Visualizing Multiple Two-Dimensional Fields of Data in a Single Image

Version of May 28, 2013

C.C. Van Egmond
Visualizing Multiple Two-Dimensional Fields of Data in a Single Image

THESIS

submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

C.C. Van Egmond
born in Heemskerk, the Netherlands

TU Delft

Computer Graphics and Visualization Group
Department of Intelligent Systems
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Graphics Group
Department of Computer Science
The University of Auckland
Auckland, New Zealand
http://www.cs.auckland.ac.nz/
Cover picture: An example of a visualization of four synthetic spatial two-dimensional data fields (three vector fields and a scalar field) generated by a texture grammar.
Visualizing Multiple Two-Dimensional Fields of Data in a Single Image

Author: C.C. Van Egmond
Student id: 1232444
Email: ccvanegmond@gmail.com

Abstract

With advancements in scientific measurement instruments and simulation techniques come increasingly complex and in many cases multivariate data. The development and improvement of techniques that are able to convey as much information as possible from this data to scientists, doctors, and other potential users is an important research topic in the field of scientific visualization.

This thesis contributes a new automated approach to the visualization of multiple two-dimensional, spatial data fields in a single image. We introduce a new generative formal grammar called texture grammar that combines implementations of existing visualization techniques and image compositing techniques to automatically produce visualizations, depending on the input fields. The image compositing techniques are perceptually motivated, and we evaluated their effectiveness with a user study. The resulting visualizations capture important characteristics of all input fields, as well as relations between the fields. An added benefit of this method is the reduction of user interaction, since a larger amount of data can be perceived at once.

Thesis Committee:

Chair: Prof. Dr. E. Eisemann, Faculty EEMCS, TU Delft
University supervisor: Dr. C.P. Botha, Faculty EEMCS, TU Delft
Company supervisor: Dr. B.C. Wünsche, Graphics Group, University of Auckland
Committee Member: Dr. Alessandro Bozzon, Faculty EEMCS, TU Delft
Preface

Over the past 15 months, I have been lucky enough to be part of two great research groups: the Computer Graphics and Visualization group (Department of Intelligent Systems) at Delft University of Technology, the Netherlands, and the Graphics Group (Computer Science Department) at the University of Auckland, New Zealand. Both groups are filled with talented and pleasant people, it has been inspiring and an absolute privilege to have had the opportunity to work in both of these groups. I would like to thank the following people in particular:

First of all my thesis committee, consisting of Prof. Dr. Elmar Eisemann, Dr. Charl Botha and Dr. Burkhard Wünsche, for their combined dedication and commitment during the project, for their encouragement and help to bring this work to a close. Their optimistic attitudes have continuously helped me to put my own skeptical views in perspective.

My supervisor in Delft, Dr. Charl Botha, for his classes in Data Visualization and Medical Visualization inspiring me to find a final project in these fields. Through his extensive network he found me an opportunity to do my thesis work in New Zealand. My daily supervisor in Auckland, Dr. Burkhard Wünsche for providing me with this very opportunity, for his continued support and advice, his enthusiasm, and for providing me with the necessary background knowledge and materials to be able to do my research effectively.

I would also like to thank my fellow students from the graphics labs in Auckland and in Delft for the great company and many interesting and often fruitful conversations during coffee breaks about all kinds of relevant and less relevant topics.

Last but certainly not least, I would like to thank my friends and family for their support (and their not expecting me to Skype them every month while I was abroad). Tony and Zureen for having me stay at their place during my time in Auckland, and most of all I am grateful to my fiancee Chrissy, who unconditionally kept me sane and motivated and always believed in my work.

C.C. Van Egmond
Delft, the Netherlands
May 28, 2013
Contents

Preface iii

Contents v

List of figures vii

1 Introduction 1
   1.1 Context ......................................................... 1
   1.2 Motivation ..................................................... 2
   1.3 Research Question ........................................... 3
   1.4 Contributions ............................................... 4
   1.5 Thesis Structure ............................................. 4

2 Background and Related Work 5
   2.1 Advancements in Scientific Visualization ....................... 5
   2.2 Visual Perception ........................................... 17

3 Design and evaluation of multi-field visualization techniques 21
   3.1 Abstract ....................................................... 21
   3.2 Introduction .................................................. 21
   3.3 Literature Review ............................................ 22
   3.4 Design ......................................................... 23
   3.5 Evaluation .................................................... 25
   3.6 Results and Discussion ...................................... 28
   3.7 Conclusions and Future Work ................................ 31

4 Visualizing multi-field data using texture grammars 33
   4.1 Concept ....................................................... 33
   4.2 Formal Grammars ............................................. 34
   4.3 Texture Grammar .............................................. 35
   4.4 User Interaction ............................................. 38
List of figures

2.1 Various stages of a typical data visualization pipeline. *Data acquisition:* collect data from resources such as temperature sensors, MRI or CT scanners, or generate data synthetically. In the *pre-processing* stage, the data is prepared for visualization, for example by filtering, or extracting interesting features from the data. In the *mapping* stage, visualization techniques are employed to map the data to a displayable form such as images or geometry. The *rendering* stage finally renders these to a display device. It should be noted that user interaction is not mentioned in this pipeline, but depending on the application can be present at all of the stages in the pipeline (e.g. changing visualization parameters in the mapping stage, or dragging and re-orienting the camera in the rendering stage). .......................................................... 6

2.2 Three examples of popular colour maps applied to an MRI scan of a human head (data set courtