effect on the sample seed, both changing its appearance (*e.g.* color coding: feasible to represent scalar distributions) and its structure (*e.g.* adding glyphs, shearing, resizing: best suited to represent special regions and features). As mentioned in § 2.1, issues from perception, cognition and psychology are addressed when specifying correspondences between the relevant field attributes and the visual representations. From such studies it can be derived that humans easily associate a change of resolution in the structure of a pattern to a perceived change of the amount of a given parameter. Similarly, rotating a directional pattern intuitively conveys a change of direction. Again, considering established studies in human vision, special color scala are used to map the range of variation of the given field parameters. Below in § 6.5, similar concepts are considered to encode for instance uncertainty, areas of interest, elevation, temperature, etc. respectively using blurring or texture irregularity, brightening and contrast, embossing, rainbow colorscale, etc. Although these correspondences have been chosen under perceptual considerations, they are interchangeable and different effects can be combined together.

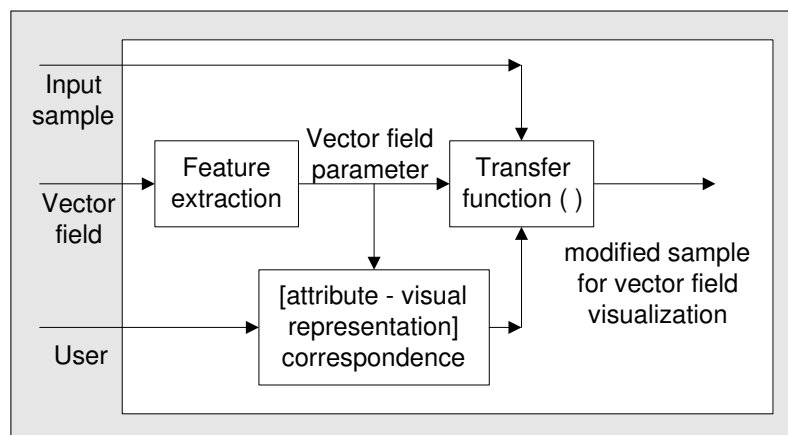


Figure 6.10: A block scheme representing a single variation of a given input sample using a transform operator, whose effect is specified according to an adequate feature correspondence, and which takes the vector field parameter as argument.

## 6.5 Filters and convolution kernels

After having discussed vector field parameters and having specified correspondences with visual representations, I need now to describe the instrument used to perform the information encoding (Fig. 6.10). A gradual and also very versatile way to map information onto a texture example is to use filtering *masks*, where the primary parameters are related to the variables to visualize. In the proposed approach, I apply different *Digital Image Processing (DSP) Filters* to vary some of the texture characteristics in the sample image, to modify and highlight some regions of interests in the output image, as well as to augment a texture with artistic effects. In general, any arbitrary *filter* can be designed, taking as argument the field variable one wants to represent and using this parameter to control an appropriate *convolution kernel*. In this way, different filter design results in different outcoming texture spaces. Filters, besides being general and besides allowing manual design, also have the advantage of offering a pixel-based mapping of the information, while most



iconic representations require too many pixels. Furthermore, the filtering effect over an image can be performed in a continuous way, using *progressive filtering*, as well as in a generic way, using more general *filter banks*.

### 6.5.1 Filter banks

I illustrate some possible schemes for *filter banks* use (Fig. 6.11, 6.12). The illustrations enlarge the block diagrams introduced in Fig. 3.21 in Chapter 3. The filtering can be performed in a linearly progressive way over the whole vector field, or choosing to affect just a specified area, independently from the rest of the image. In general, since the vector field is visualized in a per-pixel fashion, there is no limit in applying filtering operations to customize the output.

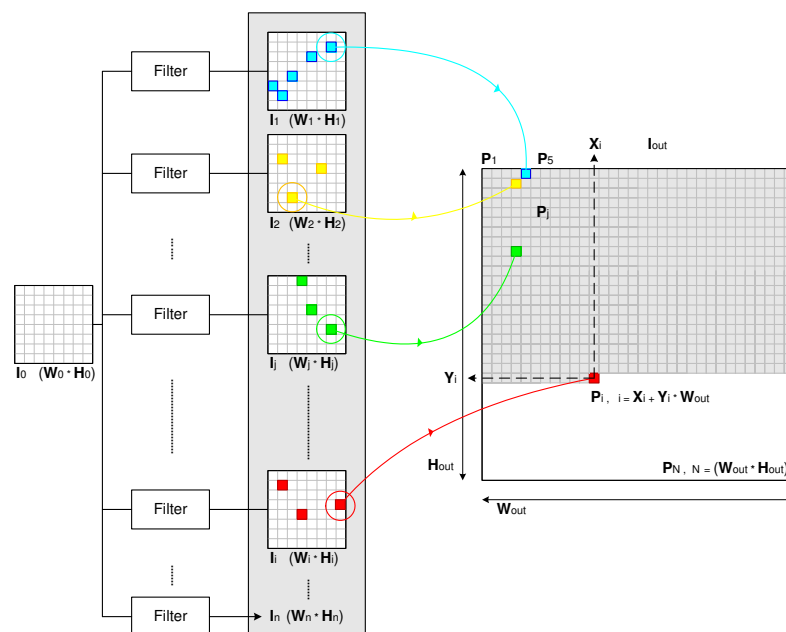


Figure 6.11: In this filtering scheme, the filter bank builds up of filters, which can be also uncorrelated and may modify the input sample in an individual way.

### 6.5.2 Convolution kernels

In order to perform a *convolution*, it is necessary to define the *kernel* of the convolution. The *convolutional kernel* of the filter reflects the filter behavior. Kernels can be *energy conservative*, which means, as terms derived from the physics, that the filter leaves the mean energy of the image unaltered. In this case the sum of the kernel elements is 1, and this leads to the fact that the sum of the total values of the luminance does not vary in the image. In the other cases, filters are *non energy preserving*. Several image processing tools provide a set of *convolutional filters*, generally so called if translation invariant and linear, to modify images. Additionally it is often possible to manually create and edit a *custom* filter.

In practice, a kernel is a matrix  $\mathbf{K}$  that describes how a pixel and its neighboring pixels (*local filters*) are used to calculate the new value of the pixel (*filter response*). Mathematically, a kernel can be represented as a *grid*, generally squared, and the performed operation - *convolution* - is the

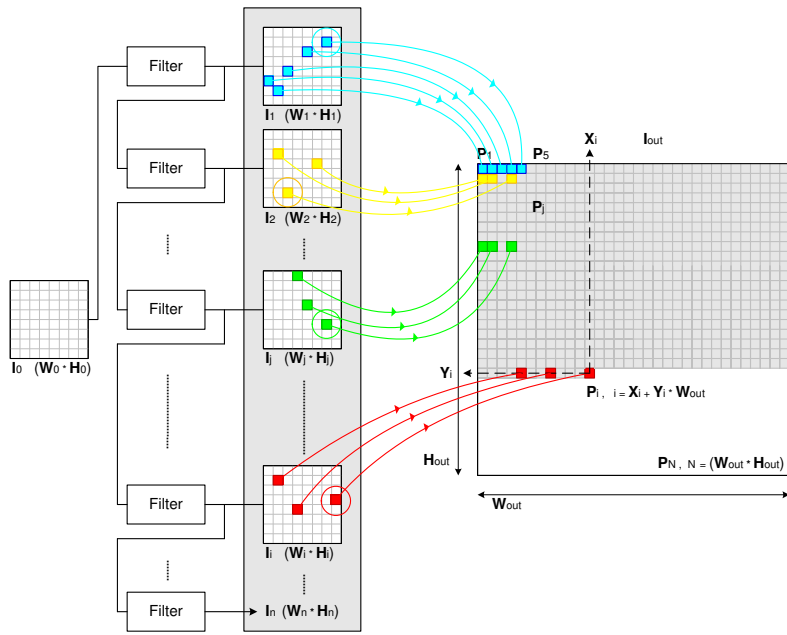


Figure 6.12: In this filtering scheme, *vice versa*, the filter bank builds up of filters, which are iteratively connected and progressively transform the input sample.

sum of the products of all the kernel elements by the image elements:

$$\mathbf{K} = \begin{pmatrix} k_{0,0} & k_{0,1} & \dots & k_{0,n} \\ k_{1,0} & k_{1,1} & \dots & k_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{n,0} & k_{n,1} & \dots & k_{n,n} \end{pmatrix} \tag{6.19}$$

$$I'(x,y) = k(i,j) \otimes I(x,y) = \sum_{i,j} k(i,j) \cdot I(x-i,y-j) \tag{6.20}$$

where  $k(i,j)$  in the equation represent the element of the convolutional kernel  $k_{i,j}$ , and  $I(x,y)$  and  $I'(x,y)$  are respectively the original and the filtered images (see also Fig. 6.13).

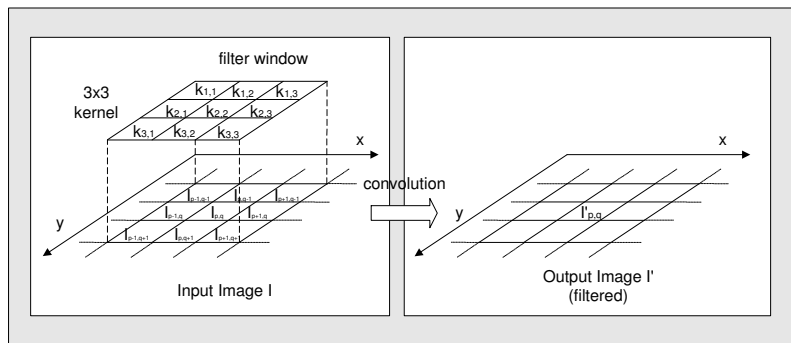


Figure 6.13: Convolution scheme relative to equation 6.20.

### 6.5.3 Filtering effects

In addition to potential structural differences between the images of the texture space (Fig. 6.9), filtering operators can transform the texture samples enlarging the texture space (refer for instance to Fig. 6.14 and to the following chapter). Starting from a simple example texture like those defined in § 3.2, image processing operators are able to arbitrarily and precisely modifying it according to the user's desires and task requirements. A simple anisotropic sample can thus represent simple lines or a more sophisticated directional structure, and can additionally be augmented with icon plots. Using iconic mapping, the calculated attributes can be mapped onto the parameters of certain parametric icons. This results in controlled image density, covering the field of different representations from sparse (classical streamline-based images) to dense (texture-like images), while one usually needs a few completely different approaches to achieve the same effects in the result. As done in Chapter 3 for more simple cases, I explain and discuss here some of the most prominent filtering operators for the task of information mapping in vector field visualization.

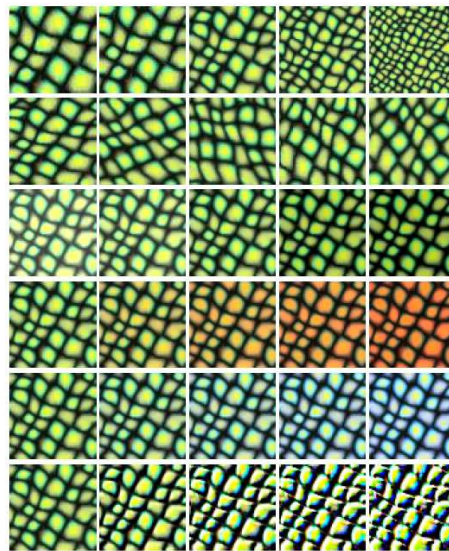


Figure 6.14: A possible texture palette obtained using image filters.

#### Input seed processing

The filters and operators that act over the input texture example (or examples) may operate at *runtime*, as well as at a *pre-* or at a *post-processing* stage.

- Filters are applied during the synthesis process to structurally modify the sample seed, as in the case of the scaling operator, which is responsible of varying the resolution of the example texture.
- Operators may modify the input texture before the synthesis starts; the rotating operators pre-computes for instance rotated versions of the original sample, generating an input set to be used as source during the synthesis.
- Further DSP filters, as blurring, brightening, embossing, coloring, may be applied directly on the synthesized output. They get extracted field variables of relevance, which have to be used as parameters to define the kernels.

### Color coding

Color maps can be used to encode tone factors over the starting sample. Different color scales can be specified to map different field variables; more details and reasons that motivate such specification and special choices are given in § 2.1. The data must represent a continuous function across the area of the image, *i.e.*, adjacent points in the data set should have adjacent data values. Color coding is particularly effective to visualize for instance pressure fields or other scalar distributions. Additional special attributes, as for example vorticity, are sometimes difficult to visualize in an effective way. Color coding can be used to add or stress a special color component at those locations of interest. Different or complementary colors can be expressively used to encode right-wise or left-wise vorticity sense of rotation. Adding *color detail* can sometimes make the fine structure of a value distribution more apparent. One solution is to use color scales with high frequency components. Such scales make small values differences more apparent [162].

As introduced in § 2.1, a lot of research has been conducted to study color perception, and several past, as well as recent works, support the efficacy of color as a differentiating variable among data. The choice of colors and transformation mapping functions is not trivial when visualization method and data sets are complex and present challenges. In the vast majority of cases, effective visualization uses a unique transfer function between data values and color, and employs perceptually equiluminant colorscales (used especially in presence of close proximity of differing colors in the representations). These colorscales ensure that two colors representing the same proportionate value on two colorscales appear equally bright. While perceptually equiluminant is, at best, an approximation across different viewers, methods have been developed to facilitate the selection and evaluation of potential colorscales for perceptual equiluminance.

**The role of color:** tying color to information is as elementary and straightforward as color technique in art: Paul Klee ironically tried to formulate the painting principles with a simple rule: “Gut malen ist einfach folgendes: richtige Farben an den richtigen Ort setzen” (To paint well is simply this: to put the right color in the right place). To some extent, this also applies to visualization. Quoting Tufte [220], as example of representation of multiple information, some fundamental uses of color in information design that can be transferred to scientific complex data visualization are *to label* (for annotations: color as nouns), *to measure* (color as quantity, for example distinguishing entities with contours and rate of change by darkening), *to represent or imitate reality* (color as representation, for instance using shadow hachures in geographical maps, representing water with blue, land with green and so on), and *to enliven and decorate* (color as beauty). Additionally, interplay of light and shadow can highlight other areas strengthening particular meanings as well as the use of backgrounds. As introduced in § 2.1, regarding *color perception*, limited but focused color is sometimes more effective than strong rainbow colors; additionally, a broad set of partial color blindness and color deficiency exist between humans and can be taken into account when choosing color gamuts.

### Compositing color with texture

Besides using color to represent the value of a single variable at a given location, it can be useful or necessary to represent the values of multiple variables at the same point at the same time. Shen and Interrante [179] present an overview of methods to achieve this task and propose a technique for automatically interweaving multiple colors through the structure of a texture pattern (Fig. 6.15-left). Such an approach can also be used as a pre-stage for the presented approach, in order to generate special textures to be used as input, as well as in a post-stage, to stress patterns that still result ambiguous or lead to confusing understanding. Color interweaving also proved to achieve interesting results in [229] (Fig. 6.15-right).

### Bump mapping, embossing

In the previous chapter, in § 5.1.2, I applied the *bump mapping* technique for the purpose of visualizing a third spatial dimension. In this way, it is possible to easily map the additional variable,

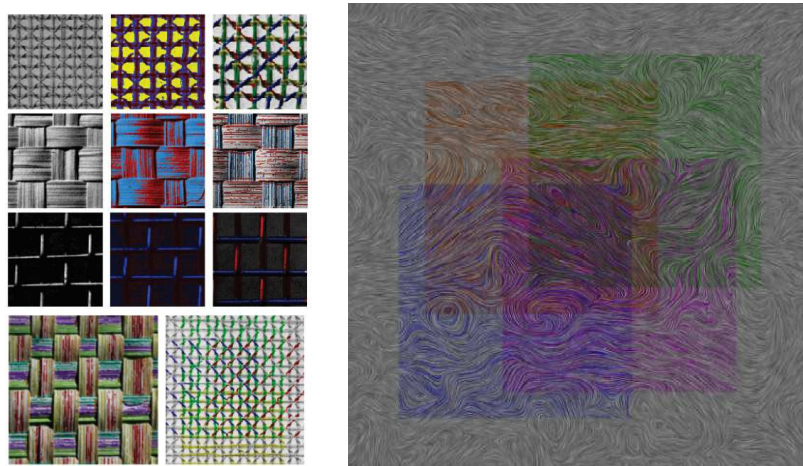


Figure 6.15: Left: compositing color with texture, right: color interweaving (image courtesy, respectively, of Haleh Hagh-Shenas and Timothy Urness).

still producing an essentially 2d output image.

But *bump mapping* is not restricted to the representation of height fields; it is, more in general, a powerful filtering operator that is able to effectively highlight a given distribution, due to its pop-out effect. The encoded variable, in fact, visually comes out of the plane. An important point is also that our visual system is not always well adapted to interpret large data sets, whereas we have superb capabilities for understanding depth-cued images. For this reason, this technique is often used in Computer Graphics to add surfaces details in an easy way, saving in geometric details. *Embossing* is for instance used to act on the normals of a surface for height mapping and to simulate geometric features and relief.



Figure 6.16: Depth of field enhancing the chessmen that threaten the knight on e3. (image courtesy of Robert Kosara).

### Depth of field, blurring

A further example of information enhancement is provided by *depth of field* considerations. Blurring different parts of an image in dependence of their relevance is a common way the human visual system works when focusing on specific elements<sup>16</sup>. In visual perception, the viewing process that leads to perception of depth and distance of objects is called *stereopsis*. The difference between *sharp* and *blurred* parts can effectively guide the viewer's attention. The viewer automatically detects sharp features, while the blurred parts provide non disturbing context for the objects

<sup>16</sup>Whenever interested in a specific part of the environment, the human eye brings the object of interest into the center of the eye where the area of most acute vision, the *fovea centralis* is located, and focuses on that object.

of interest.

Although blurring effects provide an effective way to focus on specific parts of a scene, there have been surprisingly few attempts to use depth of field in visualization. Kosara [118] conducts a study evaluating the *semantic depth of field (SDOF)*, a technique for guiding a viewer to specific information in an image (Fig. 6.16). SDOF is based on the depth-of-field effect from photography, where different parts of a picture are in- or out-of-focus, based on their distance from the focal point of the lens. Generalizing this concept, an object sharpness depends not on its physical position, but on its relevance. Viewers are immediately drawn to the sharp (that is, highly relevant) parts of the image, but they can still choose to look at other, out-of-focus objects. He designed an experiment that contained both basic perception and application components. Sense of depth is there presented as an alternative to *focus and context* display of information. He also explains how DOF is an intrinsic part of the human eye, and for this reason, because of the similarity to the familiar depth-of-field effect, it results to be a quite natural metaphor (effortless by most users) for visualization.

As explained above, the blur operation can be understood as a convolution operation of the image with a blur kernel. Gaussian filtering can be used to simulate depth of field cues.

### Contrast, brightening vs. darkening

A possible meaningful mapping could base on contrast, as well as on brightening effects. Both these operators can be effectively used in visualization, since they are able to enhance and highlight relevant areas or features. In contrasted or bright areas, information is clearly visible and more easily detectable. Highly contrasted images allow to immediately detect zones of particular interest.

### Stylistic representations

A sub-class of visual representations is represented by *stylistic glyphs* or, more in general, by *pictorial representations*. Especially from an educational and artistic point of view, the possibility of using stylistic glyphs to match special data sets or points of interest enriches the visualization and offers a broad variety of possibilities. In Computer Graphics, the use of artistic styles has been used in the last years for non-photorealistic rendering and exploits the capacity of visualization. The use of art in visualization, and especially in scientific visualization, still needs more profound research, especially from the area of perception and psychology. This is thus under the scope of this thesis, nevertheless some inspiration can be gained from considerations and preliminary experimental findings; for this reason I introduce some related concepts and a brief discussion.

Although the endeavors of artists and scientists may strongly differ, visual art can contribute guiding the design and development of effective methods for scientific visualization. In the SIGGRAPH 2001 # 32 course [2], Interrante explains how visualization can be viewed as the art of creating a pictorial representation that eloquently conveys the complexity of a data set; at the same time, "visualization differs from art in that its ultimate goal is not to please the eye, but to communicate information. The arts derive from long and experimental studies and historically leads to successful approach to visual communication, creating visual forms that are evocative and convey meaning". Prominent examples, where centuries of experience and knowledge produced several advances, include architecture and cartography. Visual art can be thus examined to gain some understanding of the basis of human perception of the surrounding world, additionally contributing with creativity. Figure 6.17 illustrates examples of 2D flow visualizations developed by students in the SIGGRAPH 2001 # 32 course. Many studies have shown that seeing is an act of imagination, since the brain tries to extract and abstract important qualities of an image or scene. This is currently a fundamental topic in computer graphics in general, as proved by the numerous recent works and research done on *sketching*. Consequently, although in many cases the targets of arts and science differ, their synergy can at least provide inspiration and new interesting insights in visualization. Kirby cites how Vibeke Sorensen, in her essay "Art, Science" [186] alludes to the

necessity of a *Renaissance team*, "to encounter the divisional chasm between artistic and scientific disciplines", which has been caused by specialization. She also argues how Leonardo da Vinci is by the most considered as the epitome of the artist-scientist combination. This ideal was soon lost to specialization, but can be reached through interdisciplinary research and collaborative efforts.

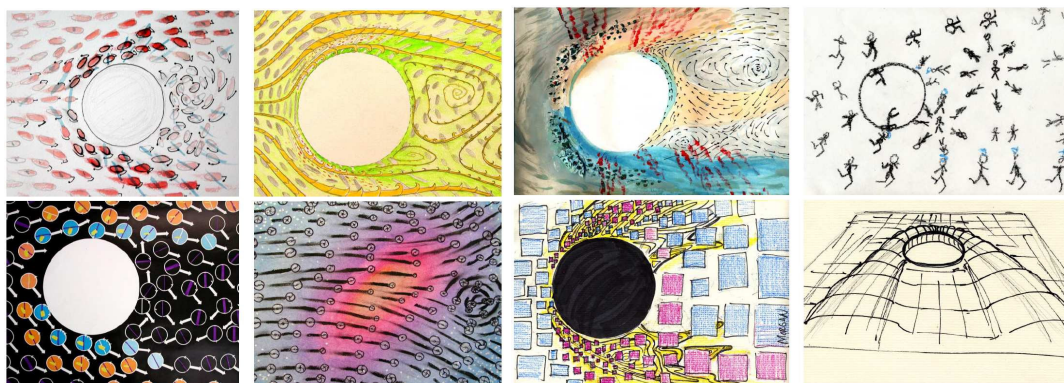


Figure 6.17: Examples of artistically inspired flow visualizations (image courtesy of Robert M. Kirby).

### Pictorial styles

Numerous are the techniques, inspired by the arts, that involve pictorial styles in visualization. Some early works attempts incorporating artistic effects in visualization, taking advantage of the visual richness and effectiveness of the arts, attempting to capture visual design knowledge into guidelines. This is motivated by the fact that artistic works have the ability to evoke powerful emotional perceptive responses. In his inspiring work *Paint by numbers* [76], Haerberly proposes non-photorealism as an alternative to photo-realistic images. He creates impressionistic paintings using an ordered collection of brush strokes and controlling their individual color, size, shape and orientation. Later, Meier [139] shows how to layer strokes, as done to build up oil paintings. Similarly, it is possible here to adopt this technique by choosing differently characterized samples and then using them to abstract the resulting visualization or to confer it a stylistic appearance. *Texture transfer* for instance recognizes a given pictorial style in an image and transfers it to another one, *Photomosaic* subdivides an image in sub-areas giving the impression of a mosaic, *illustrative visualization* and suggestive contours are used in CG to abstract shapes and images, synthesizing them in a few strokes or lines, which convey their essential essay. Illustration provides the potential to interpret physical reality and hierarchically guides the attentional focus accentuating the important information. The pictorial technique called *chiaroscuro* can confer a three dimensional aspect to images, and is *de facto* used in contrasted images as explained above. Common pictorial styles are well known examples recovered from the arts, such as *cubism*. Cubism (inspired for instance by Picasso's works) is hardly the only example of seing of essentials. *Brush strokes* were motivated for instance by Van Gogh's style. Further styles as *impressionism*, *expressionism*, *dadaism* and *pointillism*, among others, also deal with isolate visual essences such as color, form, texture and light. On the other side, *realism* focuses on the most faithful representation of the real world. Figure 6.18 illustrates examples of painterly visualizations [114].

Recent proposal based on art are for instance the pointillistic glyph-based visualization [172], which has been applied to visualize nanoparticles in formation. Through *pointillism*, Saunders *et al.* represent a single mean and standard deviation pair with a larger area square of pixellated texture in the output image (Fig. 6.19). They use spot glyphs and target motif glyphs using concentric rings of color tied to underlying particle count values. This approach, as well as others stylistic approaches, can be easily integrated in the presented work. The pointillistic visualization is in fact glyph- and texture-based and can be used for the visualization of multivariate data.

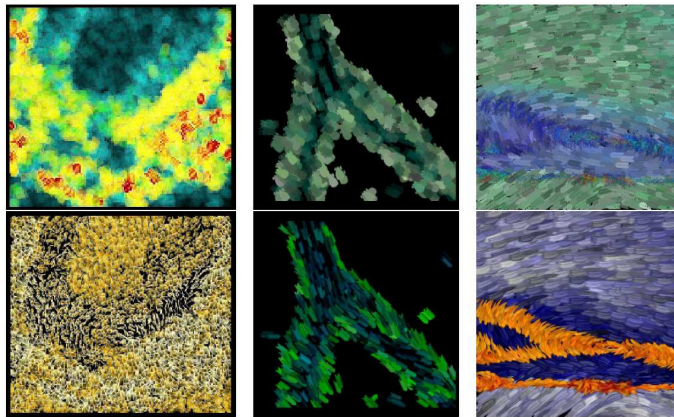


Figure 6.18: Kirby *et al.* experimented with varying the visual representation of underlying data by changing stroke shapes, texture, color, size, and placement. The top and bottom image in each pair are the same underlying data (image courtesy of Robert M. Kirby).

In the context of this research, stylistic texture can confer a sketched appearance to abstract vector field representation, as done in illustrative visualization. More complex artistic styles are not suited for the presented approach, but could contribute to aesthetic effects in controlled texture synthesis (Chapter 7).

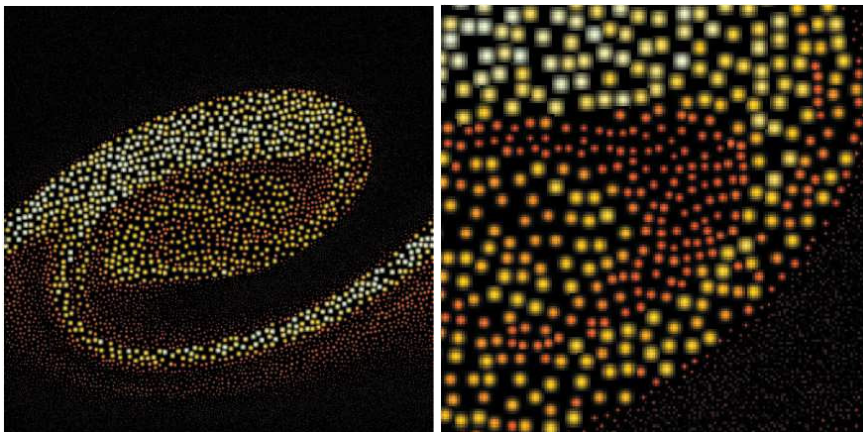


Figure 6.19: Pointillistic visualization (image courtesy of P. Coleman Saunders).

## 6.6 Adaptive information layering

As explained above, textures can incorporate many visual dimensions for intuitive and flexible data encoding, and can thus serve as effective seed primitives for use in scientific visualization. As special application, I propose here a level-based visualization approach, with a special focus on systematic layering of information. This provides an interesting methodology, especially to visualize complex *multi-dimensional* and *multi-variate* scientific datasets in an intuitive and flexible way. It allows to easily synthesize information and to derive insights from the data, as such study facilitates data exploration and analysis. The approach can combine different techniques and link them together through perceptually-based principles. The system is customizable; Fig. 6.20 shows preliminary result, where it is possible to control the information encoding through a set of adaptive sliders, this also allows specifying and superimposing layers where different portion of information is represented. In the remainder of this chapter, a discussion about this starting idea is



provided, open to further extensions and sophistication.

Considering different tasks and audiences, it is necessary to find a compromise between the need for visualizing a lot of variables at the same time and the need for abstracting and simplifying the information. For this purpose, in this final part of the chapter I introduce a methodical approach for information layering that can contribute to effective and expressive visualization of multivariate data sets. The starting idea is driven by the target of investigating and applying various visualization principles to scientific visualization and it is especially motivated by the deal of offering a simple visualization framework for different users and tasks.

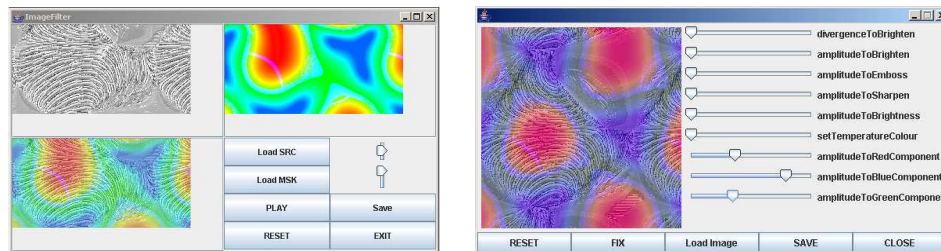


Figure 6.20: Using adaptive sliders to vary the visual representation of the data.

### User- and task-driven data depiction

When visualizing a scientific dataset, it is important to consider that such representation can be perceived in different ways: an effective visualization is not always and not for everybody an effective visualization. Especially in the case of multi-valued multi-variate datasets, users may want to observe the carried information under different perspectives: they may want to display so much information as possible at the same place, in order to investigate possible correlations and interactions between the several variables of the scalar, vectorial or tensorial dataset, or they may want to just isolate some of them to highlight their behavior. Users also may have a different level of expertise and hence, using a too complicated or specialistic visualization may result in confusion. A very simple example is given by the rainbow color scala used to map temperature scalar data to a range of colors ranging from red to blue. Although the rainbow scala is not perceptually optimal, it results of very intuitive understanding for this task: every user can interpret the information at a glance without the need of reading labels or further explanations.

Using an adaptive visualization system, it is possible to differently map the information contained in the data onto different visual representation; such *ad hoc* visualization can be also used to match or confirm user expectations and to help them in the data analysis; it puts the basis for a task- and user-oriented approach. In vision research it is important to consider the role of cognitive influence in perception, as perceptual judgment is done in context of our prior experience and expectation, and our state of situational awareness (§ 2.1). Theory from cognition proves that what is familiar results to be intuitive to understand; to quote Goethe: "That which we know, we have first seen".

#### 6.6.1 Showing composite information

In a level-based schema, filtering and focusing can be implemented to allow simple screen out of unwanted data or to highlight points and areas of interest. The interchangeability of visual representations and mapping criteria allows visualizing the data under several different perspectives. Key issues of abstraction are also considered. The main motivation is that humans use linear thinking, while understanding high-dimensionality is sometime critical. At a panel discussion at the ieev vis 2005 [67], Pat Hanrahan illustrated the main misconcepts in visualization, and explained why the followings do not have to be assumed *a priori*:

- 1.) 3d is better than 2d,

- 2.) animation is better than static,
- 3.) the more variables the better,
- 4.) more views are better than one,
- 5.) realistic is better than abstract.

The reason of this is that everything has side effects, so the answer to the question: *more is better?* is actually often: *less is more*. Consequently, a layering approach, together with user intervention, can be a valid solution, providing a custom-designed visualization for data exploration under different perspectives. Providing just a visualization could leave some possibilities unexplored or could ignore potential parameters interactions. In conclusion, adaptivity in the visualization process may contribute to efficiency, controllability and perception.

### Avoiding visual interference

An effective method to display more features simultaneously can increase the number of attributes and variables we can represent at one time, and thus, the information to visualize. Nevertheless, when integrating these features together for early vision, it is necessary to determine the amount of visual interference that occurs during visualization. The experiments of [215] show (Fig. 2.6) how color or change of curvature are instantaneously recognizable in simple cases, while it is not the case in presence of distractors. Taking into account these theories provides insights for a more effective visualization. A deep understanding of how humans perceive color, shape, and images in general, can help in *ad hoc* designing the tool that serves as instrument for such visualization, telling what is visually compelling. For this reason, methods based on human perception more successfully achieve the goal of visualization, and preattentive features are of particular interest for our visualization approach, allowing a visual pop-out of relevant features. This motivates the assumptions and the specifications done above (§ 6.4.1) for the texture and the attribute space.

### 6.6.2 Layering information

Layering information for better perception has its origin in the past, may be the most prominent example is the *Vitruvian man* of Leonardo da Vinci. In this drawing, Leonardo exemplifies the canons of human proportions, postulated by the roman architect Vitruvius in the I century a.C. The theory shows that the human proportions are seamlessly inscribable in two perfect geometric figures, the circle and the square. Leonardo illustrates this theory with the innovation of using a single drawing, where he superimposes the same human figure giving the perception of two different simultaneous images (Fig. 6.21).

Layering information allows an effective flexible data representation, offering the possibility to visualize multiple datasets simultaneously, or extracting and isolating a part of the information. Considering complex datasets, in general multi-dimensional and multi-variate, we can take advantage of texture features for effective layered data encoding. The method is general and customizable: the user-centered visualization offers easy data interpretation, also making the reading of the data more appealing, which is of great relevance, especially in education. An early example that proves this importance is given by the work of [139], which simulates painting animations, showing how painters repetitively use strokes to build up oil paintings.

The proposed visualization approach is based on layered information and designed to systematically separate relevant scientific information on different levels. When adopting layers, superimposition of information is useful in visualizing datasets via different levels of abstraction. The possibility of separating the information and presenting it with different complexity allows to combine or blend slices of information together. Thus, it is possible to show a great amount of information, but it is also possible to leave a portion of it by side, in case too many overlapping layers result to be complicated and avoid intuitive visualization.

The way each layer contributes to the final data representation can be stressed by the user by setting a *transparency rate*. This can affect a whole slice of information or locally just a portion

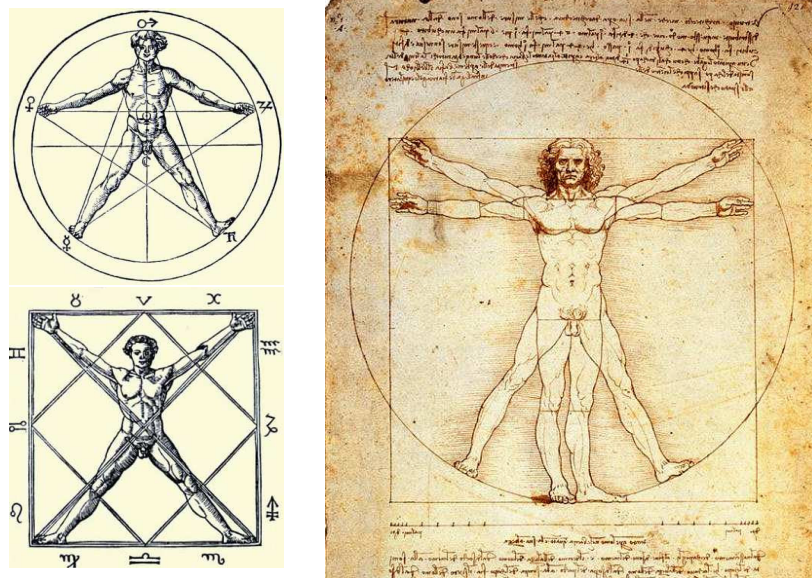


Figure 6.21: Medieval illustrations of Vitruvius's theory (left) and Leonardo's drawing with both the "homo ad circulum" and the "homo ad quadratum" (right)

of it. The representation can be controlled through a set of sliders, buttons, values and threshold settings, letting complex options blending or switching on the base of the level of expertise of the user. Also binary variables may be fundamental in the visualization, allowing to switch-on or -off some mappings or representations of features. The visualization framework builds up of all information slices and for this reason it methodically includes the peculiarities of valid existing visualization techniques. Testing different cases, it has been found that a good compromise is to use up to four layers, allowing several combinations by changing the relationship among layers. It is important to select a good tradeoff between the need for concentrating the amount of data in a single visualization, and the need for highlighting just a relevant part of it. Further, intervention is possible to customize in part the layer representation; depending on the particular application or user, not only the amount and complexity of information can vary, but also the visualization style, focusing for instance on scientific, theoretical, informative or artistic glyphs.

### Systematic information layering

As introduced, a level-based approach also allows the visualization of data sets as a combination of several methods. The information layering can be designed to systematically separate relevant scientific information on different levels. Several aspects of existing techniques, as classified in *direct*, *geometric*, *feature-based*, *texture-based* (§ 2.3), can be covered in the proposed approach.

The presented *texture-based* method is for instance well suited for the visualization of vectorial data, since a dense visualization better represents a complex field avoiding the risk of missing peculiar information, as it could occur in sparsely sampled visualization. As illustrated in the previous chapters, this intuitively conveys the basic structure of the vector field. The technique is particularly intuitive in conveying the principal characteristics (*e.g.* magnitude, curvature). Being then the method sample-based, it adapts a directional texture to the field, accordingly changing its orientation, resolution and attributes. This also allows principles of *direct visualization*, by means of visual cues and artistic styles: either to highlight singularities or to stress points or regions of interest, it is possible to use glyphs, icons and special visual representations (*feature-based visualization*). In this way it is easier to recognize the features of the vectorial dataset. The level-based approach offers finally the possibility to abstract the visualization as done in *geometric approaches*.

### 6.6.3 Comments

In summary, effectively visualizing the information contained in given data sets is a fundamental issue, as data need to be analyzed and interpreted, and information has to be extracted and understood. One of the main problems in visualization is that our ability to collect data is increasing at a faster rate than our ability to analyze it. Also, tools at our disposal may allow the storing of a huge amount of data, but effective ways of analyzing them still need to be investigated, especially understanding how and why some visual representation are easier and faster to perceive than others. Human vision, cognitive sciences and perception provide the needed feedback to achieve this target.

A visualization system that offers variable settings results to be appealing and interesting for educational issues, since different users may set parameters in different ways while analyzing the data and while trying to derive information or identify particular features of interest. Human vision theories, cognitive issues and perception, together with psychology, come as useful complementary information to computer science for such definition and provide an important step in producing perceptually optimized visualizations. In the Computer Graphics community lot of attention is recently given to such research areas and further investigation need to be conducted in this direction.

## Chapter 7

# Other applications

The approach to vector field visualization presented in this thesis finds applications in popular areas of scientific visualization, including the representation of *scalar*, *vectorial* and *tensorial fields*, which are potentially *unsteady* and *multi-variate*. Besides such research fields, covered in the preceding chapters, the texture-based synthesis approach finds interesting applications in *image processing*, for instance for controlled image generation, and especially controlled texture synthesis. This investigation has resulted in the following publications: [193, 201, 203, 196]. As introduced in Chapter 6, image processing DSP filters can be locally applied in a straightforward way. Similarly, thanks to numerous options of the information mapping, applications in *information visualization* can be found as well. In addition, there are several interdisciplinary contributions; techniques for image generation and information encoding can be further investigated and exploited to stronger connect and relate scientific *versus* information visualization, also taking advantage of perception and vision theories.

### Texture deformation and animation

Deforming and animating textures attract a lot of attention and interest in Computer Graphics. Characteristics of texture appearance may evolve or change in time and/or space, a particular characteristic may appear, vary, move or disappear, similarly as for vector fields when describing complex physical phenomena.

Up to this point, I have used texture synthesis theory for the target of flexible vector field visualization. But the underlying theory is in turn useful for *controlled texture synthesis* itself, reserving interesting applications in computer graphics and computer vision, for instance to generate *non-homogeneous* or *dynamic textures*. I introduce such ideas and discuss interesting applications in the following sections.

## 7.1 Steerable texture synthesis

### 7.1.1 Introduction and motivation

Standard texture synthesis is typically concerned (§ 2.2) with the creation of an arbitrarily sized texture from a small sample, where the pattern of the generated texture should be perceived as resembling the example. Following a Markov model approach, the texture is generated by finding best matching pixels or patches in the sample and then copying them to the target. The concept is here extended to incorporate arbitrary filters acting on the sample before matching and transferring; the filters may vary over the generated texture. Steering the filters with properties connected to the output image allows generating a variety of effects. Large textures are found and used everywhere in computer graphics these days. They are fundamental for many applications in computer graphics, computer vision and image processing. Synthesizing these textures from small

samples is in general a powerful way of saving on storage. Textures enrich synthetic objects and computer generated scenes with variety and realism, helping perception of shape, curvature and material. However, due to the complexity of processes generating textures in the real world, one sample is typically not sufficient to describe a large texture. Oftentimes, a texture seems to be generated from a few underlying processes, which blend or merge over the surface. In addition, other easily discerned properties such as lightness, saturation, color, orientation, size, *etc.* change over the surface.

Approaches to regular texture synthesis, as those cited in § 2.2, have been optimized over the last years and are capable of generating high quality results at reasonable computation times; though, most of them only synthesize homogeneous textures starting from single samples. However, it would be interesting to produce *ad hoc* modified outputs, allowing the user to vary features of a chosen pattern, still reproducing its main structure. Recently, a considerable number of techniques, including [225, 252, 185, 242, 72] recognize the lack of local control for output textures and the necessity to offer more flexibility and user intervention for a better match with real appearances of natural objects. Nevertheless, interaction on textures and the generation of non-homogeneous textures is still challenging and promises interesting applications. This work aims at progressing the idea of adding more control over the texture generation process.

Unlike stationary textures, which are characterized by stationary stochastic models, non-homogeneous textures account for a larger class of real world textures [255]. Coating patterns of various animals [224, 213] can be for instance described by textures whose elements change in a progressive fashion. The texture is stationary in a small neighborhood around each point, but the overall texture characteristics vary continuously over the texture domain. The generalization of the per-pixel texture synthesis, as presented in Chapter 3, can accommodate these effects and lead to the synthesis of dynamic processes. The main idea is again to allow the texture being generated from a set of pattern samples, now represented by more complex and structured textures. Each output texel is associated to a sample texture in the set. The set might be generated from a small discrete set of input samples that is enlarged using steerable filters. Filters change properties such as orientation or size. In this section I present obtained results and discuss areas of interest where steering texture synthesis is of relevance.

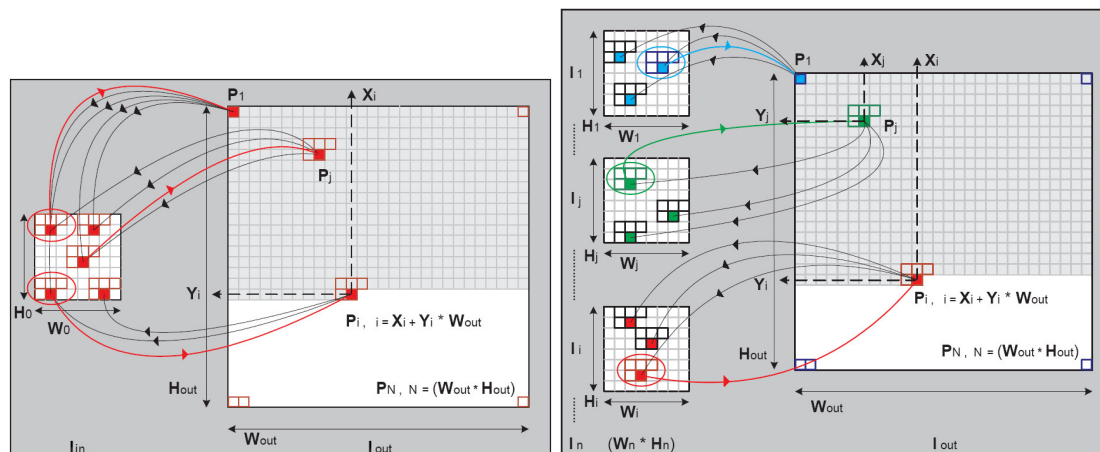


Figure 7.1: Standard pixel-based approach to texture synthesis (left): the pixels  $P_i$  of the output  $I_{out}$  are set checking the most probable pixels in the input  $I_{in}$ . Proposed approach (right): unlike the basic approach, the best matching pixels are derived from diverse samples.

### 7.1.2 Synthesizing non-homogeneous textures

As previously introduced (§ 3.2.3), the proposed method facilitates a set of samples (see Fig. 7.1-right), instead of only taking a single sample as input (Fig. 7.1-left). These samples may be

modified versions of a single generator, or, more generally, an arbitrary set of at most one sample per output pixel. This means that for each pixel in the output the algorithm chooses a specific input sample, inside which the most similar neighborhood and, thus, the output pixel are determined. The process of neighborhood matching is the key factor to enforce continuity in the output (as long as the input is continuous). The idea is conceptually simple, but allows a powerful and general way to produce a large variety of output textures.

### 7.1.3 Controlling the texture generation

When using the framework illustrated in Fig. 7.1-right (*cf.* Fig. 7.1-left), it is possible to speak about controlled texture synthesis: in fact, the approach allows non-homogeneous texture generation or vector-driven texture synthesis, in that filters or a deformation field can control the setting of the output pixels.

The concepts and notations presented here are partly similar to the explanations of Chapter 3; the considerations done in this chapter are derived from the approach to vector field visualization, and provide now an extension to more complex texture patterns, other than simple line-like directional samples. For this reason, the weighting schemes introduced in Chapter 4 are particularly useful, being complex texture samples often characterized by several principal directions. Observing Fig. 7.1-right, note how the pixels  $P_i$  in the texture  $I_{out}$  are set executing a query inside corresponding samples  $I_i$ . The crucial point is how the input set  $I_{in} = \{I_0, I_1, \dots, I_i, \dots, I_n\}$  is composed. A special input  $I_i$  may be used to set one or more specific output pixels, *i.e.*  $n \leq N$ . In the most generic case,  $n = N$ , the user may control every single output pixel  $P_i$  in an individual way, deriving it from an uniquely corresponding sample. Note that the inputs  $I_i$  might also have different pixel counts. This generalized approach is introduced in Chapter 3 and illustrated in Fig. 3.12 where the input set is composed by a matrix of samples in correspondence with the output pixels. Complex filtering and blending produce transformations between diverse textures. The input set may comprise of independent miscellaneous samples, and the desired output may transit between the various appearances. Formally, let  $I_{out}$  be the desired output texture of dimensions  $(W_{out} \times H_{out})$ , and  $P_i$  an output pixel at position  $(x, y)$ : every  $P_i$  corresponds to an array  $\underline{a}$  of dimension  $n$ :

$$\forall (x, y)_{out} \in I_{out} \Rightarrow \underline{a} \in R^n \quad (7.1)$$

The dimensionality  $n$  depends on the number of parameters (each of them can in turn be an array) that operate over the input set. Thus, the array  $\underline{a}$  contains information that reflects image characteristics and defines the input samples  $I_i$ :

$$\underline{a} \Leftrightarrow I_i \quad (7.2)$$

More specifically, we have correspondence between the current pixel at  $(x, y)$  and the particular input texture  $I_i$ :

$$\underline{a}|_{(x,y)} \Leftrightarrow I_i|_{(x,y)} \quad (7.3)$$

Now, let all the samples of  $I_{in}$  be filtered versions of an original sample  $I_0$ , and  $\mathcal{T}$  the transfer function of this filter, then

$$I_i|_{(x,y)} = I_0 \cdot \mathcal{T}(\underline{a}|_{(x,y)}) \quad (7.4)$$

where  $\mathcal{T}$  gets as arguments the  $n$  components of the array  $\underline{a}$  and transforms  $I_0$  on the base of a set of *correspondence rules*. For a straightforward explanation, a simple case study is described by the results of Fig. 7.2, 7.3, where the user intention is to just modify the sample appearance through gradual parameter variation along a specified curve or direction. Consequently, a progressive filter iteratively operates over the input  $I_0$ , generating modified versions of it. This guarantees smooth

variations and continuous outputs.

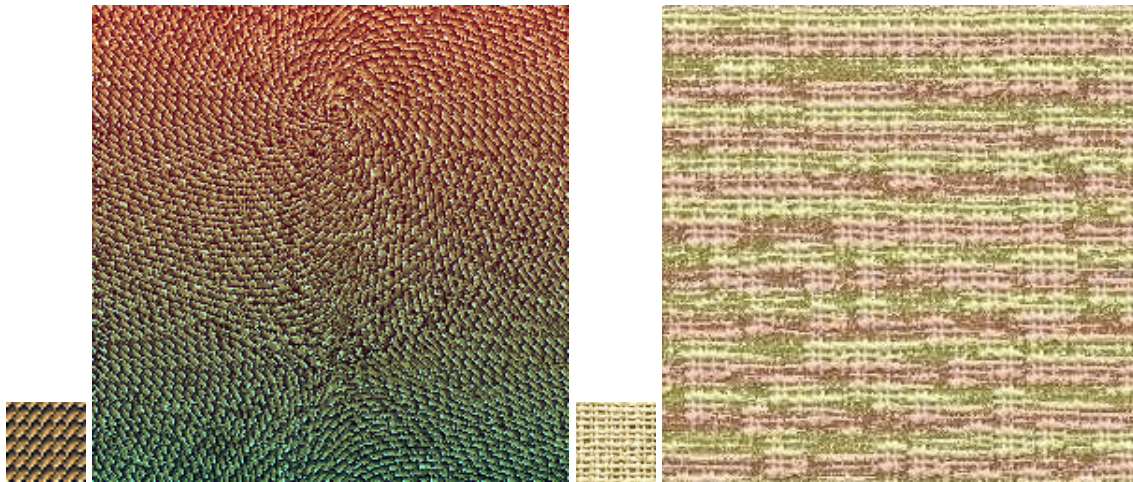


Figure 7.2: Variations along y-axis: coloring & changing direction (left); modifying cloth pattern sinusoidally (right).

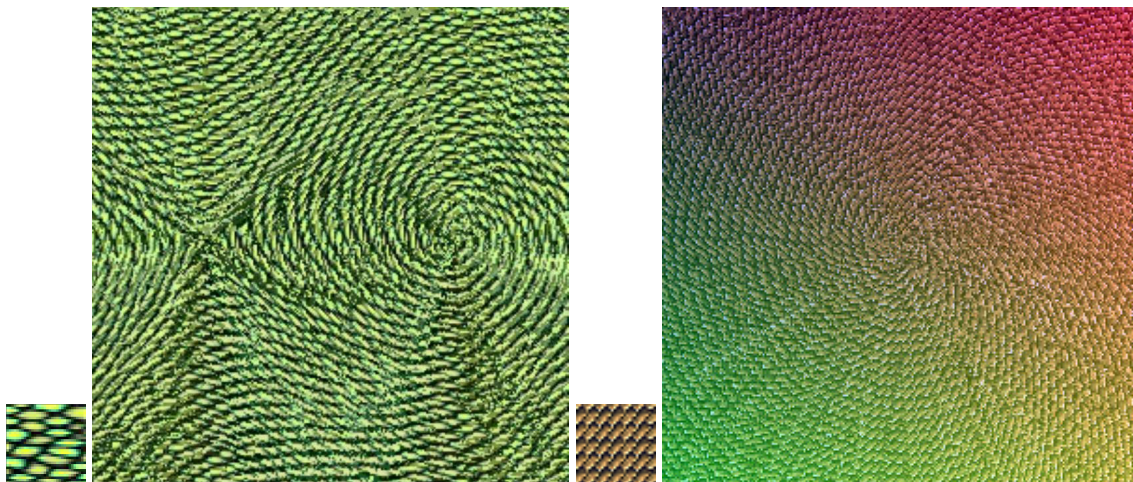


Figure 7.3: Further results: curving an original pattern along a field (left) and gradually modifying color components over the image plane.

Since this mechanism adds visual effects to each pixel according to user-controlled commands, it is possible to generalize it in a broad way. To modify a starting sample, the system implements image-processing filters, including:

**Scaling:** to change the resolution of the sample

**Rotating:** to rotate the sample along specified directions

**Coloring:** to regulate color effects and rgb-transitions

**Brightening, darkening:** to modify the sample by increasing or decreasing its luminosity properties

**Contrasting:** to enhance contrast present in the texture example

**Embossing, blurring:** to introduce emboss or blur effects



**Shearing:** to stretch the sample and enhance directionality

**Warping, bending:** to curve the sample stressing curvature

Some of the described operations are collected in Fig. 7.4. Consequently, the synthesis process requires the specification of a set of *operators*, and also a description of how these operators have to act over the samples. The way a filter is applied to transform the input may be related to several *correspondence rules* (§ 6.4) that can be pre-specified or user-selected. Such rules are for instance based on:

- the coordinate axis of the output image
- the behavior of a specified function defined over the output image
- the intensity of a force field
- the curvature information of a force field
- a magnification factor to emulate relief in the output texture
- a mixture of various rules

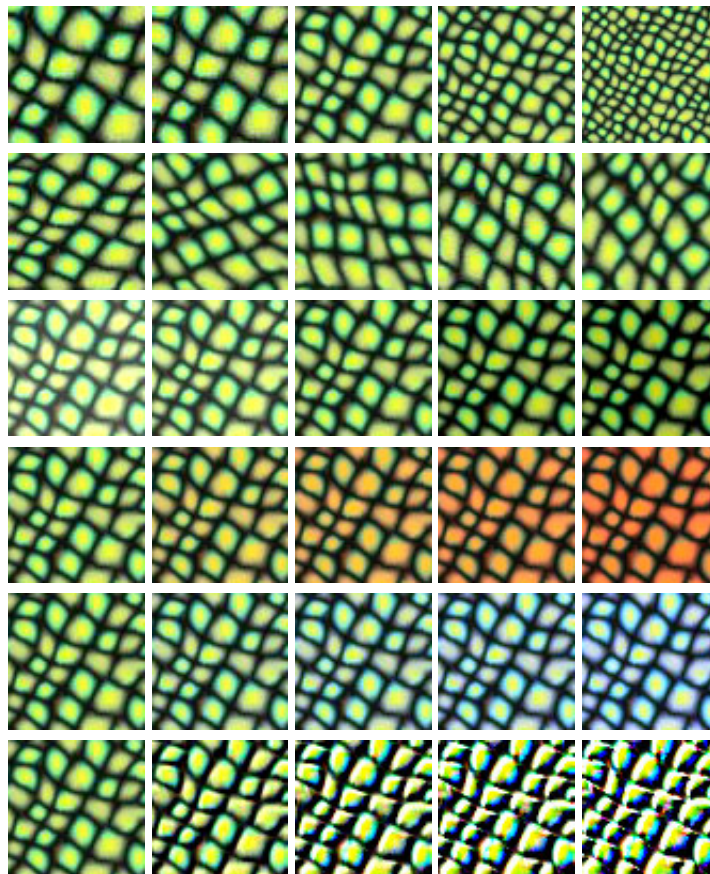


Figure 7.4: Scaling, rotating, brightening and darkening, incrementing red and blue component, embossing.

In general, it is up to the user to choose intuitive and significant mappings. In Figures 7.5, 7.6, 7.7, 7.8, 7.9, 7.10, 7.11 and 7.12 several effects are illustrated.

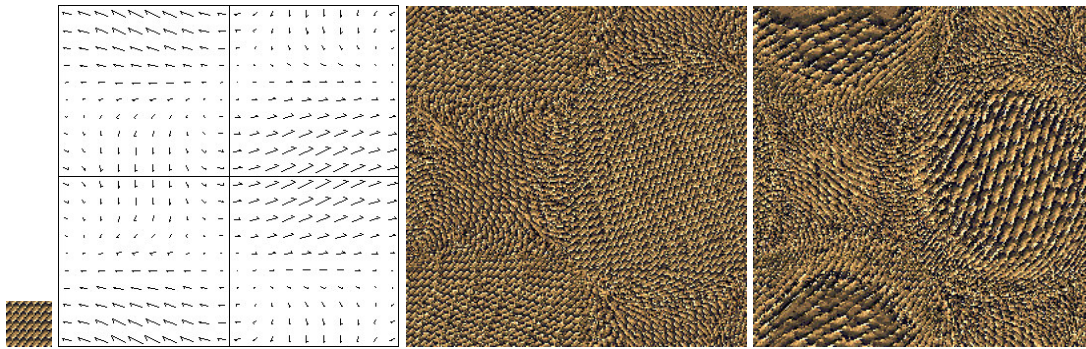


Figure 7.5: Controlled Texture Synthesis: the input sample (a) is modified through a superimposed field (b): phase information is used to change the direction of the pattern (c), magnitude information is considered in addition to scale the original pattern (d).

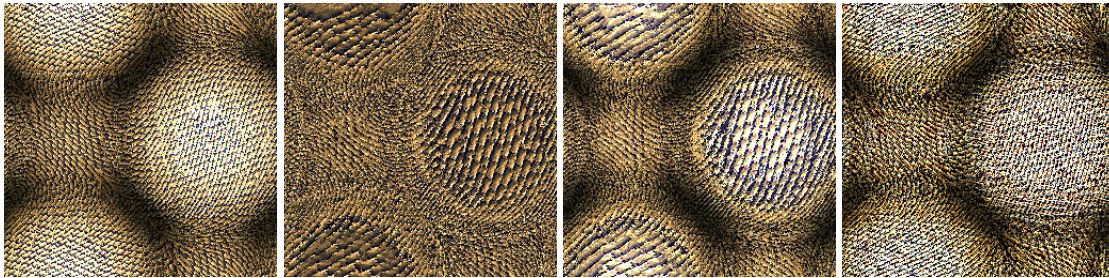


Figure 7.6: Transformation masks: enhancing brightness (a), scaling (b), scaling plus brightness (c), and embossing (d).

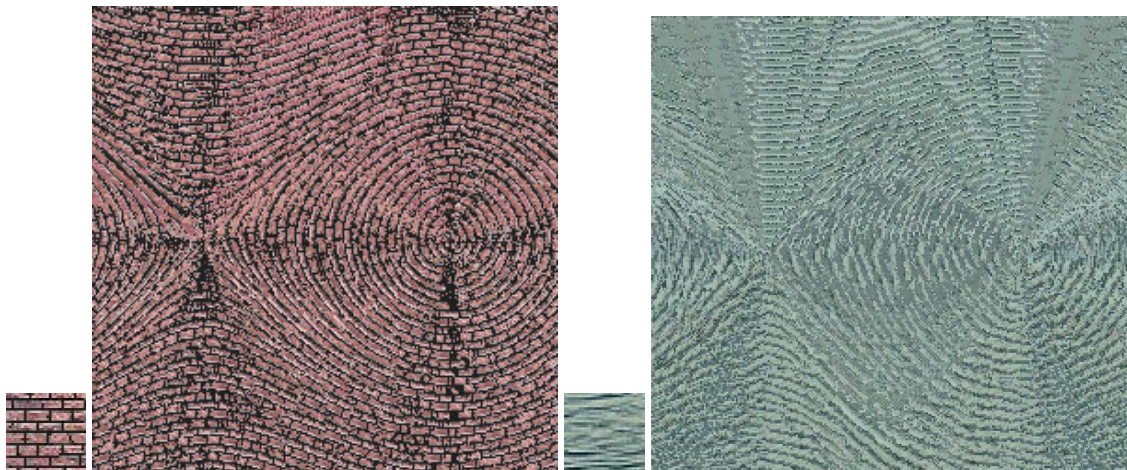


Figure 7.7: Further results: waves and bricks patterns.

#### 7.1.4 Results and discussion

This section illustrates a new method for customized texture generation. Essentially, the main novelty of this approach is a generalization of pixel-by-pixel texture synthesis that takes into account a mapping from output pixels to input sample. The main observation is that continuity of the output results from the neighborhood matching and, especially, smoothness across the different input samples is not necessary. The implemented algorithm is intuitive and the system is easy to use: the user is able to freely combine filters and parameters yielding easy and flexible control over the

generation of the output texture.

The results shown in this section were produced using the proposed method (§ 3.4) implemented using improvements presented in Chapter 4 and using the multi-resolution approach based on Gaussian Pyramids, as explained in Appendix (§ A.2.1). For the examples,  $16 \times 16$ -sized samples were mostly used and the system required a few minutes to synthesize  $256 \times 256$  and  $512 \times 512$  large output textures. Again, as the input sets can be pre-computed or pre-filtered, the processing times are not delayed, regardless of how many variables the user is modifying; consequently, there is no penalty for the added degrees of freedom.

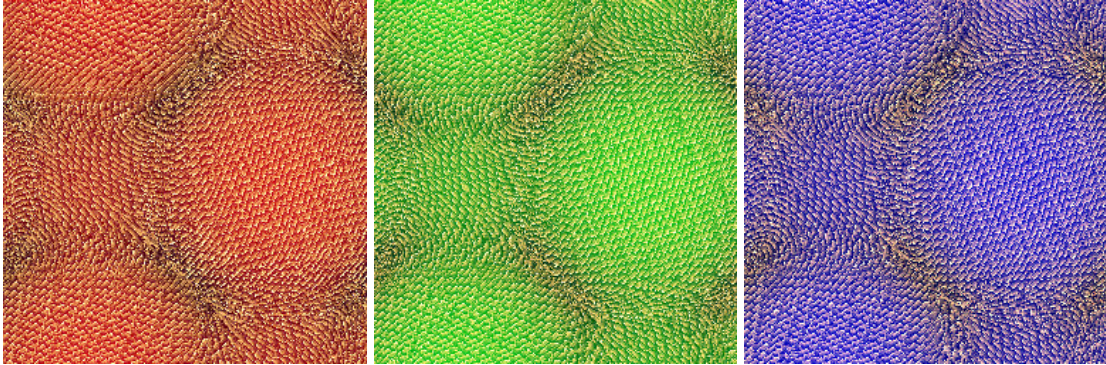


Figure 7.8: Further filtering: increasing the red (a), green (b), blue (c) component along with field amplitude variation, in a local manner.

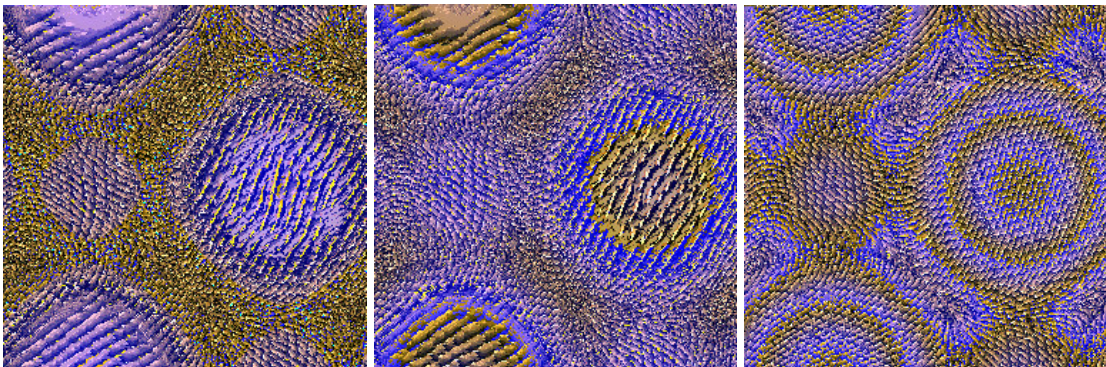


Figure 7.9: Applying different color masks: with (a, b) and without (c) magnitude scaling.

## 7.2 Time- and space-variant texture synthesis

### 7.2.1 Introduction and motivation

In addition to texture variation illustrated above, objects appearance can be influenced by several surrounding circumstances over time, and, as such, *temporal texture synthesis* plays a fundamental role. Furthermore, controllability remains a crucial point to generate desired outputs. In the spirit of Section 5.1.3, it is possible to extend the controlled static texture synthesis (§ 7.1) to the more general controlled animation of textures. In this section, I present a flexible methodology for controllable synthesis of time-varying textures. The objective is to control textures in a general way, generating motion along given directions and simultaneously influencing the texture appearance in a dynamic way. The proposed algorithm allows producing a variety of outputs and provides a smooth and continuous temporal animation of the frames.



Figure 7.10: Further results.

The contribution of this extension is to provide a straightforward methodology to perform the synthesis of dynamic textures in a user-defined way. Although approaches to synthesis of stationary processes exist, there has been comparatively little work in the specific area of field-driven texture synthesis. The extension to *steerable texture synthesis* proposed here permits intuitive sample-based and arbitrarily field-constrained texture synthesis enriched by a variety of customizable effects. In particular, this method allows local control to change the texture resolution, beside its color and other attributes.

## 7.2.2 Motion in texture synthesis

Significant techniques have emerged in the computer graphics and image processing literature; much effort has been invested in producing useful and effective algorithms, nevertheless, the need for visualization of variable complex phenomena requires further investigation. As compendium to the State of the Art on texture synthesis (§ 2.2), I succinctly introduce here some works that deal with variable texture synthesis. In particular, I review on existing approaches for the synthesis of time-varying textures. Although many advances have been achieved in texture synthesis, the lack of control still remains a focal issue in designing new synthesis techniques. As recognized by [124], most techniques that offers some kind of control, only provide little amount of texture variability and are mainly restricted to random seeding of boundary conditions, obtaining rather unpredictable results. Lefebvre and Hoppe [124] propose texture variability, but their target and approach differs from the one presented here. They desire an aperiodic infinite texture that they modify introducing new elements via drag-and-drop. Kwatra *et al.* [119] visualize textures controlled through a flow field. Nevertheless, the approaches are basically different: they use a global synthesis optimization process, which takes effect on the whole output texture, while we want local control and we can manage numerous texture attributes (such as resolution, color, shading, embossing, besides orientation) in a general way, in order to provide additional degrees of freedom for controlled synthesis of the texture variation and evolution.

Regarding statistical methods that model textures in motion or that produce a sort of variation in textures, they mainly concentrate on repetitive processes and deal with the modelling and reproduction of temporal stationarity, like in sea-waves, smoke, steam, foliage, whirlwind but also talking faces, traffic scenes etc. (Fig. 7.13). These approaches typically suggest to use a sequence of frames to simulate cyclic motion or periodic effects that are in some way similar to movement. For this task, an input sequence of samples - *input movie* - is needed. This input has the function of training data, from which the procedures directly acquire the necessary information and reproduce it through statistical learning in an output sequence. The first approach that gives a statistical characterization of textures is the early work of Julesz [103]; successively, about twenty years ago,



Figure 7.11: Examples of a variety of synthesized textures.

he introduced [106] the concept of *textons* as "*putative elementary units of texture perception*" and therewith opened the road to a very extensive research, also in the field of modelling motion in texture.

In recent years, Wei and Levoy [241] propose a 3d extension to their model to create solid textures or, as particular case, temporal textures, in case the motion data is local and stationary both in space and time. Bar-Josef *et al.* [12] employ *multi-resolution analysis (MRA)* of the spatial struc-



Figure 7.12: Examples of a variety of synthesized textures.

ture of 2d textures and extend the idea to dynamic textures (*movie texture*), they directly analyze a given input movie and generate a similar one through statistical learning. Akin to this, Pullen and Bregler [159] propose, modelling local dynamics, a multi-level sampling approach to synthesize *motion textures*: new (cyclic) motions that are statistically similar to the original. Li *et al.* [127] propose a technique named motion texture for synthesizing human-figure motion: they model a motion texton by a *Linear Dynamic System (LDS)*. Schoedl *et al.* [173] also model textons with *LDS* for *video texture*, looping the original frames in a manner that the synthetic reproduction is minimally noticeable to the user. Doretto *et al.* [49] generate *dynamic textures*. Dynamic textures are sequences of images of moving scenes that exhibit temporal regularity, intended in a statistical sense. In the specific case of spatially coherent textures (textures that exhibit temporal statistics), Soatto *et al.* [184] (and [49] for both spatial and temporal regularity) synthesize a *homogenized* version of the original sequence, through a model designed for maximum-likelihood or minimal prediction error variance. They use *LDS* to model a texture by an *auto-regressive, moving average (ARMA) multi-scale process*. Similarly, Fitzgibbon [61] uses an *autoregressive (AR) model*. Again for stationary data, Szummer and Picard [190] use a *spatial-temporal autoregressive model (STAR)*, which provides a base for both recognition and synthesis. This model produces convincing results, nevertheless, it cannot capture curvature and rotational motion.

Modelling more complex variations - *nonlinear dynamics* - is difficult, it requires the use of multiple linear systems, and thus it is still challenging [127].



Figure 7.13: Temporal regularity is exploited in animation of clouds, smoke, fire, steam, waves, waterfall.

### 7.2.3 Animating textured images

The works cited above mainly consider sequences of images that exhibit certain stationary properties, specifically repetitivity or cyclicity, in time. For this reason, most methods are limited to the visualization of natural processes like sea-waves or wavy water, rising steam, fire, smoke, foliage, whirlwind, *etc.* Another problem is that the synthesized motion may lack global variations when the training data is limited. These techniques require a starting frames sequence, or *input movie*, which has to be learned, capturing the essence of the dynamic process, from the system and then reproduced. Therefore, these methods particularly focus on *statistical learning*, having as task texture analysis and recognition besides texture synthesis. They usually assume the texture to have been generated from an unknown stochastic source process, which they need to estimate and model.

Also for this reason, the idea presented here is different from previous works. The intention is now not constrained to reproduce repetitive or cyclic motions; the target is to model general variations of textures. Instead of visualizing cyclic processes such as repetitive waves, I concentrate on distorting a given pattern by progressively varying some of the attributes that define its structure. Starting with an input texture, a control field, and a suitable synthesis algorithm, it is possible to arbitrarily modify textures in a variety of ways and then to continuously animate this transformation. Briefly, a sequence of frames is synthesized, that depicts the evolution of a texture over time. The algorithm still works at a *per-pixel* level, and the synthesis is performed in *multi-resolution*.

#### Field-driven synthesis

The motion and variation in time are controlled through the selection of a control field. This field is potentially multi-valued and multi-dimensional; it varies the texture structure aligning and adapting it along new directions. Such vector field is variable over time, hence, at each time step the synthesis process produces a corresponding texture frame. User intervention is allowed in defining the field to influence a given example. In this way, textures may be arbitrarily controlled, deformed, varied over time.

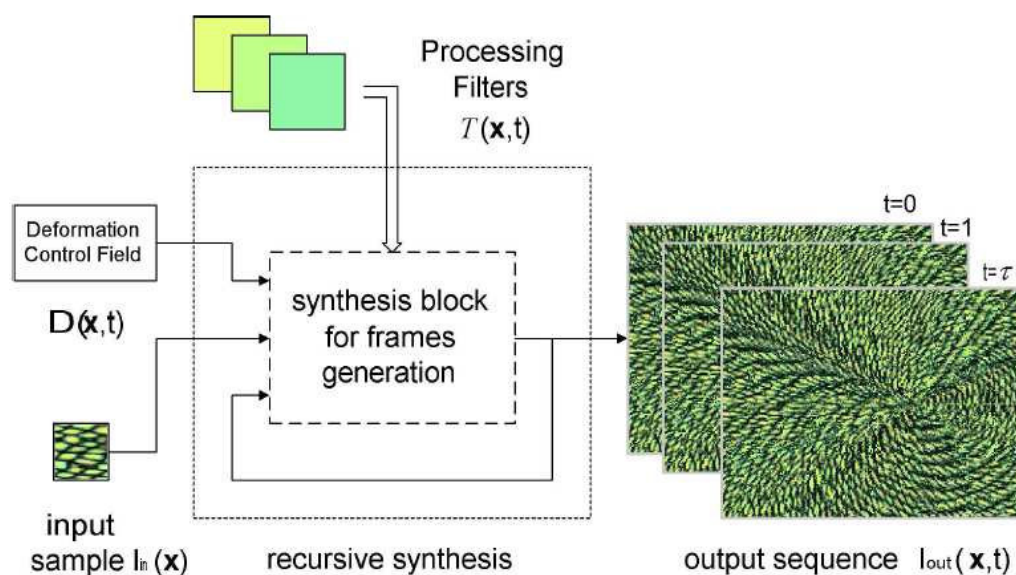


Figure 7.14: Block diagram for the generation of multiple subsequent frames: their animated succession generates a time varying texture.

### Recursive passes

The block scheme of Figure 7.14 sketches the generation of the output sequence. As done in § 5.1.3 for the visualization of unsteady field variation, the synthesis is recursively iterated a desired number  $\tau$  of frames. The dashed block representing the single frame generation (used for standard texture synthesis) has been appropriately extended to additionally acquire information from the previous synthesized frame. For this scope, the cubic neighborhood model defined in § 5.1.3 is here used, again building a three-dimensional structure around the pixels. Figure 7.15 shows how to construct spatio-temporal neighborhoods; it illustrates three-sized models in single-resolution. Figure 7.16 shows a five-sized model in multi-resolution.

Let consider the current pixel to synthesize, then its cubic neighborhood incorporates the adjacent pixels in the *L-shaped neighborhood* - spatial information - plus a number of *square-shaped neighborhoods* - temporal information - at corresponding location from the underlying complete layers at previous time steps. Such neighboring pixels carry coherence information; in this way, this synthesis procedure achieves continuity and preserves smoothness in the spatial domain  $(x, y)$  and in the temporal domain  $t$  as well. In fact, when synthesizing temporal evolution of a texture, the individual frames are not independent realizations from a stationary distribution, for there is a temporal coherence intrinsic in the temporal field that drives the animation.

### Algorithm

Referring to Figure 7.14, let  $I_{in}(\mathbf{x})$  be again the selected input sample; I define  $I_{out}(\mathbf{x})$  to be an output texture in desired resolution, where  $\mathbf{x}$  is a vector, for simplicity in two-dimensions:  $\mathbf{x} = (x, y)$ . The target is to generate an animation of  $I_{out}(\mathbf{x})$  under the action of a control field, now called in general *deformation field*  $D(\mathbf{x}, t)$ , and during an arbitrary period of time  $\tau$ :  $t \in [\tau]$ . The basic pattern of the texture is modified by the time-varying deformation field and its features are potentially controlled through *ad hoc* definition of transfer functions  $T(\mathbf{x}, t)$  (similarly as done in Chapter 6). More precisely,  $D(\mathbf{x}, t)$  and  $T(\mathbf{x}, t)$  influence the specified texture example over time, respectively by forcing it along new directions, and by modifying its appearance.

Formally, I synthesize a texture sequence  $I_{out}(\mathbf{x}, t)$ , with  $t = 0, 1, \dots, \tau$ , and  $I_{out}(\mathbf{x}) \in \mathbb{R}^2$ . As drafted below in the *Implementation steps* paragraph, this frame collection  $\{I_{out}(\mathbf{x})\}_{t=1, \dots, \tau}$  is gen-



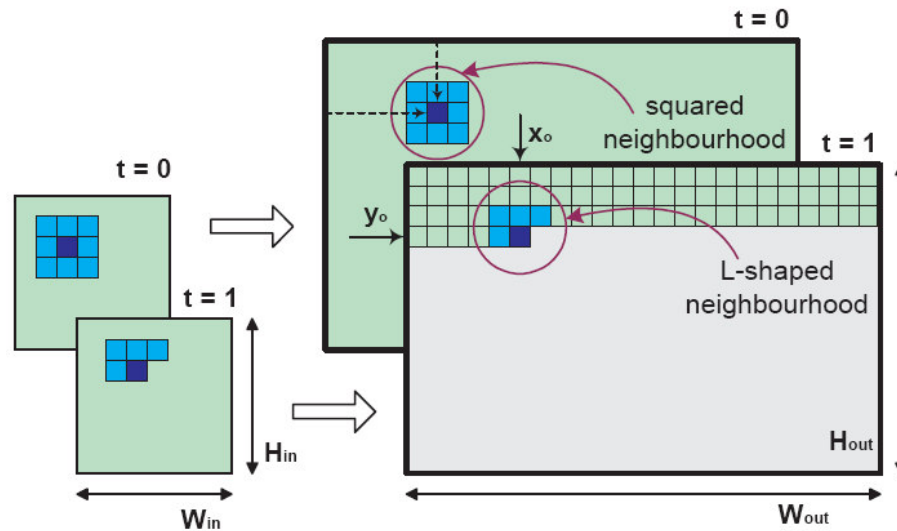


Figure 7.15: Synthesis schema for the generation of two successive frames ( $t = 0, t = 1$ ) using a neighborhood of size = 3.

erated in an automatic way with a recursive system (refer to Figure 7.14): the synthesis of each temporal frame is influenced by information derived from the previous frame (or from a set of previous frames). This facilitates achieving smoothness along the temporal evolution. Figure 7.15 illustrates the synthesis of two successive frames in single pass resolution. Assuming to have already completed the starting frame ( $t = 0$ ) using standard planar synthesis, it is now desired to synthesize a generic pixel (the dark blue one) inside the following frame ( $t = 1$ ). At this synthesis stage, the light green part of the image has been synthesized, and, proceeding in scan-line order, the neighboring pixels above and on the left of the current pixel at position  $(x_0, y_0)$  are known, together with the pixels belonging to previous frames. In order to incorporate temporal information, the pixel at the corresponding location  $(x_0, y_0)$  that belongs to the previous step is considered. That frame has been entirely synthesized, therefore the complete squared neighborhood around that pixel is known. In this way, it is possible to build three-dimensional neighborhoods (bright blue pixels), which comprise of the L-shaped neighborhood from the current frame  $t$  plus the corresponding squared neighborhood from previous time step ( $t - 1$ ). The pixels that build the extended neighborhood have to be adequately taken into consideration: it is important to note that the preceding temporal frames contribute to the 3d-neighborhood with different - non-uniform - weights, as pixels in the current frame present less correlation with pixels that belong to previous frames (see also Chapter 4). As done in § 5.1.3, concepts from § 4.5.1 lead here to *spherical* and *ellipsoidal* weighting schemes for the neighborhood models, which are depicted in Figure 5.5. Such weighting schemes are particularly beneficial for complex texture patterns.

This is also shown in Figure 7.16, which describes the technique for a three-frames synthesis process in case of larger sized neighborhood (size = 5) and with two levels of multi-resolution image pyramids. From left to right, the illustrations show the texture evolution in time: the right-most output slice represents the current frame. Looking instead from right to left, the frames go back in time and the influence of the layers with respect to the right most one decreases. From bottom (coarse scale) to top (details), multi-pass synthesis is executed using sub-band transforms.

### Implementation steps

The algorithm synthesizes all the output pixels of a single frame, and this for each frame in succession. Since the concepts have been extensively explained in the previous chapters, I just provide here a brief step-by-step summary.

A cubic neighborhood  $N^3$  is build for every output pixel  $P_o(x, y)$ , with  $x \in [0, W_{out}]$ ,  $y \in [0, H_{out}]$ , being  $W_{out}$  and  $H_{out}$ , respectively, the width and the height of the output image. A similarity metric, which accounts for these neighboring pixel values, is computed using least squares, and is then used as distance function to measure neighborhood similarity. In this way the best matching pixels can be chosen.

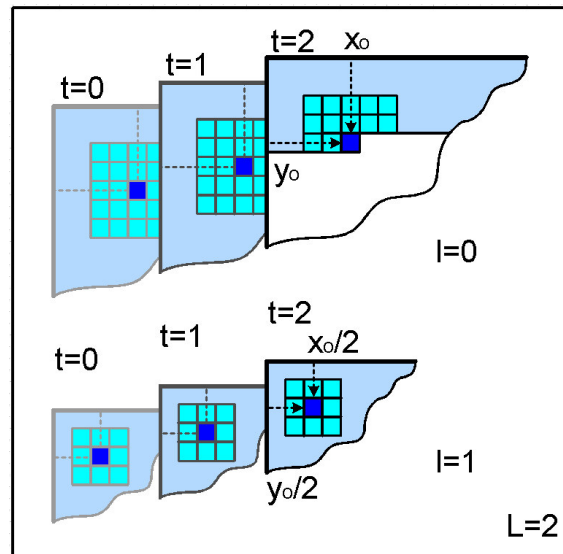


Figure 7.16: Five-sized neighborhood model (L-shaped plus square-shaped) with two-level multi-resolution synthesis ( $L = 2$ ). Lighter colors in the synthesized frames means weaker correlation to the current one ( $t = 2$ ).

The fundamental steps are:

1. *Initialization*: set values of output image dimension, time period, pyramid levels; define the controlling deformation field
2. *First step*: perform two-dimensional synthesis ( $t = 0$ )
3. *Further steps*: perform three-dimensional synthesis ( $t > 0$ )

The algorithm from § 3.4 is used for the first step of the algorithm. The synthesis steps for the generation of the frames at general time step  $t > 0$  are conducted as follows:

- Build a neighborhood  $N^3$  for every pixel  $P_o(x, y)$ , including the L-shaped neighborhood and further squared neighborhoods (in number and resolution depending on the size of  $N^3$  and on the number of pyramid levels  $L$ )
- Evaluate the deformation field  $D(x, y; t)$  at the current output position and time instant, calculate prominent field features (phase, magnitude, curl, ...)
- Use this information to accordingly modify the input sample influencing its structure and setting the calculated directions to be the new orientation
- Build all possible  $N^3$  around  $P_i(i, j)$  inside the input sample and calculate the similarity metrics
- Compare the entries of the array of input neighborhoods with  $N^3(P_o(x, y))$ , on the base of distance function, and choose the most similar one

- Select that value  $P_i(i, j)|_{Max\_similarity}$  and set it to be the current output pixel
- Proceed in scan-line order till the output texture is completed
- Repeat this procedure for all levels  $l$  of image pyramid and for all following temporal frames  $t$  within the sequence of temporal range  $\tau$

### Filtering

As shown in Figure 7.16, in addition to the use of a deformation field, textures may be modified over time in a progressive way through filtering operators. Such functionalities are integrated in this approach to allow more manipulation. Figures 6.11 and 6.12 from Chapter 6 show possible blocks schemes for generic and progressive filtering. This operation may be inserted in the temporal synthesis scheme to extend its functionalities (note the filter blocks in Figure 7.14).

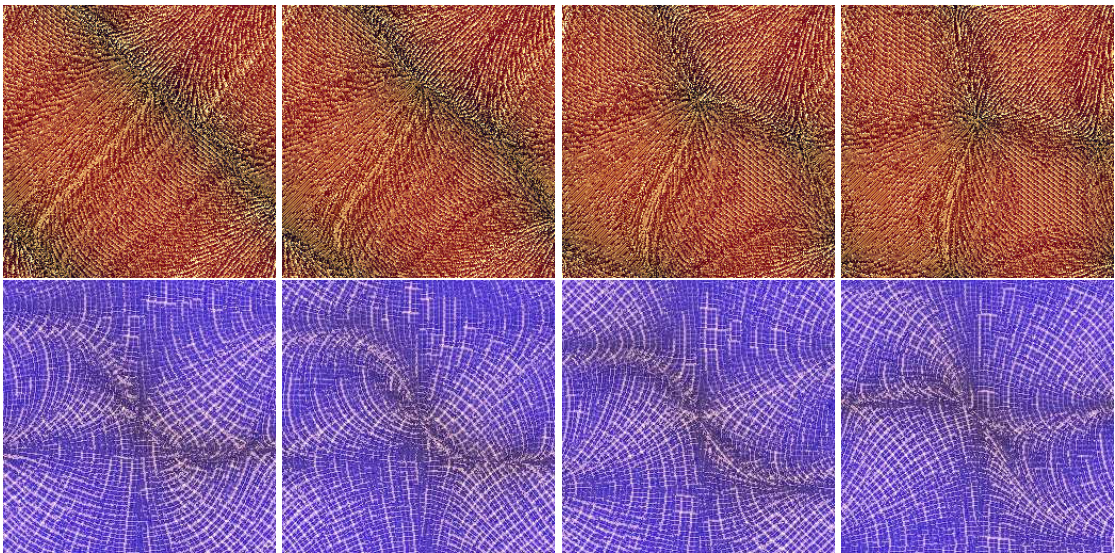


Figure 7.17: Examples of filtered texture frames sequences. In both cases, four temporal frames extracted from a longer sequence are shown. They represent the time-varying texture at different time instants.

### 7.2.4 Results and discussion

The approach has been tested in a variety of cases, in particular for structured textures (Figure 7.18, 7.19, 7.20). Mainly directional or semi-structured samples have been employed, since patterns that present accentuated features along a major axis better exploit movement in the given directions. Using textures having anisotropic pattern, it is easier to visualize and enhance the information carried by the control vector field. Complex patterns (Fig. 7.17) may present artifacts when adopting small neighborhoods for the synthesis. Anyway, patterns containing more principal directions can be as well successfully employed adopting the weighting schemes of Chapter 4.

The values that were typically used to produce the outputs presented in this section are  $16 \times 16$  or  $32 \times 32$  pixels for the size of the input samples, and  $256 \times 256$  and  $256 \times 512$  points resolution for the generated outputs. The synthesis time is essentially comparable to the other pixel-based approaches. The length of the frames sequence can be arbitrary and animations could be in theory endless. Therefore it is possible to generate infinitely long sequence of images that can be looped seamlessly by opportunely designing a cyclic controlling function, which can repetitively deform the pattern without producing artifacts or discontinuities between different animation cycles.

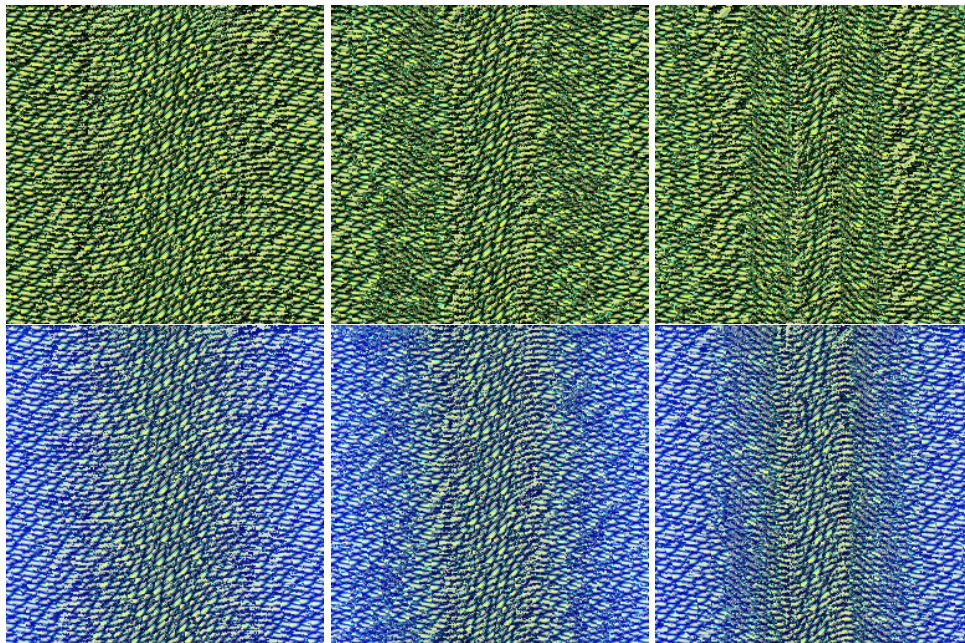


Figure 7.18: A sinusoidal distortion field modifies a sample (top), and a filter based on a blue component additionally highlights the field intensity in the same frame series (bottom).

The algorithm is based on a recursive synthesis procedure that uses the introduced three-dimensional neighborhood model. This approach is promising for many applications, including the visualization of animated textures and the texturing of dynamically varying surfaces, as it is possible to generate animations of textures in an automatic and straightforward way. The algorithm does not present any restriction for the choice of the control field expression and of the filtering parameters. This offers a technique, which is easily adaptable for a variety of applications.

### Limitations and optimization

When using texture patterns characterized by having a complex structure, a large sized neighborhood is required to allow the synthesis algorithm to learn and reproduce the sample statistics. In such a case, the computational complexity rapidly increases. In addition, the eventual use of time-varying scaling operators over the input sample also leads to a larger neighborhood size.

An important point to consider is the following: in a few cases, the occurrence of a sort of flickering effects in the transition between some frames during the animation was noted. The input sample may occasionally strongly varies (through rotation, scaling and other operators) between the different time steps: in such a case, the collection of successive neighborhoods for the best pixel choice could present discontinuity. This problem is solved by considering, for the neighborhood matching, only samples that were *a posteriori* re-rotated, with respect to the current pixel as center of the operation. This guaranties better and smoother results. Occasional spatial or temporal discontinuities might occur due to the pattern structure of the chosen sample and they result from the nature of the algorithm: in this case they can be removed in a simple way by using a larger neighborhood or a higher pyramid levels number, or could be blurred away. Smoothing for output refinement also could be applied at a post-processing stage.

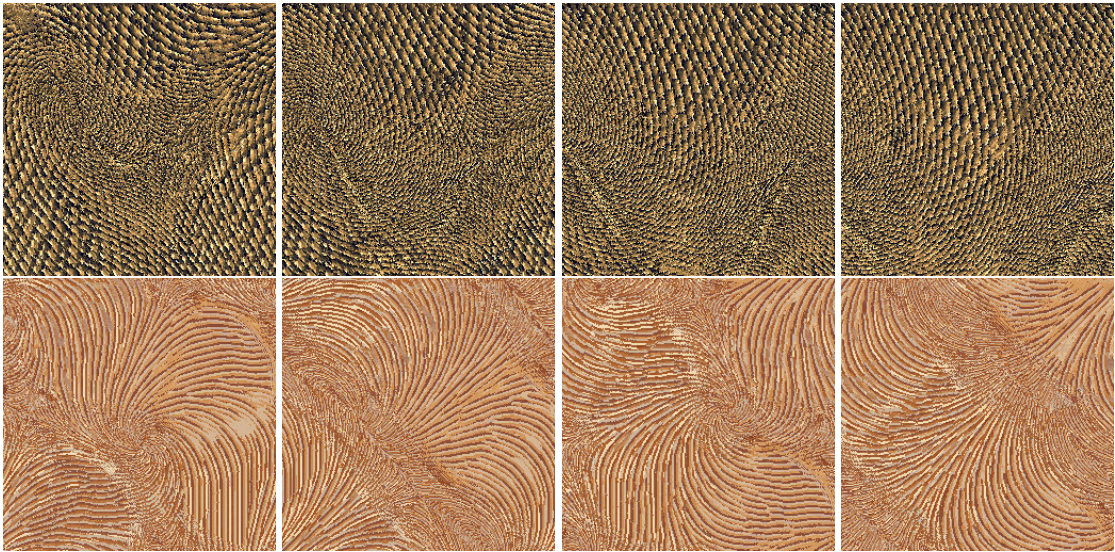


Figure 7.19: A *fabric* (top) and a *directional* (bottom) texture are controlled by a vector field. Scaling of the sample reflects magnitude information of the field; the resolution of the original sample is accordingly locally adjusted.

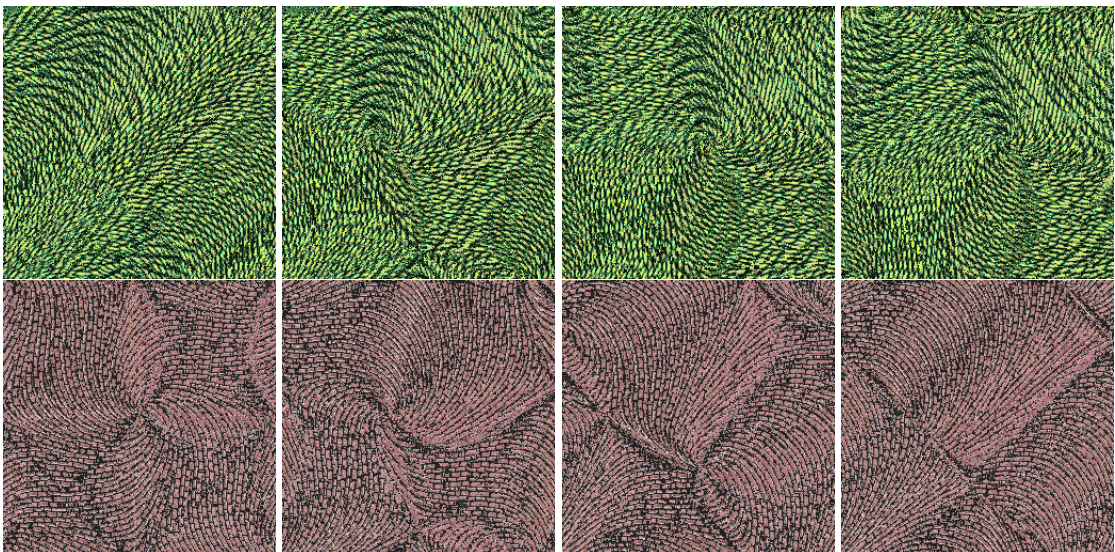


Figure 7.20: Example of a varying grey-scale and bricks-like texture patterns.

### 7.3 Concluding remarks

Concluding, this chapter outlines a technique for the synthesis of non-homogeneous varying textures and for the generation of continuous texture animations. Interesting applications include decoration of deformed surfaces or surfaces in motion such as cloths or other materials. Textures are a valid solution for helping shape and material perception, and local control during the generation and variation process is fundamental to augment such features.

The method is simple and general, but also flexible and capable of generating a variety of effects. User intervention is offered: it is possible to produce specific output sequences through easy and intuitive setting of control parameters and formulæ. The synthesized output frames conserve appearance and structural properties similar to the input sample. They are controlled through a

specified vector field: they have to be adapted locally to follow specified directions, and accordingly vary their structure and attribute in a non-homogeneous way. Resolution of the original texture pattern, as well as color or shading attributes, can be easily varied. This synthesis procedure, being pixel-based, does not run at interactive rates. The advantages, on the other hand, include the smoothness of the outputs and the presence of extra degrees of freedom in the texture synthesis process, since local control has effect over individual pixels. In conclusion, the main contribution of this work is to offer a general and intuitive technique to synthesize variable textures and later to animate them. In this way, a broad variety of pattern variation can be described. Starting from such concepts, it is possible to extend and apply the presented ideas to several fields of interest, also integrating different concepts together. I briefly discuss some possible remarkable cases in the following.

### 7.3.1 Texture mixture and metamorphosis

Sometimes in the real world, a single object exhibits more than one uniform look: the object may be built up of different materials and its appearance may derive from a mixture of several different appearances. Composing together dissimilar aspects in the same object is an important task with many applications in texturing natural 3d objects. Interesting traditional examples of metamorphosis are those by M. C. Escher (*e.g. Metamorphosis III* [59]). Applying the proposed method, this task can be solved without the need of blending functions. Nevertheless, the use of blending could serve to produce a user-designed continuous set of input samples. Morphing, as well as changing the aspect of textures depending on surrounding circumstances or lighting effects, is thus another interesting point to investigate in. Possible extensions can be achieved by investigating new kinds of filters and various artistic effects (§ 6.5.3) in order to further differentiate and map particular samples and appearances to different regions of the output. Transitions and blending transformations between different images also need to be further explored.

### 7.3.2 Solid textures

Generating solid textures is of interest in general for texture synthesis applications, but also in particular in the material and biological sciences [94]. A precise quantitative characterization of heterogeneous materials is needed to study structures that are built or grown from these materials. As alternative to producing temporal texture frames, also spatial texture slices can be produced using the same approach (Fig. 7.21).

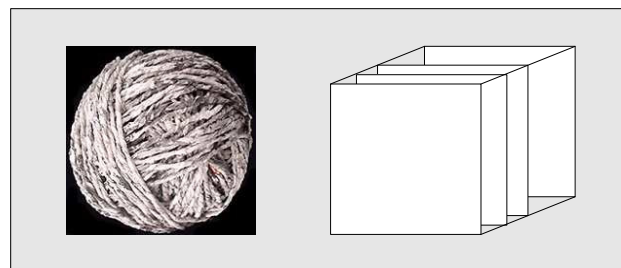


Figure 7.21: Solid textures can be obtained generating spatially consistent texture slices.

### 7.3.3 Inpainting

As extension to steerable texture synthesis, *controllable inpainting* can be performed. The technique of *inpainting* usually attempts re-synthesizing a portion of an image where a little area is missing or has been damaged [20]. Inpainting also can be used to remove unwanted little objects from a scene, replacing them with the background. Using the proposed steerable technique, an

interesting extension could be to remove a portion of a texture and replacing it synthesizing a user-designed or field-driven patch as substitute. The missing region is thus constrained by specified conditions and can be extrapolated using the steerable texture synthesis. The pixel-based synthesis still can guarantee continuity at the borders or boundary conditions where the patch is cut and later pasted; this can be optimized using a *spiral order synthesis* instead of *scan line order synthesis* as proposed in standard image inpainting (Fig. 7.22).

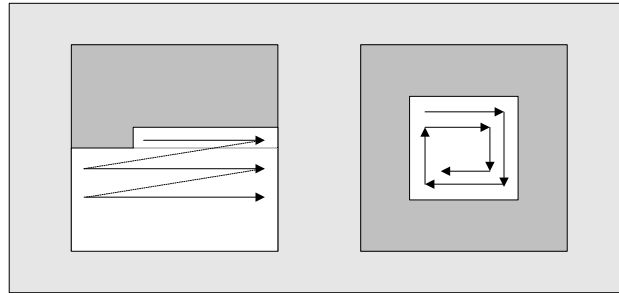


Figure 7.22: *Scan line order* for causal image generation (left) vs. *spiraling order* synthesis for steerable inpainting (right). In both illustrations, the dark grey part of the image is already synthesized while the white parts need to be synthesized.





## Chapter 8

# Conclusions

In this work, a novel approach to scientific visualization and steerable texture synthesis is presented. Vector field visualization and synthesis techniques for controlled, field-driven texture generation are proposed, discussed, and extended to allow more control and degrees of freedom in the image creation. Concepts from perception and cognition, as well as statistical theory for standard texture synthesis, were investigated and used to motivate and improve the proposed techniques. The approach results to be general, flexible, and open to further extensions and integrations.

The approach to visualization of vectorial data sets can be interpreted as an hybrid algorithm, which combines features of direct intuitive visualization, such as simplicity, intuitivity, generality, together with features from dense visualization techniques, such as powerful information encoding, locality of calculation, accuracy. The variety of tasks and the different level of expertise and experience of users motivate and strongly require such versatility.

The steerable generation of non-homogeneous textures offers several degrees of freedom in the synthesis process, allowing a variety of effects for appealing output images. Textural elements used as primitive provide for this task a numerous set of visual dimensions for arbitrary variation.

In general, the proposed techniques bring together concepts from human vision and perception, statistics and texture synthesis, and visualization, in an interesting interdisciplinary research that promises encouraging results for several fields of applications, offering a number of open directions for future work.

### 8.1 Summary and contributions

Vector fields can be visualized in a straightforward manner by setting the color of output pixels on the base of computed similarity functions. The proposed method is texture-based, but instead of blurring and smearing an initial noise texture along the directions of the field, it locally adapts and transforms a chosen basic pattern of an anisotropic texture to represent the features and the variation of the field. Consequently, unlike other methods, this one offers arbitrary degrees of freedom in representing the appearance of the resulting field. Continuity is achieved thanks to a statistical approach based on neighborhood comparisons, which is here adapted to incorporate spatial as well as temporal information in order to guarantee correlation; hence, also areas of strong curvature can be accurately visualized. To incorporate the benefits of feature-based approaches, field analysis is provided to extract and better detect special attributes, such as singularities, and to potentially track them along the field evolution. In addition, user options contribute to local control, to allow observing and interpreting particular values in the data in a user-customized way. This adds useful flexibility to visualization of multivariate vector fields. In general, the proposed method allows any procedural or manual way to define a mapping from vector space to example image space. Additional degrees of freedom offer ways for user intervention (*e.g.* filtering and blending) or taking into consideration perception issues. This also makes this methodology useful and adaptable for a broad range of applications. For all these reasons, I use the term *steerable*, as

the process of synthesizing textures is driven by the content and by the features of the vector or deformation field to visualize.

The main novelty and contributions to the field of computer graphics are here summarized:

### Vector field visualization:

The proposed algorithm allows smooth continuous visualization of vector fields, both steady and unsteady. The approach works particularly well for multi-parameter vector fields, and can be applied to scalar, vector and tensor field visualization, being in general designed for multi-variate multi-dimensional data sets.

- *Local control*: a novel algorithm has been introduced; it provides local control and several degrees of freedom for the visualization of vectorial data sets. The pixel-based synthesis algorithm allows a direct control onto the output values of the resulting image, up to every pixel. Non-ambiguous one-to-one mapping leads thus to effective local control. See Chapter 3. A multi-resolution approach, based on image pyramids is applied for improvement (see Appendix A).
- *Prominent field attributes*: improvements have been proposed to allow accurate representation of information also in particular areas of interest, such as singularities or critical areas of strong curvature. For this purpose, *ad hoc* specification of the input textural elements has been done to improve the results, guaranteeing smoothness and accuracy. See Chapters 3, 6.
- *Multivariate data*: visualization of multiparameter data and flexible information encoding is straightforward using the proposed approach. Complicated multi-dimensional and multi-variate data sets, as well as higher order fields, can be visualized. The synthesis technique allows powerful information encoding taking advantage of the texture visual dimensions. See Chapter 5, 6. *Unsteady field visualization*: extensions for temporal evolution of vectorial data are presented, and frames animation is possible using an adapted simple scheme.
- *Multi-field visualization*: the approach has been also extended to the visualization of multi-fields, and especially dual-fields, generating an effective interwoven representation of two co-existent vectorial data sets. See Section 5.3.

### Texture synthesis:

- *Neighborhood models and weighting schemes*: In this new approach to constrained texture synthesis, as opposed to standard texture synthesis, novel neighborhood models, together with appropriate filtering functionals and weighting schemes have been proposed to enhance texture feature of directionality and to preserve texture pattern complexity. See Chapter 5, 4.
- *Non-homogeneous textures*: Non-homogeneous textures can be easily designed following users' needs and tasks requirements; blending and texture metamorphosis can be integrated. The use of *filter banks*, as well as that of a *matrix of input texture seeds* have been introduced. See Chapter 7.
- *Ad hoc output generation*: locality in the control of the image generation is provided: pre-processing, post-processing and on-line processing contribute to free and arbitrary generation of desired outputs. See Chapter 7.
- *Steerable texture synthesis*: Through a control vector field, a given texture pattern can be modified, transformed, deformed. Texture effects on shape and material perception can be easily customized. See Section 7.1. *Flow textures and texture animation*: extensions of the algorithm allow the generation of flow textures and a field-driven animation of textures. See Section 7.2.

**Perceptually motivated visualization:**

- *Intuitive feature correspondence*: the approach allows a simple way to map field attributes onto visual representation for adequate mapping and representation of the vectorial information. The textural elements used as seed in the visualization approach yield a particularly easy and effective feature specification and encoding. See Chapters 3, 6.
- *Design of visual primitives*: textures are well suited to serve as paradigm for effective visualization. Taking advantage of their various visual dimensions, expressive and meaningful information encoding is possible. In this work I propose and discuss some solutions, which can be interesting for several applications. Specially defining a texture input set is a powerful instrument for visualization. See Chapter 6.
- *Layer-based visualization*: taking advantage of the texture visual dimensions, layered visualization is a good solution for adaptive representation of information: for comparison between several data sets, for analysis of several co-existent distributions potential relative interference, as well as for data abstraction and simplification. Rates of transparency helps constraining the influence of the several levels of information. See Section 6.6.

The main benefits of the proposed approach are:

**Simplicity**

One significant technical contribution of this work is to present a single straightforward method to generate a wide variety of vector field images, ranging from dense texture-like to hand-drawing abstracted and sketched streamline-like styles. Indeed just the sample characterizing the resulting appearance has to be chosen or designed, while the essential structure of the algorithm does not change. The method is sample-based, thus every kind of sample image can be theoretically chosen. Obviously, as discussed and motivated in Chapter 3, depending on the particular case, a sample can yield a better result than another one. However, even though the results are of best quality for anisotropic ordered or quite structured textures, the method also works for stochastic textures. Although quasi-isotropic patterns are almost orientation-insensitive, their synthesis along a vector field may enhance their latent directions, making apparent unexpected orientation dependencies that exist, due for instance to subtle pattern structuring or to structures that stems from illumination process. The proposed steerable texture synthesis approach also allows a straightforward generation of a variety of effects, which are beneficial for instance for the synthesis of non-homogeneous and anisometric textures.

**Generality**

Since the algorithms simply set pixel colors, no restriction exists for the choice of the data set or functional of the control field. To test the generality of the algorithm under various conditions, I analyzed a number of different synthetic distributions, specially designed to characterize different vector field classes, also characterized by singularities, areas of strong curvature or rapid changes of direction. This has been done to test the algorithm also in cases some features, often critical to be visualized, occur. The final images effectively capture the input sample appearance and represent the vector field information. Regarding textures, I also have tested the approach using many different images from standard texture sets. The used MIT VisTex database contains for instance real world textures photographed under natural lighting conditions. Additional results have been achieved combining different textures together, to investigate more complicated and general cases. The use of textures in visualization proves to be a valid and flexible solution to several targeted visualization problems.

### Controllability and versatility

The method is more time-consuming than interactive LIC-based visualization methods, nevertheless it is comparable to texture synthesis based approaches, in addition being able to offer a wider range of encoding options and controls settings to influence the rendering of the resulting image with extra degrees of freedom. A nice feature of this method is the possibility to allow multi-way transitions among more than two textures, thanks to the use of a matrix of input seeds. The use of transfer functions and filtering operators over the sample images can be freely used, providing a great variety of potential texture transformations and effects.

## 8.2 Future work

Numerous are the possible future directions. This work is a first step to contribute to the synergy of visualization and many related disciplines of Computer Graphics. The interdisciplinarity of the research is open to several further investigations and extensions. The potential of this area of research has actually no limits like advances in computer graphics constantly show. Especially a deeper investigation in human vision could improve and optimize the design and specification of the proposed ideas and techniques, integrating them with more advanced theories. Parallel contributions from perception and cognition, as well as from psychology should stronger contribute to the effectiveness of visualization methods in general.

I see in particular the following points for future work and believe they reserve interesting applications:

### Handling advanced topology

Beyond existing vector field visualization methods, only feature-based approaches allow a topological analysis of the data set. With the proposed approach, eigen-analysis is performed and integrated to augment the field representation with its characterizing topological features. The ability to map singularities onto arbitrary suitable representations, also taking advantage of the layered approach, contributes to intuitive representation, whereas the most approaches only work for distributions with a limited set of singularities and need topological simplification to handle more complicated cases [42, 217, 245]. Nevertheless, more advanced field analysis [243, 244], such as considering higher order singularities, could be taken into account for integration in the proposed algorithm, making it more sophisticated and allowing it to automatically deal with higher order mathematical aspects and attributes, for a more rigorous and specialistic analysis of the data sets. Taking into consideration such aspects of the data, that are usually disregarded by the most visualization approaches, could be a relevant contribution. Furthermore, given a pre-processing extraction or selection of such features, their visual representation would add no additional complexity, since the algorithm still simply works setting feature correspondences and output pixel colors. For example, the evolution and variation of singularities over time, when visualizing unsteady vector fields, has drawn increasing attention in the last years. *Time-dependent topology* deals with changes that occur, over time, between stable states. Such changes are called *bifurcations*. Some of them, called *local bifurcations*, only affect the nature of a singular point or a closed orbit and the corresponding new stable state is found in the neighborhood. Among local bifurcations, *Hopf bifurcations*, *pairwise annihilations*<sup>1</sup> and *fold bifurcations* are the most prominent examples. Others, called *global bifurcations*, entail significant changes in the global structure of the flow and, involving large domains, cannot be deduced from local information. An example of such class is given by *basin bifurcations*. Refer to [218, 208, 209] for more detail. Vortex regions are further special features of interest to eventually consider. There exist several methods to find *vortex regions* and *vortex cores*<sup>2</sup> (refer to [155] for a detailed distinction). One possibility

<sup>1</sup>Two singularities with opposite indices

<sup>2</sup>The integral curve within a vortex that has a minimum curvature.

is to search for regions of low pressures [164]. Alternatively, Jeong and Hussain [96] define a vortex as a region where two eigenvalues of the symmetric matrix  $S^2 + S'^2$  are negative, where  $S$  and  $S'$  are the symmetric and antisymmetric parts of the Jacobian of the vector field, respectively:  $S = \frac{1}{2}(J + J^T)$  and  $S' = \frac{1}{2}(J - J^T)$ . This method is known as the  $\lambda_2$  method. Recent research [169, 207] has provided solutions to extract and track vortex core lines, also in time-dependent 3D flow fields.

### Output refinement

The results shown in this thesis are purely obtained using the MRF-based algorithm. To better illustrate the algorithm itself, no kind of refinement was used, but in case it is desired to smooth the resulting images, improvements can be achieved using simple directional low-pass filtering (as done for instance in several visualization approaches, as LIC, LEA, IBFV), to improve the vector field along the flow lines, improving spatial and temporal image correlation. This refinement process can be particularly useful to ulteriorly smooth temporal frames, increasing continuity along their sequential animation. One could need this postprocessing step when using sample images with coarse structure, which may cause artifacts or aliasing, noticeable in areas of strong curvature of the field, or where the curvature radius is larger than the resolution of the sample. Several smoothing filters are possible for such improvement: LIC can remove the effects of artifacts while preserving and enhancing the directional correlation resulting from the controlled synthesis phase. In this way, when necessary, potential discontinuities can be disguised. Non-linear diffusion techniques can be considered: the image is smoothed in the direction of the field, whereas it is sharpened in the orthogonal direction. Obviously, further solutions for optimal filtering can be further investigated.

### Advection

The LEA method proposed by Jobard [98] combines the advantages of the Lagrangian and Eulerian formalisms for continuous visualization of flow fields. Briefly, a dense collection of particles is integrated backward in time (Lagrangian step), while the color distribution of the image pixels is updated in place (Eulerian step). Briefly, a dense collection of particles is represented by a noise texture, and is then transported according to the motion of the particles<sup>3</sup>. LEA method achieves very good results at interactive rates, A combination with our method would possibly take advantage from LEA efficiency, still offering the rich variety of texture-based information encoding. A problem to address is anyway the potential texture distortion and spatio-temporal aliasing effects that could occur during the advection process, being then eventually visible during the frames animation. A possible solution could be to periodically reset texture coordinates (fading up and down mechanism for coordinate re-initialization) with respect to the texture structure, constraining texture distortion into bounds, in case the visible distortion should exceed the limit, which is proportional to the effective flow distortion. A combination with the LEA method would allow to visualize particle flow using the MRF-based algorithm.

### Integrating texture metamorphosis

In the presented work, the appearance of the resulting output is directly determined by the set of samples chosen as input patterns. The variety of such inputs provides non-homogeneous images. Typically, I make use of a pre-defined matrix of samples, or let the user choose a set of rules to condition the behavior of a set of filter banks, which operate over original samples in a progressive continuous way. Texture mixture and metamorphosis can therefore be achieved using uncorrelated samples and letting one transit into further image targets. To achieve more sophisticated results,

<sup>3</sup>The Lagrangian coordinate for the texture transport can be computed using a numerical scheme for convection equations [189]:  $\frac{\partial \rho(\mathbf{x}, t)}{\partial t} + \mathbf{v}(\mathbf{x}, t) \cdot \nabla \rho(\mathbf{x}, t) = 0$ , where  $\mathbf{v}$  is the vector field,  $\rho(\mathbf{x}, t)$  is the property field, where property values representing the particles are stored.

hardware-based blending methods could be used to let samples vary in a user defined way, also allowing to choose between a large variety of intermediate appearances. The method of Matusik *et al.* [137] could be for this scope integrated. The work of Liu *et al.* [131] is specially valid to morph textures containing discernable and similar patterns and could be also used to generate morphing sequences for the input set. Nevertheless, this method still needs user experience to recognize and select the texture features, in order to establish the right morphing correspondences.

### 3D Extension

Texture synthesis has revolutionized the construction of texture maps and the application of textures to surfaces. Nevertheless, problems still exist when texturing surfaces. *Texture mapping* over surfaces particularly requires textures to be adequately designed and modelled to match the geometry of the three dimensional object. For this reason, local control over the texture can allow more precision and flexibility for the mapping. Furthermore, being able to produce several sorts of variations in a texture consents to avoid the geometric simulation of such features and visual details, still providing a realistic and detailed resulting output.

**Texturing surfaces:** Early studies by Interrante [90, 72] generate custom-fitted textures. Such studies show how the orientation of anisotropic patterns, with respect to the lines of maximum and minimum normal curvature over an object surface, may affect an observer's perception of the underlying surface shape<sup>4</sup>. Texturing a surface can not only profoundly enhance its visual richness, but can also significantly affect the perception of the object geometry. Finding principal directions in arbitrary double curved surfaces is not always trivial [73], and seams, projective distortions, repetition artifacts should be avoided. Researchers in perceptual psychology have been investigating the effects of various texture pattern characteristics on surface shape perception through controlled observer experiments [93]. In order to adapt the method proposed in this thesis to generate textures on arbitrary surfaces, a possibility could be to extract critical points and curvature values from the mesh, and use these set of points to control the change of parameters along the texture, the vector field, or the flow [168]. Further valid approaches to estimate curvature and find critical points are those by Interrante [92], Alliez [8] and Rusinkiewicz [168], or methods based on *Morse theory*<sup>5</sup> and Reeb graphs [191]. Inconsistencies due to surface curvature could be then avoided using the presented approach, which can arbitrarily adapt the appearance and structure of the sample, for instance using blending or other filtering operators and transformations, such as scaling, over the sample.

**3D Textures:** Extensions to the three-dimensional domain could be implemented. A possibility is to use 2d slices as done for the temporal animation, in this case adding the third spatial dimension  $z$  to achieve slice-based 3D textures. The extension to the three-dimensional domain finds applications not only for textures, but in the same way also for vector field visualization over surfaces and 3D vector fields. Issues of clutter and occlusion should be considered and could be addressed for instance by selectively fading out uninteresting regions, using filtering and layering.

### Discriminability and uniqueness of information encoding

Deeper investigation and cooperation with human vision theory could improve the concepts used here to guarantee a 1-to-1 mapping of field features onto adequate visual representations. Especially, considerations on color space and optimal color gamuts need deeper inquiry. This will contribute to an improved, more rigorous specification of the entries of the input matrix, the filtering functions, the transformation operators, which control the output generation. Perception issues are fundamental to guarantee the human eye to distinguish between different encoding, to

<sup>4</sup>Recent studies in vision research support the idea that the principal directions play an important role in surface shape understanding.

<sup>5</sup>Morse theory [140] bases on early ideas developed in the context of topography. In differential topology, Morse theory gives a direct way of analyzing the topology of a manifold by studying differentiable functions on that manifold, obtaining information about their homology.

avoid redundancy, confusion or interference, and to recognize meaningful correspondences to data features. Human theory itself is a field that still bases its theories in the past fifties, but promising research is recently being conducted and reserves great potential for integration in computer vision applications. Hence, further investigation in this direction is of particular interest.

### **Fast implementation**

As explained in the implementation section (§ 3.4), the main drawback of the proposed approach is represented by its performances. Although the algorithm processing time is comparable with those of standard pixel-based approaches to texture synthesis, it suffers from that intrinsic computational slowness. For this reason, improving and speeding up the algorithm could be a point to further work on. Nevertheless, discussion with scientists from the fields of engineering, and especially aerospace engineering, fluid dynamics, but also magnetics and electrics, proved that often, in practice, the rendering of vector fields do not necessarily need interactive rates in the visualization (what we are used to in computer graphics); while the option of powerful mapping and flexible visual representations are a more critical and fundamental requisite. Consequently, in most practical visualization applications, images can be pre-computed and later rendered.





## Appendix A

# Multi-resolution synthesis

### A.1 Progressive refinement

In Chapter 3, for clarity and brevity, the algorithm implementation is first shown in single resolution. However, principles of multi-resolution image generation from image processing can be used. Such theory has been successfully applied to texture synthesis; this guaranties to achieve similar results to those obtained using single resolution, though requiring smaller neighborhoods and hence reducing the computation time. The *multi-resolution* approach to image generation can be briefly summarized as follows. 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: successive spatial frequency bands from the input example are sampled. This process is repeated until the last and finest level of the example is reached, as it is done in painting processes adding brush strokes in multiple passes. In this way, this synthesis procedure is helpful to capture image features on several scales. For pixel-by-pixel synthesis methods, the size of the neighborhood depends on the structure of the texture we want to synthesize: the neighborhood has to comprise enough information in order to be able to reconstruct the texture pattern in the output image. Using multi-resolution, smaller neighborhoods produce analogous results to larger ones in single-resolution. More details about the multi-pass approach and relative smoothing filtering are given below.

### A.2 Image pyramids: analysis and synthesis steps

A linear image transform represents an image  $I(x, y)$  as a weighted sum of *basis* functions. The image is thus represented as a sum over an indexed collection of functions  $g_i(x, y)$ :

$$I(x, y) = \sum_i y_i g_i(x, y) \quad (\text{A.1})$$

where  $y_i$  are the transform coefficients that are computed from the image signal by projecting it onto a set of *projection* functions  $h_i(x, y)$ :

$$y_i = \sum_{x, y} h_i(x, y) I(x, y) \quad (\text{A.2})$$

In many image processing applications, an image is decomposed into a set of subbands, and the information within each subband is processed more or less independently of that in the other subbands. The subbands are computed by convolving the image with a bank of linear filters. Each of the projection function is a translated (or shifted) copy of one of the convolution kernels [180, 33]. An *image pyramid* is a particular type of subband transform. The defining characteristic of an image pyramid is that the basis or projection functions are translated and dilated copies (by

a factor of  $2^j$  for some integer  $j$ ) of one another. The subbands are computed by convolving and sub-sampling. For each successive value of  $j$ , the sub-sampling factor is increased by a factor of 2. This yields a set of subband images of different sizes - hence the name *image pyramid* - that correspond to different frequency bands. In an independent context, mathematicians developed a form of continuous function representation called *wavelets* that are very closely related to image pyramids. Both wavelets and image pyramids can be implemented in an efficient recursive manner [83].

The *Image Pyramid* is a hierarchical structure composed of  $L$  levels of the same image at different resolutions. An image with the highest resolution - *detail* - is at the bottom level of a pyramid, while an image with the coarser resolution - *large-scale features* - is at a higher pyramid level (refer also to [83, 26, 154] for more details on multi-resolution sampling theory). Image pyramids are obtained in the simplest case using a low-pass filtered version of the original image. In summary thus, the procedure is a multi-resolution synthesis process based on *sub-band transforms*, and the pyramid is composed by a multi-scale set of image levels.

There are two basic pyramid operations [34], namely, *reduce* and *expand* (refer to the pseudocode listed below in Table A.1). The *reduce* operation involves filtering and *down-sampling* an image to obtain a new image at the *coarser resolution*. The *expand* operation involves *up-sampling* and interpolating an image to obtain an image at the *finer resolution*. The process of constructing a pyramid from an image is called *decompose*, which involves continuous *reduce* operations. While the process of obtaining an image from a pyramid is called *collapse*, which involves continuous *expand* operations (Fig. A.1).

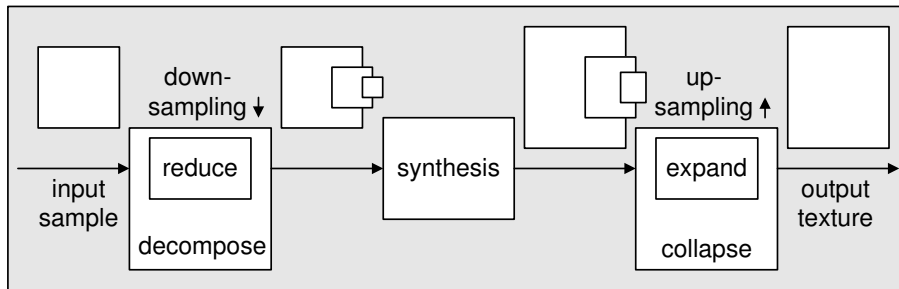


Figure A.1: Illustration of the multi-resolution synthesis, using *reduce* and *expand* operations.

Different pyramids correspond to different trade-off between spatial and frequency resolution. The *Gaussian* [153, 241], *Laplacian* [33, 83], *steerable* [83, 182] and *feature-based* [26] pyramids are among the most popular ones used in image processing. Gaussian and Laplacian pyramids respectively involve Gaussian and Laplacian-of-Gaussian base functions, while steerable pyramid uses wavelet transforms for decomposing an image. In the pyramid-based texture synthesis, the Laplacian and the steerable pyramids decompose a texture into multiple bands of spatial frequencies and orientations respectively. Each pyramid level represents certain texture features at a particular frequency or orientation. The histogram of each pyramid level is chosen as a descriptor of the related features. The synthesis is based on the idea that a new texture could be generated by matching all the available features (histograms) with the training texture [83]. For the research done in this thesis, I considered Gaussian, Laplacian and Steerable pyramids for the decomposition of vector field and textured images. Anyway, although steerable pyramids could be better suited for directional samples, being orientation sensitive, they are calculation intensive, since the orientation of the patterns constantly varies in the applications considered in this thesis. The Gaussian case results to be the best trade-off between complexity and efficiency. Gaussian *reduce* and *expand* only require simple up- and down-sampling and the last level of the output Gaussian pyramid directly provides the desired output, so that no final *collapse* operation is needed. In the following, for completeness, I provide an explanation of all three possible image decomposition approaches, spending more effort in the explanation of the Gaussian case and providing a detailed illustration for the extension of the basic algorithm of Chapter 3.

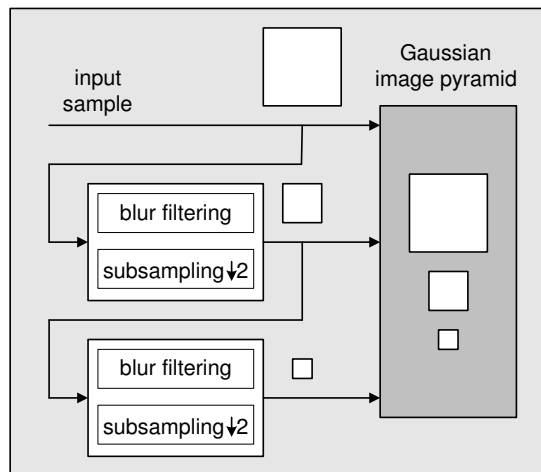


Figure A.2: Illustration of the multi-resolution synthesis, using `reduce` and `expand` operations.

### A.2.1 Gaussian image pyramids

The multi-resolution analysis and image pyramids can be simply implemented using *Gaussian* pyramids. Here, each level is obtained via successive filtering and *down-sampling* operations by a factor of 2 ( $\downarrow 2$ ); *i.e.* the images result to be blurred and decimated versions of the original one, through *low pass filtering*. Using Gaussian pyramids, the desired output image is simply available at the highest resolution pyramid level (Fig. A.2). Using image pyramid, a neighborhood  $N(x,y,l)$  of a pixel  $(x,y)$  at resolution level  $l$  in the pyramid, contains pixels of the same level  $l$  in the current resolution, and pixels from lower resolution levels (Fig. A.3). The position of the central pixel in the multi-resolution neighborhood is consistent with the parent levels ( $x$  vs.  $x/2$  and  $y$  vs.  $y/2$  in Fig. A.3). An extended multi-resolution neighborhood can be handled building *neighborhood vectors* that include the pixels of the causal and squared neighborhoods in scan-line order. Again, similarity between two multi-resolution neighborhoods is computed, for instance using the sum of the squared distances of all pixels within them, and these lower resolution pixels constrain the synthesis process so that the added high frequency details will be consistent with the already synthesized low frequency structures. In fact, in the output image each of its spatial frequency bands is generated so that, as higher frequency information is added, textural similarity is preserved.

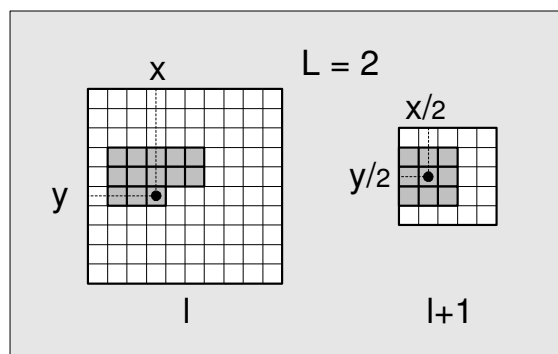


Figure A.3: Example of 2-level Gaussian pyramid. Level  $l + 1$  is obtained down-sampling by a factor of 2 ( $\downarrow 2$ ) previous level  $l$ . A causal multiresolution neighborhood with size  $\{5 \times 5, L = 2\}$  is considered.

In Tables A.1, A.2, A.3 below, I illustrate the modified version of the pseudocode presented in § 3.4, now implemented in multiresolution. Pyramids  $G_{I_m}$  of the input samples  $I_m$  are built in

Function synthesizePixel (multiresolution)	
1	for(l = 0; x < L; l++) {
2	for(x = 0; x < W <sub>out</sub> (l); x++) {
3	for(y = 0; y < H <sub>out</sub> (l); y++) {
4	N <sub>l</sub> (x,y) <sub>out</sub> = calculateNeighborhood(x, y, l);
5	{f <sub>(x,y)}</sub> = calculateFeatures(x, y);
6	transformSample({f(x, y)});
7	for(i = 0; i < W <sub>in</sub> (l); i++) {
8	for(j = 0; j < H <sub>in</sub> (l); j++) {
9	N <sub>l</sub> (i,j) <sub>in</sub> = calculateNeighborhood(i, j, l);
10	distance <sub>i,j</sub> = compareNeighborhoods(N <sub>l</sub> (x,y) <sub>out</sub> , N <sub>l</sub> (i,j) <sub>in</sub> );
11	}
12	}
13	minDistance = findMinimumDistance({distance <sub>i,j</sub> });
14	bestMatch = getBestPixelValue(minDistance);
15	G <sub>l<sub>out</sub></sub> (x, y, l) = synthesizeOutputPixel(bestMatch);
16	}
17	}
18	I <sub>out</sub> = collapse(G <sub>out</sub> );
19	}

Table A.1: Multi-resolution version of the synthesis procedure.

an *analysis* process, and a pyramid  $G_{I_{out}}$  of the output image  $I_{out}$  is generated during the *synthesis* process, proceeding from lower to higher frequency levels. In this way, each higher resolution level is constructed from already synthesized lower resolution levels. The considerations done for the algorithm in single resolution apply now separately to each level of the pyramid.

	Initialization
$L$	$\leftarrow$ setPyramidLevels( $L$ )
$G_{I_{in}}$	$\leftarrow$ buildPyramid( $I_{in}$ )
$G_{I_{out}}$	$\leftarrow$ buildPyramid( $I_{out}$ )
$W_{out}$	$\leftarrow$ setOutputWidth( $W_{out}$ )
$H_{out}$	$\leftarrow$ setOutputHeight( $H_{out}$ )
$n$	$\leftarrow$ setNeighborhoodSize( $n$ )
$I_{in}$	$\leftarrow$ selectInputSample( $I_{in}$ )

Table A.2: Initialization

### A.2.2 Laplacian pyramids

In *Laplacian* pyramid formulation [26, 83], *band pass* filtering is used. Also in this case, the pyramid represents the image at different levels of resolution, where each level carries information related to a given space of frequencies. At each level  $l$  of the pyramid  $L$ , frequency information at point  $(x, y)$  of the image  $I$  at level  $l$  is given by:

$$L_I(x, y, l) = (G_I(l) - 2 \uparrow [G_I(l+1)])(x, y) \quad (\text{A.3})$$

where  $G_I(l)$  is a low-pass down-sampling operation:

$$G_I(l) = 2 \downarrow [G_I(l-1) \otimes g] \quad (\text{A.4})$$

where  $2 \uparrow [\cdot]$  and  $2 \downarrow [\cdot]$  are the  $2 \times$  up- and down-sampling operations respectively,  $g$  is a two

Variable	Meaning
$I_{in}$	input sample image
$I_{out}$	output sample image
$G_{I_{in}}$	Gaussian pyramid built from $I_{in}$
$G_{I_{out}}$	Gaussian pyramid built from $I_{out}$
$L$	number of levels of image pyramid ( $0 \leq l < L$ )
$G(l)$	$l$ -th level of Gaussian pyramid $G$
$G(l, x, y)$	pixel at position $(x, y)$ and level $l$ of $G$
$W_{out}$	horizontal size of sample output image $I_{out}$
$H_{out}$	vertical size of sample output image
$N_l(x, y) _{out}$	extended (multiresolution) neighborhood of pixel $(x, y)$
$A$	vector field magnitude at current output position $P(x, y) _{out}$
$\theta$	vector field angle of phase at current output position
$W_{in}$	horizontal size of sample input image $I_{in}$
$H_{in}$	vertical size of sample input image
distance	difference between the extended neighborhoods
bestMatch	best match chosen by distance comparison

Table A.3: Table of symbols

dimensional Gaussian kernel or convolution mask, and  $G_l(0) = I$ . Each level of the Laplacian pyramid contains the information from a one octave spatial frequency band of the input [19].

The Laplacian pyramid is computed using the basic operations `reduce` and `expand`. The `reduce` operation applies a low-pass filter and then subsamples by a factor of two (padding with zeros in between pixels) and then applies the same low-pass filter. A commonly used low-pass filter kernel (applied separately to the rows and columns of an image) is:  $\frac{1}{16}(1, 4, 6, 4, 1)$  [83]. One complete level of the pyramid consists of two images,  $l^0$  (a low-pass image), and  $b^0$  (a high-pass image), that are computed as follows:

$$\begin{aligned} l^0 &= \text{reduce}(\text{im}) \\ b^0 &= \text{im} - \text{expand}(l^0) \end{aligned} \quad (\text{A.5})$$

where `im` is the original input image. Note that the original image can be trivially reconstructed from  $l^0$  and  $b^0$ :

$$\text{reconstructedImage} = b^0 + \text{expand}(l^0) \quad (\text{A.6})$$

The next level of the pyramid is constructed by applying the same set of operations to the  $l^0$  image, yielding two new images  $l^1$  and  $b^1$ . The full pyramid is constructed (via the `makePyramid` function) by successively splitting the low-pass image  $l^i$  into two new images  $l^{i+1}$  (a new low-pass image) and  $b^{i+1}$  (a new high-pass image). The combined effect of the recursive low-pass filtering and sub-sampling operations yields a subband transform whose basis functions are (approximately) Gaussian functions. In other words, the transform represents an image as a sum of shifted, scaled, and dilated (approximately) Gaussian functions. The projection functions of this transform are (approximately) Laplacian-of-Gaussian (mexican-hat) functions, hence the name Laplacian pyramid (Fig. A.4). Note that the pyramid is *not* computed by convolving the image directly with the projection functions. The recursive application of the `reduce` and `expand` operations yields the same result, but much more efficiently. In the end, we get a collection of pyramid subband images consisting of several bandpass images and one leftover lowpass image. These images have different sizes because of the sub-sampling operations: the smaller images

correspond to the lower spatial frequency bands (coarser scales). Note that the original image can always be recovered from the pyramid representation (via the `collapsePyramid` function) by inverting the sequence of operations, as exemplified above.

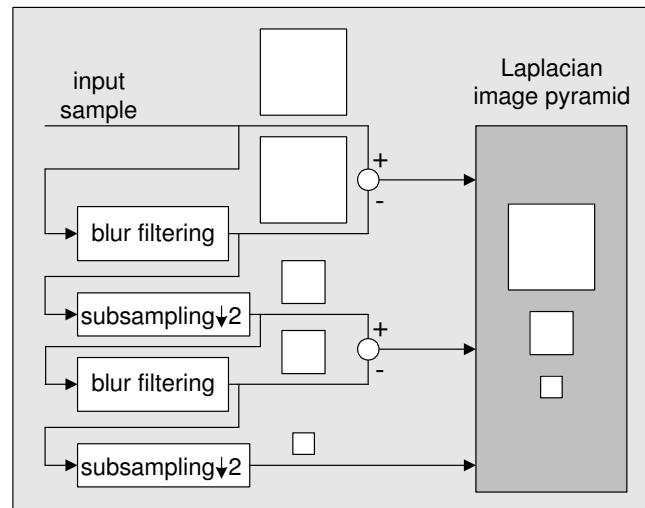


Figure A.4: Obtaining a Laplacian image pyramid.

### A.2.3 Steerable pyramids

Texture that have oriented or elongated structures are not captured by the Laplacian pyramid analysis because its basis functions are (approximately) radially symmetric. To synthesize anisotropic textures, it is possible to adopt the *steerable* pyramid transform [149, 181]. Like the Laplacian pyramid, this transformation decomposes the image into several spatial frequency bands. In addition, it further divides each frequency band into a set of orientation bands. In this way, information related to the orientation of the image features is taken into account. In Fig. A.5, the left-hand side of the diagram is the analysis part (`makePyramid`) and the right hand side is the synthesis part (`collapsePyramid`). The circles in between represent the decomposed subband images. The transform begins with a high-pass ( $H$ )/low-pass ( $L$ ) split using a low-pass filter with a radially symmetric response: the high-pass band corresponds to the four corners of the spatial frequency domain. Each successive level of the pyramid is constructed from the previous level's low-pass band by applying a bank of band-pass filters and a low-pass filter ( $B$  are oriented bandpass filters). The orientation decomposition at each level of the pyramid is steerable [65], that is, the response of a turned to any orientation can be obtained through a linear combination of the responses of the four basis filters computed at the same location. The steerability property is important because it implies that the pyramid representation is locally rotation-invariant. The steerable pyramid, unlike most discrete wavelet transforms used in image compression algorithms, is not-orthogonal and over-complete; the number of pixels in the pyramid is much greater than the number of pixels in the input image (note that only the low-pass band is sub-sampled). This is done to minimize the amount of aliasing within each subband. Avoiding aliasing is critical because the pyramid-based texture analysis/synthesis algorithm treats each subband independently. Improvements to reduce the redundancy have been proposed. The steerable pyramid is self-inverting: the filters on the synthesis side of the system diagram are the same as those on the analysis side of the diagram. This allows the reconstruction (synthesis side) to be efficiently computed despite the non-orthogonality.

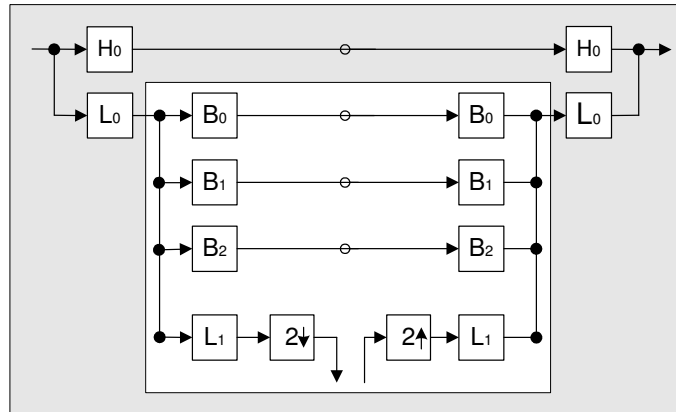


Figure A.5: System diagram for steerable image pyramid, redrawn from [83]. The input image is initially split into high- and lowpass bands. The lowpass band is then further split into a lower-frequency band and a set of oriented subbands [154].





# List of Figures

1.1	Using steerable texture synthesis for vector field visualization. . . . .	1
1.2	Generating controlled texture synthesis. . . . .	2
2.1	Figures and ground interchanging in the pictures. . . . .	6
2.2	Similarity in color and shape (left), proximity (center), and continuity (right). . .	6
2.3	Concepts of closure (left), area (center), and simmetry (right). . . . .	7
2.4	Joseph Albers' experiments: the same color (top-right) in the two central boxes looks different under different backgrounds (top-left); different colors (bottom-right) in the two central boxes look alike under some given backgrounds (bottom-left). . . . .	7
2.5	Color sensitivity (left) and CIE chromaticity diagram (right). . . . .	9
2.6	Treisman's preattentive experiments: target searching based on a difference of hue (up-left) and shape (up-right). In the bottom-left image, on the right, horizontal boundary detection is detected, while vertical boundary defined by conjunction of features (red circles and blue squares on the left, blue circles and red squares on the right) is not intuitive. In the bottom-right image, an example of a conjunction search for a target red circle is presented . . . . .	10
2.7	Galileo's text and illustration. . . . .	11
2.8	Synthesizing arbitrary resolution textures (right) starting from a little sample (left). . . . .	12
2.9	Samples of textures: regular (a), near-regular (b), irregular (c), near-stochastic (d) and stochastic (e). . . . .	13
2.10	Patch-based approach: Efros and Freeman's <i>image quilting</i> (image courtesy of Alyosha Efros). . . . .	16
2.11	Wei and Levoy's pixel based approach (image courtesy of Li-Yi Wei). . . . .	17
2.12	Hybrid texture synthesis (image courtesy of Andy Nealen). . . . .	18
2.13	Texturing surfaces (image courtesy of Greg Turk). . . . .	19
2.14	Wei and Levoy's surface texture synthesis: given a texture sample (a) and a model (b), they synthesize a similar texture directly over the model surface (c) (image courtesy of Li-Yi Wei). . . . .	19
2.15	Wei's solid textures from 2d view: (a) fire, (b) smoke, (c) ocean waves (image courtesy of Li-Yi Wei). . . . .	19
2.16	Texture transfer (image courtesy of Michael Ashikhmin). . . . .	20
2.17	Efros and Freeman's <i>texture transfer</i> : they transfer the rice texture (left) onto another image (center) for a strikingly different result (image courtesy of Alyosha Efros). . . . .	21
2.18	Direct visualization. . . . .	23
2.19	Grid-seeded streamlines (left) and optimized placement (right) (image courtesy of Greg Turk). . . . .	25

2.20	Spot noise (image courtesy of Jack van Wijk). . . . .	26
2.21	Texture-based techniques: Image based flow visualization (image courtesy of Jack van Wijk). . . . .	26
2.22	Left: LIC-based vector field. Right: frames obtained varying the opacity value of the advected noise array (image courtesy of Bruno Jobard). . . . .	28
3.1	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. . . . .	31
3.2	Basic visualization block scheme. . . . .	32
3.3	Correspondences between sample images and vector field values lead to a meaningful representation of the data set. . . . .	32
3.4	Steerable texture synthesis approach to vector field visualization. . . . .	33
3.5	Standard pixel-based approach to texture synthesis: the pixels $P_i$ of the output $I_{out}$ are set checking the most probable pixels in the input $I_{in}$ . . . . .	34
3.6	Locality and stationarity properties in textures (image courtesy of Marc Levoy). . . . .	35
3.7	Different input appearances lead to different appearances in the resulting vector field. . . . .	36
3.8	Possible set of grey-scale anisotropic patterns used as input seeds. . . . .	37
3.9	A sub-set of the VisTex texture database. . . . .	38
3.10	A sub-set of the MeasTex texture database. . . . .	38
3.11	Correspondences (potentially bijective) between samples and vector field values. . . . .	39
3.12	Synthesis process (cf. Fig. 3.5) using the matrix of samples seeds. . . . .	40
3.13	Pyramid levels for the multiresolution synthesis process of the image in Figure 3. . . . .	41
3.14	Correspondence between samples and vector values: transforming a sample through scaling and rotating operators. . . . .	42
3.15	Rotated sample input images. . . . .	43
3.16	The rotating operator over the sample, generates <i>diamond</i> structures, whose border windows need to be cut away. . . . .	43
3.17	Scaled sample input images. . . . .	43
3.18	The resizing operator over the sample image (left) re-scales it by a factor $A$ , generating sample versions in different resolution (left: $A = 1$ , center: $A > 1$ , right: $A < 1$ ). . . . .	44
3.19	Transforming the input sample $I_{in}$ to $I_{out}$ using a field-driven transfer function $T(I_{in}, \Phi(x, y))$ (top), and example of rotation by 90 degrees as particular operator $T_R$ , where $\phi_R(x, y)$ represents the field feature of orientation, or angle of rotation, at a given output position $(x, y)$ (bottom). . . . .	45
3.20	Generalization of Figure 3.19 to generic transfer functions. This general scheme illustrates a set of transformation operators (rotation, up-scaling, re-coloring) over the input sample image $I_{in}$ . Correspondences are set between transfer functions $T_i(I)$ and the field variables $\phi_i(x, y)$ taken as parameters. . . . .	45
3.21	Two possible filter bank schemes for independent and iterative sample transformation. . . . .	46
3.22	Causal or L-shaped neighborhood structure. . . . .	47
3.23	Samples characterized by structures with different complexity require different neighborhood sizes. . . . .	47
3.24	Building toroidal neighborhoods (right-left and up-down). . . . .	48

3.25	Circular and elliptical (§ 4.3) weighting schemes. . . . .	49
3.26	Proposed approach: unlike basic texture synthesis approaches (Fig. 3.5), the best matching pixels are here derived from diverse samples that resembles the vectorial information. . . . .	51
3.27	Visualization of critical points. . . . .	52
3.28	Different parts of the vector field visualized using the same output resolution. . .	53
3.29	Using a small scale example texture might lead to aliasing artifacts in the synthesized visualization. . . . .	54
3.30	Synthesized outputs: two vector fields are obtained both using two sets of different input sample images chosen from Figure 3.8 . . . . .	55
3.31	Examples of synthesized vector fields. . . . .	56
4.1	Standard texture synthesis <i>vs.</i> controlled texture synthesis: the controlled, field-driven synthesis algorithm requires the input pattern to be synthesized along new directions. . . . .	61
4.2	Standard texture synthesis <i>vs.</i> controlled texture synthesis: the pixels $P_i$ of the output $I_{out}$ are set checking the most probable pixels in the input $I_{in}$ (left); the best matching pixels are derived from diverse samples (right). . . . .	62
4.3	Different complexity in the structure resolution of the example textures. . . . .	63
4.4	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. . . . .	63
4.5	Circular and elliptical weighting schemes. . . . .	65
4.6	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. . . . .	66
4.7	Directional enhancement using the elliptical weighting scheme. . . . .	67
4.8	Elliptical weighting schemes, with different eccentricity and orientation (top); weighting schemes designed by composition of oriented elliptical Gaussians (bottom). . . . .	69
5.1	Vector field visualization enhanced by a brightness map. . . . .	76
5.2	Block diagram for the generation of one frame (left) and resulting example (right). . . . .	79
5.3	Block diagram (left) for visualization of unsteady fields frames (right). . . . .	80
5.4	Spatio-temporal coordinate axis system and 3d neighborhoods for $n=3$ , $n=5$ . . . . .	80
5.5	Spherical (left) and ellipsoidal (right) weighting schemes for 3D neighborhoods, with respect to the uniform scheme of Fig. 5.4. for $N=3$ , $N=5$ . . . . .	81
5.6	Example of temporal frames generation. A grey-scale sample is adapted locally to produce a sequence of frames for the time-varying vector field. . . . .	81
5.7	A set of generated temporal frames. . . . .	82
5.8	A set of generated temporal frames. . . . .	83
5.9	Screenshot of the blender application, the sliders are responsible for controlling the intensity of the encoding in the information masks (left). Generated frame sets with different rate of transparency (right). . . . .	84
5.10	Diffusion tensor image (DTI) visualized using ellipsoids (left) and concepts from painting (right) (image courtesy of David H. Laidlaw). . . . .	85
5.11	Left: a thick streamline (bottom) is constructed from a 1D streamline (top). Right: polygons are generated according to an adaptive step-size algorithm that allows for smaller polygons to be generated around areas of high curvature. . . . .	87

5.12	Left: using texture-mapped thick streamlines to visualize a flow field. Right: an illustration of texture outlining used to disambiguate streamline orientation. . . .	88
5.13	Illustration of using texture attributes to represent a scalar distribution. The scale of the texture is here related to Reynolds shear stress - a scalar field used to characterize regions where drag is generated in turbulent boundary layers. . . . .	89
5.14	Examples of the diversity of natural textures that can be applied to a vector field. A circular flow is used demonstrate each example. . . . .	89
5.15	Scheme block for simultaneous dual field representation. . . . .	90
5.16	Relative phase difference between the two vector fields at a given point. . . . .	91
5.17	Samples generation for dual fields representation. . . . .	92
5.18	Dual fields representation. . . . .	93
5.19	Dual fields representation. . . . .	93
6.1	Field features extraction and selection for data mapping during the synthesis stages. . . . .	96
6.2	Block scheme illustrating field features (including zero crossing, eigen-analysis, feature extraction and classification). . . . .	98
6.3	Center (a) vs. non-center singularities: general curvilinear sector (b), hyperbolic (c), parabolic (d) and elliptic (e) sectors (illustrations derived from [216]). . . . .	100
6.4	Critical point classification: repelling focus, repelling node, saddle point, center, attracting node, attracting focus (clockwise from top-left corner). . . . .	103
6.5	Repelling critical points: node source, focus source, improper node source, spiral source. . . . .	103
6.6	Attracting critical points: node sink, focus sink, improper node sink, spiral sink. . . . .	104
6.7	Higher order singularities: dipole and monkey saddle. . . . .	105
6.8	Generalized block scheme: augmenting the MRF-based approach. . . . .	106
6.9	A small potential texture palette. Scale increases along the horizontal axis, regularity increases along the vertical axis, and intensity increases along the left-to-right descending diagonal. . . . .	108
6.10	A block scheme representing a single variation of a given input sample using a transform operator, whose effect is specified according to an adequate feature correspondence, and which takes the vector field parameter as argument. . . . .	110
6.11	In this filtering scheme, the filter bank builds up of filters, which can be also uncorrelated and may modify the input sample in an individual way. . . . .	111
6.12	In this filtering scheme, <i>vice versa</i> , the filter bank builds up of filters, which are iteratively connected and progressively transform the input sample. . . . .	112
6.13	Convolution scheme relative to equation 6.20. . . . .	112
6.14	A possible texture palette obtained using image filters. . . . .	113
6.15	Left: compositing color with texture, right: color interweaving (image courtesy, respectively, of Haleh Hagh-Shenas and Timothy Urness). . . . .	115
6.16	Depth of field enhancing the chessmen that threaten the knight on e3. (image courtesy of Robert Kosara). . . . .	115
6.17	Examples of artistically inspired flow visualizations (image courtesy of Robert M. Kirby). . . . .	117
6.18	Kirby <i>et al.</i> experimented with varying the visual representation of underlying data by changing stroke shapes, texture, color, size, and placement. The top and bottom image in each pair are the same underlying data (image courtesy of Robert M. Kirby). . . . .	118
6.19	Pointillistic visualization (image courtesy of P. Coleman Saunders). . . . .	118

6.20	Using adaptive sliders to vary the visual representation of the data. . . . .	119
6.21	Medieval illustrations of Vitruvius's theory (left) and Leonardo's drawing with both the "homo ad circulum" and the "homo ad quadratum" (right) . . . . .	121
7.1	Standard pixel-based approach to texture synthesis (left): the pixels $P_i$ of the output $I_{out}$ are set checking the most probable pixels in the input $I_{in}$ . Proposed approach (right): unlike the basic approach, the best matching pixels are derived from diverse samples. . . . .	124
7.2	Variations along y-axis: coloring & changing direction (left); modifying cloth pattern sinusoidally (right). . . . .	126
7.3	Further results: curving an original pattern along a field (left) and gradually modifying color components over the image plane. . . . .	126
7.4	Scaling, rotating, brightening and darkening, incrementing red and blue component, embossing. . . . .	127
7.5	Controlled Texture Synthesis: the input sample (a) is modified through a superimposed field (b): phase information is used to change the direction of the pattern (c), magnitude information is considered in addition to scale the original pattern (d). . . . .	128
7.6	Transformation masks: enhancing brightness (a), scaling (b), scaling plus brightness (c), and embossing (d). . . . .	128
7.7	Further results: waves and bricks patterns. . . . .	128
7.8	Further filtering: increasing the red (a), green (b), blue (c) component along with field amplitude variation, in a local manner. . . . .	129
7.9	Applying different color masks: with (a, b) and without (c) magnitude scaling. . . . .	129
7.10	Further results. . . . .	130
7.11	Examples of a variety of synthesized textures. . . . .	131
7.12	Examples of a variety of synthesized textures. . . . .	132
7.13	Temporal regularity is exploited in animation of clouds, smoke, fire, steam, waves, waterfall. . . . .	133
7.14	Block diagram for the generation of multiple subsequent frames: their animated succession generates a time varying texture. . . . .	134
7.15	Synthesis schema for the generation of two successive frames ( $t = 0, t = 1$ ) using a neighborhood of size = 3. . . . .	135
7.16	Five-sized neighborhood model (L-shaped plus square-shaped) with two-level multi-resolution synthesis ( $L = 2$ ). Lighter colors in the synthesized frames means weaker correlation to the current one ( $t = 2$ ). . . . .	136
7.17	Examples of filtered texture frames sequences. In both cases, four temporal frames extracted from a longer sequence are shown. They represent the time-varying texture at different time instants. . . . .	137
7.18	A sinusoidal distortion field modifies a sample (top), and a filter based on a blue component additionally highlights the field intensity in the same frame series (bottom). . . . .	138
7.19	A <i>fabric</i> (top) and a <i>directional</i> (bottom) texture are controlled by a vector field. Scaling of the sample reflects magnitude information of the field; the resolution of the original sample is accordingly locally adjusted. . . . .	139
7.20	Example of a varying grey-scale and bricks-like texture patterns. . . . .	139
7.21	Solid textures can be obtained generating spatially consistent texture slices. . . . .	140
7.22	<i>Scan line order</i> for causal image generation (left) vs. <i>spiraling order</i> synthesis for steerable inpainting (right). In both illustrations, the dark grey part of the image is already synthesized while the white parts need to be synthesized. . . . .	141

A.1	Illustration of the multi-resolution synthesis, using <code>reduce</code> and <code>expand</code> operations. . . . .	152
A.2	Illustration of the multi-resolution synthesis, using <code>reduce</code> and <code>expand</code> operations. . . . .	153
A.3	Example of 2-level Gaussian pyramid. Level $l + 1$ is obtained down-sampling by a factor of 2 ( $\downarrow 2$ ) previous level $l$ . A causal multiresolution neighborhood with size $\{5 \times 5, L = 2\}$ is considered. . . . .	153
A.4	Obtaining a Laplacian image pyramid. . . . .	156
A.5	System diagram for steerable image pyramid, redrawn from [83]. The input image is initially split into high- and lowpass bands. The lowpass band is then further split into a lower-frequency band and a set of oriented subbands [154]. . . . .	157

# List of Tables

3.1	Synthesis procedure (specially designed for rotation and scaling transformations).	54
3.2	Pre-computation of field features and input samples for generalized synthesis procedure. Note that in this case the input samples $I_{in}(x,y)$ ( $W_{in} \times H_{in}$ ) are pre-computed transforming $I_{in}$ in dependance of the features of the vector field calculated at the output locations $(x,y)$ .	57
3.3	This function exhaustively compares the current output neighborhood with all neighborhoods within the corresponding input sample.	58
3.4	This function compares the current output neighborhood with those within the corresponding input sample until a specified threshold distance is reached.	58
3.5	Table of symbols	58
3.6	Table of functions	59
4.1	Controlled texture synthesis (fabric and green-scale patterns) along curved vector fields. Using a five-sized kernel, the proposed elliptical weighting scheme (right) shows significant improvements with respect to the circular and the standard squared neighborhood.	68
4.2	Obtained results for fabric, bricks, tissue, straw and green-scale patterns patterns.	69
5.1	Temporal synthesis procedure (consistent with <code>synthesizePixel-a</code> of Table 3.1).	82
5.2	Temporal frames generation and succession	82
5.3	Interwoven synthesis of two co-located vector fields. The pseudo-code is consistent with a simplified version of that from <code>synthesizePixel-a</code> of Table 3.1.	91
5.4	Table of symbols and methods	92
A.1	Multi-resolution version of the synthesis procedure.	154
A.2	Initialization	154
A.3	Table of symbols	155

## **Akademische Werdegang**

Francesca Taponocco studierte *Telecommunications Engineering* an der Fakultät für Ingenieurwissenschaft der Universität zu Pisa, in Italien. Sie absolvierte ihr Studium und bekam ihr Diplom als Ingenieur im Oktober 2000.

Seit 2001 arbeitete sie als Wissenschaftliche Mitarbeiterin innerhalb des Fachgebiets Graphisch-Interaktive Systeme (GRIS), im Fachbereich Informatik, an der Technischen Universität Darmstadt. Sie beschäftigte sich mit Informationsvisualisierung, Polarkurvenapproximation, sowie prozeduralen Modellierung natürlicher Prozesse. Sie spezialisierte sich dann in den Forschungsgebieten von Vektorfeldvisualisierung und Textursynthese, und präsentierte ihre Ergebnisse in verschiedenen internationalen Grafikkonferenzen. In 2005 erhielt sie den zweite Platz bei der Verleihung des "Best Paper Award" innerhalb des INI-GraphicsNet für ihre Forschungsarbeit.

Sie promovierte im Oktober 2006.



# Bibliography

- [1] Meastex. texture database for the measurement of texture classification algorithms, the university of queensland. <http://www.cssip.edu.au/meastex/meastex.html>.
- [2] Nonphotorealistic rendering in scientific visualization. Siggraph 2001, course # 32, <http://www.siggraph.org/s2001/conference/courses/crs32.html>.
- [3] Vistex database. vision and modeling group of the mit media laboratory. <http://vismod.media.mit.edu/vismod/imagery/VisionTexture/vistex.html>.
- [4] D. Acevedo, D. Laidlaw, C. Jackson, and F. Drury. Using visual design expertise to characterize the effectiveness of 2d scientific visualization methods. *Poster IEEE Visualization, 2005*.
- [5] J. Albers. *One Plus One Equals Three or More: Factual Facts and Actual Facts*. Search Versus Re-Search (Hartford, 1969).
- [6] J. Albers. *Interaction of Color*. Yale University Press, 1971.
- [7] M. Alexa and W. Müller. Visualization by examples: Mapping data to visual representations using few correspondences. In E. Gröller, H. Löffelmann, and W. Ribarsky, editors, *Data Visualization '99*, Eurographics, pages 23–32. Springer-Verlag Wien, May 1999.
- [8] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. Anisotropic polygonal remeshing. In *Siggraph'03*, pages 485–493.
- [9] A. A. Andronov, E. A. Leontovich, I. I. Gordon, and A. G. Maier. *Theory of Bifurcations of Dynamic Systems on a Plane*. Wiley, New York, 1973.
- [10] M. Ashikhmin. Synthesizing natural textures. In *Proceedings of 2001 ACM Symposium on Interactive 3D Graphics*, pages 217–226, March 2001. Research Triangle Park, North Carolina March 19-21.
- [11] A. Bair, D. H. House, and C. Ware. Perceptually optimizing textures for layered surfaces. In *Applied Perception in Graphics and Visualization, 2005*.
- [12] Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and M. Werman. Texture mixing and texture movie synthesis using statistical learning. In *IEEE Transactions on Visualization and Computer Graphics*, volume 7(2), pages 120–135. IEEE Computer Society, 2001.
- [13] J. Becker and M. Rumpf. Visualization of time-dependent velocity fields by texture transport. In D. Bartz, editor, *Visualization in Scientific Computing '98*, Eurographics, pages 91–102. Springer-Verlag Wien New York, 1998.
- [14] J. T. Behrens. Principles and procedures of exploratory data analysis. *Psychological Methods*, (2):131–160, 1997.
- [15] R. Behrens. *Design in the visual arts*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1984.

- [16] J. R. Bergen. Theories of visual texture perception. volume 10b, pages 114–134. Macmillan, New York, 1991.
- [17] J. R. Bergen and E. H. Adelson. Early vision and texture perception. *Nature*, 333:363–367, 1988.
- [18] J. R. Bergen and B. Julesz. Rapid discrimination of visual patterns. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:857–863, 1983.
- [19] J. R. Bergen and M. S. Landy. Computational modeling of visual texture segregation, July 1991. MIT Press, Cambridge MA.
- [20] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *SIGGraph-2000*, pages 417–424, 2000.
- [21] M. Bertalmio, L. A. Vese, G. Sapiro, and S. Osher. Simultaneous structure and texture image inpainting. In *CVPR*, pages 707–712. IEEE Computer Society, 2003.
- [22] J. Bertin. Semiology of graphics. *U. Wisconsin Press*, 1983.
- [23] P. Bhat, S. Ingram, and G. Turk. Geometric texture synthesis by example. In *Symposium on Geometry Processing*, pages 43–46, 2004.
- [24] J. F. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics SIGGRAPH '78 Proceedings*, volume 12, pages 286–292, Aug. 1978.
- [25] J. F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph*, 1(3):235–256, 1982.
- [26] J. S. D. Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. volume 31 of *Computer Graphics Proceedings, Annual Conference Series*, pages 361–368. ACM SIGGRAPH, Aug. 1997.
- [27] J. S. D. Bonet and P. Viola. Texture recognition using a non-parametric multi-scale statistical model. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, 1998.
- [28] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, July 1965.
- [29] P. Brodatz. *Textures: A photographic album for artists and designers*. Dover Publications, New York, 1966.
- [30] P. Brodatz. *Wood and wood grains: a photographic album for artists and designers*. Dover Publications, New York, 1966.
- [31] S. Brooks and N. A. Dodgson. Self-similarity based texture editing. *ACM Transactions on Graphics*, 21(3):653–656, July 2002. Proceedings of ACM SIGGRAPH 2002.
- [32] J. M. Buhmann, D. W. Fellner, M. Held, J. Ketterer, and J. Puzicha. Dithered color quantization. 17(3):C219–C231, Sept. 1998. Proceedings of Eurographics'98.
- [33] P. J. Burt and E. H. Adelson. The Laplacian pyramid as a compact image code. *IEEE Trans. Commun.*, 31(4):532–540, Apr. 1983.
- [34] P. J. Burt and E. H. Adelson. A multiresolution spline with applications to image mosaic. *ACM Trans. on Graphics*, (2):217–236, 1983.
- [35] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of SIGGRAPH 93*, Computer Graphics Proceedings, Annual Conference Series, pages 263–272, Aug. 1993.

- [36] R. Chellappa, R. L. Kashyap, and B. S. Manjunath. Model based texture segmentation and classification. In *Handbook of Pattern Recognition and Computer Vision*, 1997. Chapter II:2.
- [37] H. Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68:361–368, 1973.
- [38] M. F. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for image and texture generation. In J. Hodgins and J. C. Hart, editors, *Proceedings of ACM SIGGRAPH 2003*, volume 22(3) of *ACM Transactions on Graphics*, pages 287–294. ACM Press, 2003.
- [39] R. Crawfis and N. L. Max. Texture splats for 3D scalar and field visualization. In G. M. Nielson and R. D. Bergeron, editors, *IEEE Visualization*, pages 261–267. IEEE Computer Society, 1993.
- [40] G. Cross and A. K. Jain. Markov random field texture models. *IEEE Trans. Pattern Anal. Machine Intell.*, 5(1):25–39, Jan. 1983.
- [41] W. C. de Leeuw and R. van Liere. Spotting structure in complex time dependent flow. In H. Hagen, G. M. Nielson, and F. H. Post, editors, *Scientific Visualization*, pages 47–53. IEEE Computer Society, 1997.
- [42] W. C. de Leeuw and R. van Liere. Collapsing flow topology using area metrics. In *IEEE Visualization*, pages 349–354, 1999.
- [43] W. C. de Leeuw and R. van Liere. Visualization of global flow structures using multiple levels of topology. In *Joint Eurographics-IEEE TCVG Symposium on Visualization*, pages 45–52, May 1999.
- [44] W. C. de Leeuw and J. J. van Wijk. A probe for local flow visualization. In *IEEE Visualization '93 Proceedings*, pages 39–45. IEEE Computer Society, Oct. 1993.
- [45] W. C. 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, Oct. 1995.
- [46] U. Diewald, T. Preußner, 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.
- [47] J.-M. Dischler and D. Ghazanfarpour. A geometrical based method for highly complex structured textures generation. *Computer Graphics Forum*, 14(4):203–215, Oct. 1995.
- [48] J.-M. Dischler, K. Maritaud, B. Lévy, and D. Ghazanfarpour. Texture particles. *Computer Graphics Forum*, 21(3), 2002.
- [49] G. Doretto, E. Jones, and S. Soatto. Spatially homogeneous dynamic textures. In *Proceedings of European Conference on Computer Vision ECCV '04*, pages Vol II: 591–602, Prague, Czech Republic, may 2004.
- [50] I. Drori, D. Cohen-Or, and H. Yeshurun. Fragment-based image completion. In *ACM Siggraph*, pages 303–312, 2003.
- [51] R. C. Dubes and A. K. Jain. Random field models in image analysis. *Journal of applied statistics*, 16(2):131–164, 1989.
- [52] F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. *Proceedings of ACM Siggraph*, 21(3):257–266, 2002.
- [53] D. Ebert, K. Musgrave, D. Peachey, K. Perlin, and Worley. *Texturing and Modeling: A Procedural Approach*. Academic Press, Oct. 1994.

- [54] A. A. Efros and W. 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.
- [55] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *International Conference on Computer Vision*, pages 1033–1038, 1999.
- [56] J. L. Encarnação. Computer graphics visions and challenges: A european perspective. *IEEE Computer Graphics and Applications*, 26(4):83–89, 2006.
- [57] J. Encarnação, W. Straßer, and R. Klein. *Graphische Datenverarbeitung 1*, volume 1. R. Oldenbourg Verlag, München, 4. edition, 1996.
- [58] J. Encarnação, W. Straßer, and R. Klein. *Graphische Datenverarbeitung 2*, volume 2. Oldenbourg, München, 4. edition, 1997.
- [59] M. C. Escher. Metamorphosis iii. <http://mcescher.com/>.
- [60] D. W. Fellner and C. Helmberg. Robust rendering of general ellipses and elliptical arcs. *ACM Trans. Gr.*, 12(3):251–276, July 1993.
- [61] A. W. Fitzgibbon. Stochastic rigidity: Image registration for nowhere-static scenes. In *Proceedings of International Conference on Computer Vision ICCV '01*, volume 1, pages 662–670, Vancouver, BC, Canada, july, 2001.
- [62] K. W. Fleischer, D. H. Laidlaw, B. L. Currin, and A. H. Barr. Cellular texture generation. In *ACM SIGGRAPH*, pages 239–248, 1995.
- [63] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, and R. Phillips. *Introduction to Computer Graphics*. Addison-Wesley, 1993.
- [64] A. Fournier, D. Fussell, and L. Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, June 1982.
- [65] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(9):891–906, Sept. 1991.
- [66] W. T. Freeman, T. R. Jones, and E. C. Pasztor. Example-based super-resolution. *IEEE Computer Graphics and Applications*, 22(2):56–65, 2002.
- [67] K. Gaither, D. Ebert, D. Weiskopf, and P. Hanrahan. The visualization process: The path from data to insight. Panel, *ieee vis 2005*, pages =.
- [68] K. Gaither, B. Geisler, D. H. Laidlaw, and D. Ebert. Panel: In the eye of the beholder: The role of perception in scientific visualization. In *Proceedings of IEEE Visualization Conference*, pages 567–568, October 2004.
- [69] G. Galilei. *Istoria e dimostrazioni intorno alle macchie solari*. 1613. Roma, appresso Giacomo Mascardi.
- [70] H. Garcke, T. Preußner, M. Rumpf, A. C. Telea, U. Weikard, and J. J. van Wijk. A phase field model for continuous clustering on vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):230–241, 2001.
- [71] J. J. Gibson. *The Perception of the Visual World*. 1950. Houghton Mifflin, Boston, MA.
- [72] G. Gorla, V. Interrante, and G. Sapiro. Growing fitted textures. *SIGGRAPH 2001 Conference Sketches and Applications*, page 191, Aug. 2001.

- [73] G. Gorla, V. Interrante, and G. Sapiro. Texture synthesis for 3D shape representation. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):512–524, 2003.
- [74] B. Guo, L. Liang, C. Liu, H.-Y. Shum, and Y. Xu. Real-time texture synthesis by patch-based sampling. Technical Report MSR-TR-2001-40, Microsoft Research (MSR), Mar. 2001.
- [75] B. Guo, H. Shum, and Y.-Q. Xu. Chaos mosaic: Fast and memory efficient texture synthesis. Technical Report MSR-TR-2000-32, Microsoft Research (MSR), Apr. 2000.
- [76] P. E. Haeberli. Paint by numbers: Abstract image representations. In F. Baskett, editor, *Computer Graphics SIGGRAPH '90 Proceedings*, volume 24, pages 207–214, Aug. 1990.
- [77] C. Hansen and C. R. Johnson, editors. *The Visualization Handbook*. Academic Press, 2004.
- [78] R. M. Haralick. Statistical and structural approaches to texture. *Proceedings of the IEEE*, 67(5):786–804, May 1979.
- [79] J. W. Harris and H. Stoker. *Handbuch of Mathematics and Computational Science*. Springer, 1998.
- [80] C. G. Healey. Choosing effective colours for data visualization. In *IEEE Visualization*, pages 263–270, 1996.
- [81] C. G. Healey, K. S. Booth, and J. T. Enns. Visualizing real-time multivariate data using preattentive processing. *ACM Transactions on Modeling and Computer Simulation*, 5(3):190–221, July 1995.
- [82] C. G. Healey and J. T. Enns. Building perceptual textures to visualize multidimensional datasets. In *IEEE Visualization '98 (VIS '98)*, pages 111–118, Washington - Brussels - Tokyo, Oct. 1998. IEEE.
- [83] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. In *SIGGRAPH 95 Conference Proceedings*, pages 229–238. ACM SIGGRAPH, Aug. 1995. Los Angeles, California, 06-11 August 1995.
- [84] H.-C. Hege and D. Stalling. Fast LIC with piecewise polynomial filter kernels. In H.-C. Hege and K. Polthier, editors, *Mathematical Visualization*, pages 295–314. Springer Verlag, Heidelberg, 1998.
- [85] J. L. Helman and L. Hesselink. Representation and display of vector field topology in fluid flow data sets. *IEEE Computer*, 22(8):27–36, August 1989.
- [86] J. L. Helman and L. Hesselink. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics & Applications*, 11(3):36–46, May 1991.
- [87] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. 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.
- [88] M. W. Hirsch and S. Smale. *Differential equations, dynamical systems, and linear algebra*. Academic Press, New York, 1974.
- [89] I. Hotz, L. Feng, H. Hagen, B. Hamann, and K. I. Joy. Tensor fields visualization using a metric interpretation. *IEEE Transactions on Visualization and Computer Graphics*, pages 269–281, 2005. in Joachim Weickert and Hans Hagen, eds., *Visualization and Image Processing of Tensor Fields*. Heidelberg, Germany.

- [90] V. Interrante. Investigating the effect of texture orientation on the perception of 3D surface shape. In *Human Vision and Electronic Imaging HVEI VI, SPIE 2001*, pages 330–339.
- [91] V. Interrante. Harnessing natural textures for multivariate visualization. *IEEE Computer Graphics and Applications*, 20(6):6–11, 2000.
- [92] V. Interrante, H. Fuchs, and S. M. Pizer. Conveying the 3D shape of smoothly curving transparent surfaces via texture. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):98–117, Apr./June 1997.
- [93] V. Interrante, S. Kim, and H. Hagh-Shenas. Conveying 3d shape with texture: Recent advances and experimental findings. *Human Vision and Electronic Imaging VII, SPIE 4662*, 2002.
- [94] R. Jagnow, J. Dorsey, and H. E. Rushmeier. Stereological techniques for solid textures. *ACM Trans. Graph*, 23(3):329–335, 2004.
- [95] A. K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1989.
- [96] J. Jeong and F. Hussain. On the identification of a vortex. In *Journal of Fluid Mechanics*, pages 69–94, 1995.
- [97] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Hardware-accelerated texture advection for unsteady flow visualization. In *IEEE Visualization 2000*, pages 155–162, Oct. 2000.
- [98] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Lagrangian-eulerian advection for unsteady flow visualization. In T. Ertl, K. Joy, and A. Varshney, editors, *Proceedings of the Conference on Visualization 2001 (VIS-01)*, pages 53–60, Piscataway, NJ, Oct. 21–26 2001. IEEE Computer Society.
- [99] B. Jobard, G. Erlebacher, and M. Y. Hussaini. Lagrangian-eulerian advection of noise and dye textures for unsteady flow visualization. In *IEEE TVCG*, volume 8, pages 211–222, 2002.
- [100] B. Jobard and W. Lefer. Creating evenly-spaced streamlines of arbitrary density. *Proceedings of the 8th Eurographics Workshop on Visualization in Scientific Computing*, pages 43–55, 1997.
- [101] B. Jobard and W. Lefer. The motion map: Efficient computation of steady flow animations. In R. Yagel and H. Hagen, editors, *IEEE Visualization '97*, pages 323–328, Phoenix, Arizona, USA, Oct. 1997. IEEE Press.
- [102] B. Jobard and W. Lefer. Unsteady flow visualization by animating evenly-spaced streamlines. *Comput. Graph. Forum*, 19(3), 2000.
- [103] B. Julesz. Visual pattern discrimination. *IT-8(2)*:84–92, 1962. IRE Transaction on Information Theory.
- [104] B. Julesz. *Foundations of Cyclopean Perception*. University of Chicago Press, Chicago, Illinois, 1971.
- [105] B. Julesz. Experiments in the visual perception of texture. *Scientific American*, 232(4):34–43, Apr. 1975.
- [106] B. Julesz. Textons, the elements of texture perception, and their interactions. 290:91–97, Mar. 1981. *Nature*.
- [107] B. Julesz. A theory of preattentive texture discrimination based on first-order statistics of textons. *Biological Cybernetics*, 41:131–181, 1981.

- [108] B. Julesz and R. Bergen. Textons, the fundamental elements in preattentive vision and perception of textures. *Bell System Tech.*, 62(6):1619–1645, 1983.
- [109] B. Julesz and T. M. Caelli. On the limits of fourier decompositions in visual texture perception. *Perception*, 8:69–73, 1978.
- [110] R. L. Kashyap and A. Khotanzad. A model-based method for rotation invariant texture classification. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8(4):472–481, July 1986.
- [111] G. Kepes. *The Language of Vision*. Chicago, Theobald, 1948.
- [112] S. Kim, H. Hagh-Shenas, and V. Interrante. Showing shape with texture: Two directions seem better than one. In *Proceedings of Human Vision and Electronic Imaging VIII (HVEI'03)*, volume 5007 of *SPIE Proceedings Series*, pages 332–339, Bellingham, Washington, 2003. SPIE.
- [113] G. L. Kindlmann. Superquadric tensor glyphs. In O. Deussen, C. D. Hansen, D. A. Keim, and D. Saupe, editors, *VisSym*, pages 147–154. Eurographics Association, 2004.
- [114] R. M. Kirby, D. F. Keefe, and D. H. Laidlaw. Painting and visualization, Mar. 22 2004. In *Visualization Handbook*. Academic Press.
- [115] R. V. Klassen and S. J. Harrington. Shadowed hedgehogs: A technique for visualizing 2D slices of 3D vector fields. In *IEEE Visualization*, pages 148–162, 1991.
- [116] L. Kobbelt, M. Stamminger, and H.-P. Seidel. Using subdivision on hierarchical data to reconstruct radiosity distribution. 16(3):C347–C355, 1997. Proceedings of Eurographics'97.
- [117] R. Kosara, C. G. Healey, V. Interrante, D. H. Laidlaw, and C. Ware. User studies: Why, how, and when? *IEEE Computer Graphics and Applications*, 23(4):20–25, 2003.
- [118] R. Kosara, S. Miksch, and H. Hauser. Semantic depth of field. In *2001 IEEE Symposium on Information Visualization (INFOVIS '01)*, pages 97–104, Washington - Brussels - Tokyo, Oct. 2001. IEEE.
- [119] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. Texture optimization for example-based synthesis. *j-TOG ACM Transactions on Graphics*, 24(3):795–802, July 2005.
- [120] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. GraphCut textures: Image and video synthesis using graph cuts. pages 277–286, 2003. *ACM Transactions on Graphics, Siggraph*.
- [121] E. P. Lafortune, S.-C. Foo, K. E. Torrance, and D. P. Greenberg. Non-linear approximation of reflectance functions. In *Proc. SIGGRAPH '97*, volume 31, pages 117–126, 1997.
- [122] D. H. Laidlaw, E. T. Ahrens, D. Kremers, M. J. Avalos, R. E. Jacobs, and C. Readhead. Visualizing diffusion tensor images of the mouse spinal cord. In *IEEE Visualization*, pages 127–134, 1998.
- [123] D. H. Laidlaw, R. M. Kirby, C. D. Jackson, J. S. Davidson, T. S. Miller, M. da Silva, W. H. Warren, and M. J. Tarr. Comparing 2D vector field visualization methods: A user study. *IEEE Trans. Vis. Comput. Graph*, 11(1):59–70, 2005.
- [124] S. Lefebvre and H. Hoppe. Parallel controllable texture synthesis. *ACM Trans. Graph*, 24(3):777–786, 2005.
- [125] J.-P. Lewis. Texture synthesis for digital painting. In H. Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 245–252, July 1984.

- [126] J. P. Lewis. Generalized stochastic subdivision. *ACM Trans. Graph*, 6(3):167–190, 1987.
- [127] Y. Li, T. Wang, and H.-Y. Shum. Motion texture: A two-level statistical model for character motion synthesis. In J. Hughes, editor, *SIGGRAPH 2002 Conference Proceedings*, Annual Conference Series, pages 465–471. ACM Press/ACM SIGGRAPH, 2002.
- [128] L. Liang, C. Liu, Y.-Q. Xu, B. Gu, and H.-Y. Shum. Real-time texture synthesis by patch-based sampling. In J. Hodgins, editor, *ACM Transactions on Graphics*, volume 20, pages 127–150. ACM Press, July 2001. ISSN 0730-0301.
- [129] X. Liu, Y. Hu, J. Zhang, X. Tong, B. Guo, and H.-Y. Shum. Synthesis and rendering of bidirectional texture functions on arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(3):278–289, 2004.
- [130] Y. Liu, W.-C. Lin, and J. Hays. Near-regular texture analysis and manipulation. *ACM Trans. Graph*, 23(3):368–376, 2004.
- [131] Z. Liu, C. Liu, H.-Y. Shum, and Y. Yu. Pattern-based texture metamorphosis. In *Pacific Conference on Computer Graphics and Applications*, pages 184–193. IEEE Computer Society, 2002.
- [132] Y. L. Lous. Report on the First Eurographics Workshop on Visualization in Scientific Computing. 9(4):371–372, Dec. 1990.
- [133] M. R. Luetzgen, W. C. Karl, A. S. Willsky, and R. Tenney. Multiscale representations of markov random fields. *IEEE Trans. Signal Processing*, 41(12):3377–3396, Dec. 1993.
- [134] J. Malik, S. Belongie, J. Shi, and T. Leung. Textons, contours and regions: Cue integration in image segmentation. In *Proceedings of the 7th IEEE International Conference on Computer Vision (ICCV-99)*, volume II, pages 918–925, Los Alamitos, CA, Sept. 20–27 1999. IEEE.
- [135] J. Malik and P. Perona. A computational model of texture perception. Technical Report UCB/CSD 89/491, Computer Science Division, University of California, Berkeley, CA, Feb. 1989.
- [136] J. Mao and A. K. Jain. Texture classification and segmentation using multiresolution simultaneous autoregressive models. *Pattern Recognition*, 25(2):173–188, 1992.
- [137] W. Matusik, M. Zwicker, and F. Durand. Texture design using a simplicial complex of morphable textures. *ACM Trans. Graph*, 24(3):787–794, 2005.
- [138] N. Max and B. Becker. Flow visualization using moving textures. In *Proceedings of the ICASW/LaRC Symposium on Visualizing Time-Varying Data*, pages 77–87, Sept. 1995.
- [139] B. J. Meier. Painterly rendering for animation. *SIGGRAPH Conference Proceedings*, pages 477–484, 1996. In H. Rushmeier, Ed., New Orleans, Louisiana.
- [140] M. Morse. The foundations of a theory of the calculus of variations in the large in m-space. *Trans. Amer. Math. Soc.*, 1929.
- [141] K. Mullet and D. Sano. *Designing visual interfaces: Communication oriented techniques*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1995.
- [142] A. Nealen and M. Alexa. Hybrid texture synthesis. In P. Christensen and D. Cohen-Or, editors, *Proceedings of the 14th Eurographics Workshop on Rendering*, pages 97–105, Aire-la-Ville, Switzerland, June 25–27 2003. Eurographics Association.
- [143] A. Nealen and M. Alexa. Fast and high quality overlap repair for patch-based texture synthesis. In *Computer Graphics International*, pages 582–585. IEEE Computer Society, 2004.



- [144] F. Neyret and M.-P. Cani. Pattern-based texturing revisited. In *ACM SIGGRAPH*, pages 235–242, 1999.
- [145] H. Nothdurft. Saliency from feature contrast: additivity across dimensions. *Vision Research*, 40:1183–1201, 2000.
- [146] D. R. Peachey. Solid texturing of complex surfaces. In *Computer Graphics SIGGRAPH '85 Proceedings*, pages 279–286, July 1985. volume 19, number 3.
- [147] A. P. Pentland. Fractal-based description of natural scenes. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 6(6):661–674, Nov. 1984.
- [148] K. Perlin. An image synthesizer. *Computer Graphics SIGGRAPH '85*, 19(3):287–296, July 1985.
- [149] P. Perona. Deformable kernels for early vision. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pages 222–227, 1991.
- [150] P. Perona and J. Malik. Scale space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Machine Intell.*, 12(7):629–639, July 1990.
- [151] H. Poincaré. *Mémoire sur les courbes définies par les équations différentielles*. Gauthier-Villar, Paris, 1880-1890.
- [152] K. Popat. Conjoint probabilistic subband modeling. 1997. Massachusetts Institute of Technology, Media Lab.
- [153] K. Popat and R. W. Picard. Cluster-based probability model and its application to image and texture processing. *IEEE Transactions on Image Processing*, 6(2):268–284, 1997.
- [154] J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40(1):49–70, Dec. 2000.
- [155] F. H. Post, R. S. L. 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.
- [156] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The state of the art in flow visualisation: Feature extraction and tracking. *Computer Graphics Forum*, 22(4), 2003.
- [157] E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 465–470. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
- [158] T. Preußner and M. Rumpf. Anisotropic nonlinear diffusion in flow visualization. In *IEEE Visualization '99*, pages 325–332, Oct. 1999.
- [159] K. Pullen and C. Bregler. Motion capture assisted animation: Texturing and synthesis. In *Proceedings of SIGGRAPH 2002*. ACM Press, July, 2002.
- [160] P. T. Quinlan and G. W. Humphreys. Visual search for targets defined by combinations of color, shape, and size: An examination of task constraints on feature and conjunction searches. In *Perception & Psychophysics*, volume 41, 5, pages 455–472, 1987.
- [161] A. R. Rao and G. L. Lohse. Identifying high level features of texture perception. *Graphical Models and Image Processing, CVGIP*, 55(3):218–233, May 1993.

- [162] P. Rheingans and C. Landreth. Perceptual principles for effective visualizations, Jan. 19 1995. In *Perceptual Issues in Visualization*, pages 59-73. Springer.
- [163] T. Rieger and F. Taponecco. Interactive information visualization of entity-relationship-data. In *WSCG*, pages 99–106, 2002.
- [164] S. K. Robinson. Coherent motions in the turbulent boundary layer. In *Annual Review of Fluid Mechanics*, volume 23, pages 601–639, 1991.
- [165] M. Roth and R. Peikert. A higher-order method for finding vortex core lines. In *IEEE Visualization '98 (VIS '98)*, pages 143–150, Washington - Brussels - Tokyo, Oct. 1998. IEEE.
- [166] Y. Rubner and C. Tomasi. *Perceptual Metrics for Image Database Navigation*. Kluwer, Dec. 2000.
- [167] H. E. Rushmeier, H. H. Barrett, P. Rheingans, S. P. Uselton, and A. Watson. Perceptual measures for effective visualizations. In *IEEE Visualization*, pages 515–517, 1997.
- [168] S. Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *Symposium on 3D Data Processing, Visualization, and Transmission, 3DPVT*, pages 486–493. IEEE Computer Society, 2004.
- [169] J. Sahner, T. Weinkauff, and H.-C. Hege. Galilean invariant extraction and iconic representation of vortex core lines. In K. Brodlie, D. Duke, and K. Joy, editors, *Eurographics / IEEE VGTC Symposium on Visualization*, pages 151–160, Leeds, United Kingdom, 2005. Eurographics Association.
- [170] A. Sanna, B. Montrucchio, and P. Montuschi. A survey on visualization of vector fields by texture-based methods. 1(1):13–27, 2000. *Research Developments in Pattern Recognition*.
- [171] A. Sanna, C. Zunino, B. Montrucchio, and P. Montuschi. Adding a scalar value to texture-based vector field representations by local contrast analysis. In D. Ebert, P. Brunet, and I. Navazo, editors, *Proceedings of the symposium on Data visualisation 2002*, pages 035–041, Barcelona, Spain, 2002. Eurographics Association.
- [172] P. C. Saunders, S. C. Garrick, and V. Interrante. Visualization of nanoparticle formation in turbulent flows. In *IEEE Visualization*, page 23. IEEE Computer Society, 2004.
- [173] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. In *Proceedings of SIGGRAPH 2000*, pages 489–498. ACM Press, July, 2000.
- [174] G. Scheuermann, H. Hagen, H. Kruger, M. Menzel, and A. Rockwood. Visualization of higher order singularities in vector fields. In *IEEE Visualization '97 (VIS '97)*, pages 67–74, Washington - Brussels - Tokyo, Oct. 1997. IEEE.
- [175] H.-P. Seidel. Polar forms for geometrically continuous spline curves of arbitrary degree. *ACM Trans. Gr.*, 12(1):1–34, Jan. 1993.
- [176] H.-W. Shen, C. R. Johnson, and K.-L. Ma. Visualizing vector fields using line integral convolution and dye advection (Graphics: S. 102). In *Proceedings of the Symposium on Volume Visualization*, pages 63–70, New York, Oct. 28–29 1996. ACM Press.
- [177] H.-W. Shen and D. L. Kao. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Trans. Vis. Comput. Graph.*, 4(2):98–108, 1998.
- [178] H.-W. Shen, G. Li, and U. Bordoloi. Interactive visualization of three-dimensional vector fields with flexible appearance control. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):434–445, 2004.

- [179] H. H. Shen and V. Interrante. Compositing color with texture for multi-variate visualization. In S. N. Spencer, editor, *GRAPHITE*, pages 443–446. ACM, 2005.
- [180] E. P. Simoncelli and E. H. Adelson. Subband transforms. *Subband Image Coding*, pages 143–192, 1990.
- [181] E. P. Simoncelli, W. T. Freeman, E. H. Adelson, and D. J. Heeger. Shiftable multiscale transforms. *IEEE Transactions on Information Theory*, 38(2):587–607, 1992.
- [182] E. P. Simoncelli and J. Portilla. Texture characterization via joint statistics of wavelet coefficient magnitudes. In *Proceedings of the 5th IEEE Int'l Conf on Image Processing*, pages 62–66, Chicago, Illinois, oct 1988.
- [183] J. R. Smith and S.-F. Chang. Integrated spatial and feature image query. *Multimedia Syst*, 7(2):129–140, 1999.
- [184] S. Soatto, G. Doretto, and Y. N. Wu. Dynamic textures. In *Proceedings of ICCV '01*, pages 439–446, Vancouver, BC, Canada, july, 2001.
- [185] C. Soler, M.-P. Cani, and A. Angelidis. Hierarchical pattern mapping. *ACM Trans. Graph*, 21(3):673–680, 2002.
- [186] V. Sorensen. Art, science, 1995. School of Cinema-Television, University of Southern California.
- [187] M. Spivak. *A Comprehensive Introduction to Differential Geometry*. Publish or Perish, Inc., Boston, 1979.
- [188] D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 249–256, Aug. 1995.
- [189] J. Stam. Stable fluids. In *Proceedings of ACM SIGGRAPH 1999*, pages 121–128, 1999.
- [190] M. Szummer and R. W. Picard. Temporal texture modeling. In *Proceedings of ICIP 1996*, pages 823–826, Lausanne, Switzerland, June 01 1996.
- [191] S. Takahashi, T. Ikeda, Y. Shinagawa, T. L. Kunii, and M. Ueda. Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data. *Computer Graphics Forum*, 14(3):181–192, Aug. 1995. Proceedings of Eurographics '95. ISSN 1067-7055.
- [192] F. Taponecco. Visualizing vector fields and their flow. In *Topics, Reports on Computer Graphics*, volume 17, 3/2005.
- [193] F. Taponecco. User-defined texture synthesis. In *WSCG 2004*, pages 251–258, Plzen, Czech Republic, 2004.
- [194] F. Taponecco. Using potential theory and dense texture-based visualization for external motion applications. In *IEEE International Conference on Computational Intelligence for Modelling, Control and Automation*, pages 426–431, Vienna, Austria, 2005. IEEE Press.
- [195] F. Taponecco. Dense texture-based visualization of unsteady and multi-variate vector fields. In *Journal of COMPUTERS & GRAPHICS*, pages 353–358. Elsevier Science, 2006. Special Issue on Computer Graphics in Italy, C&G, An International Journal of Systems & Applications in Computer Graphics.
- [196] F. Taponecco. Local control for temporal evolution of textured images. In *International Conference on Computer Graphics Theory and Applications, GRAPP 2006*, pages 57–60, Setúbal, Portugal, 2006.

- [197] F. Taponocco. A study on textures and their perceptual visual dimensions as application for flexible and effective scientific visualization. In G. Gallo, S. Battiato, and F. Stanco, editors, *Eurographics Italian Chapter Conference*, pages 123–127, Catania, Italy, 2006. Eurographics.
- [198] F. Taponocco and M. Alexa. Scan converting spirals. In *WSCG International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 115–120, 2002.
- [199] F. Taponocco and M. Alexa. Piecewise circular approximation of spirals and polar polynomials. In *WSCG 2003, in co-operation with EUROGRAPHICS*, Plzen, Czech Republic, 2003.
- [200] F. Taponocco and M. Alexa. Vector field visualization using markov random field texture synthesis. pages 195–202, Grenoble, France, 2003. ACM Press, New York. Proceedings of Eurographics/IEEE TVCG Symposium on Data Visualization.
- [201] F. Taponocco and M. Alexa. Steerable texture synthesis. In *Proceedings of Eurographics*, pages 57–60, Grenoble, France, 2004. ACM Press.
- [202] F. Taponocco and W. Mueller. Visual data mining of time-dependent data. In *Topics, Reports on Computer Graphics*, volume 15, 6/2003.
- [203] F. Taponocco and T. Rieger. A flexible approach to non-homogeneous texture generation. 2005. WSCG 2005, International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision.
- [204] F. Taponocco, T. Rieger, T. Urness, and V. Interrante. Elliptical weighting for directional enhancement in controlled texture synthesis. Research Poster, ACM SIGGRAPH 2006, Boston, MA.
- [205] F. Taponocco, T. Urness, and V. Interrante. Directional enhancement in texture-based vector field visualization. accepted for publication in Proceedings of ACM Siggraph GRAPHITE 2006.
- [206] A. Telea and J. J. van Wijk. Simplified representation of vector fields. In *IEEE Visualization '99*, pages 35–42, Oct. 1999.
- [207] H. Theisel, J. Sahner, T. Weinkauff, H.-C. Hege, and H.-P. Seidel. Extraction of parallel vector surfaces in 3D time-dependent fields and application to vortex core line tracking. In *IEEE Visualization*, page 80. IEEE Computer Society, 2005.
- [208] H. Theisel, T. Weinkauff, H.-C. Hege, and H.-P. Seidel. Stream line and path line oriented topology for 2D time-dependent vector fields. In *IEEE Visualization*, pages 321–328. IEEE Computer Society, 2004.
- [209] H. Theisel, T. Weinkauff, H.-C. Hege, and H.-P. Seidel. Topological methods for 2D time-dependent vector fields based on stream lines and path lines. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):383–394, July - August 2005.
- [210] J. T. Todd and F. D. Reichel. Visual perception of smoothly curved surfaces from double-projected contour patterns. *Journal of Experimental Psychology: Human Perception and Performance*, 16(3):665–674, 1990.
- [211] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. *Proceedings of the 1998 IEEE International Conference on Computer Vision*, pages 839–846, 1998.
- [212] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H.-Y. Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Trans. Graph*, 21(3):665–672, 2002.

- [213] L. Tonietto and M. Walter. Towards local control for image-based texture synthesis. In *SIBGRAPI*, page 252. IEEE Computer Society, 2002.
- [214] L. Tonietto and M. Walter. Morphing textures with texton masks. In *SIBGRAPI*, pages 348–353. IEEE Computer Society, 2004.
- [215] A. Treisman. Preattentive processing in vision. *CVGIP: Image Understanding*, 31(2):156–177, Aug. 1985.
- [216] X. Tricoche, G. Scheuermann, and H. Hagen. Higher order singularities in piecewise linear vector fields. In R. Cipolla and R. Martin, editors, *Proceedings of the 9th IMA Conference on the Mathematics of Surfaces (IMA-00)*, volume IX of *The Mathematics of Surfaces*, pages 99–113, London, Berlin, Heidelberg, Sept. 4–7 2000. Springer.
- [217] X. Tricoche, G. Scheuermann, and H. Hagen. Continuous topology simplification of planar vector fields. In T. Ertl, K. Joy, and A. Varshney, editors, *Proceedings of the Conference on Visualization 2001 (VIS-01)*, pages 159–166, Piscataway, NJ, Oct. 21–26 2001. IEEE Computer Society.
- [218] X. Tricoche, G. Scheuermann, and H. Hagen. Topology-Based visualization of Time-Dependent 2D vector fields. In D. Ebert, J. M. Favre, and R. Peikert, editors, *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym-01)*, pages 117–126, Wien, Austria, May 28–30 2001. Springer-Verlag.
- [219] M. Tuceryan and A. K. Jain. Texture analysis. *Handbook of Pattern Recognition and Computer Vision*, pages 235–276, 1993.
- [220] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut, 1983.
- [221] E. R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, Connecticut, 1990.
- [222] J. W. Tukey. Exploratory data analysis. *Reading, MA: Addison-Wesley Publishing Company*, 1977.
- [223] J. W. Tukey. We need both exploratory and confirmatory. *The American Statistician*, 34:23–25, 1980.
- [224] G. Turk. Generating textures on arbitrary surfaces using reaction-diffusion. In *Proceedings of ACM SIGGRAPH 1991*, pages 289–298. ACM Press / ACM SIGGRAPH, 1991.
- [225] G. 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.
- [226] G. Turk and D. Banks. Image-guided streamline placement. *Proceedings of SIGGRAPH 96*, pages 453–460, 1996.
- [227] T. Urness, V. Interrante, E. Longmire, and I. Marusic. Flow visualization using natural textures, 2005. University of Minnesota, Technical Report 05-014.
- [228] T. Urness, V. Interrante, E. Longmire, I. Marusic, S. O’Neill, and T. W. Jones. Strategies for the visualization of multiple co-located vector fields, 2005. University of Minnesota, Technical Report 05-032.
- [229] T. Urness, V. Interrante, I. Marusic, E. Longmire, and B. Ganapathisubramani. Effectively visualizing multi-valued flow data using color and texture. In G. Turk, J. J. van Wijk, and R. M. II, editors, *IEEE Visualization*, pages 115–121. IEEE Computer Society, 2003.

- [230] J. J. van Wijk. Spot noise-texture synthesis for data visualization. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, volume 25, pages 309–318, July 1991.
- [231] J. J. van Wijk. Flow visualization with surface particles. *IEEE Computer Graphics & Applications*, 13(4):18–24, July 1993.
- [232] J. 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).
- [233] V. Verma, D. Kao, and A. Pang. Plic: Bridging the gap between streamlines and lic. *Proc. Symposium on Data Visualization 1999*, pages 341–348, 1999.
- [234] C. Ware. Color sequences for univariate maps: theory, experiments, and principles. *IEEE Computer Graphics and Applications*, 8(5):41–49, Sept. 1988.
- [235] C. Ware. *Information visualization: perception for design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.
- [236] C. Ware and J. C. Beatty. Using color dimensions to display data dimensions. *Human Factors*, 30(2):127–142, Apr. 1988.
- [237] C. Ware and W. Knight. Using visual texture for information display. *ACM Trans. Graph*, 14(1):3–20, 1995.
- [238] R. Wegenkittl and E. Gröller. Fast oriented line integral convolution for vector field visualization via the internet. In *IEEE Visualization '97*, pages 309–316, Nov. 1997.
- [239] L.-Y. Wei. Deterministic texture analysis and synthesis using tree structure vector quantization. In *SIBGRAPI*, pages 207–214. IEEE Computer Society, 1999.
- [240] L.-Y. Wei. Texture synthesis from multiple sources. In *SIGGRAPH 2003, Applications and Sketches*. ACM Press, 2003.
- [241] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 479–488, New York, July 23–28 2000. ACM Press / ACM SIGGRAPH / Addison Wesley Longman.
- [242] L.-Y. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH*, pages 355–360, New York, NY 10036, USA, 2001. ACM Press.
- [243] T. Weinkauff, H. Theisel, H.-C. Hege, and H.-P. Seidel. Topological construction and visualization of higher order 3D vector fields. *Comput. Graph. Forum*, 23(3):469–478, 2004.
- [244] T. Weinkauff, H. Theisel, K. Shi, H.-C. Hege, and H.-P. Seidel. Extracting higher order critical points and topological simplification of 3D vector fields. In *IEEE Visualization*, page 71. IEEE Computer Society, 2005.
- [245] D. Weiskopf and G. Erlebacher. Flow visualization overview, 2004. in *Handbook of Visualization*.
- [246] D. Weiskopf, G. Erlebacher, M. Hopf, and T. Ertl. Hardware accelerated lagrangian-eulerian texture advection for 2dflow visualizations. In *Proceedings of the Vision Modeling and Visualization Conference*, pages 439–446, Nov. 2002.
- [247] R. Wilson and M. Spann. *Image Segmentation and Uncertainty*. Wiley, New York, 1988.
- [248] A. Witkin and M. Kass. Reaction-diffusion textures. In T. W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 299–308. ACM Press / ACM SIGGRAPH, July 1991.

- [249] J. M. Wolfe. Guided search 2.0: a revised model for visual search. In *Psychonomic Bulletin & Review*, volume 30,2, pages 201–238, 1994.
- [250] J. M. Wolfe, N. Klempen, and K. Dahlen. Postattentive vision. In *Journal of Experimental Psychology: Human Perception & Performance*, 2000. volume 26,2, pages 693–716.
- [251] S. Worley. A cellular texture basis function. In *Proceedings of ACM SIGGRAPH 1996*, pages 291–294, 1996.
- [252] L. Ying, A. Hertzmann, H. Biermann, and D. Zorin. Texture and shape synthesis on surfaces. In *EG Workshop on Rendering*, pages 301–312, 2001.
- [253] S. Zelinka and M. Garland. Towards real-time texture synthesis with the jump map. In S. Gibson and P. Debevec, editors, *Proceedings of the 13th Eurographics Workshop on Rendering (RENDERING TECHNIQUES-02)*, pages 99–104, Aire-la-Ville, Switzerland, June 26–28 2002. Eurographics Association.
- [254] S. Zelinka and M. Garland. Jump map-based interactive texture synthesis. *ACM Transactions on Graphics*, 23(4):930–962, Oct. 2004.
- [255] J. Zhang, K. Zhou, L. Velho, B. Guo, and H.-Y. Shum. Synthesis of progressively variant textures on arbitrary surfaces. *ACM Transactions on Graphics, TOG (Siggraph '03 Proceedings)*, 22(3):295–302, 2003.
- [256] S. C. Zhu, Y. N. Wu, and D. Mumford. Filters, random-fields and maximum-entropy (frame): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, Mar. 1998.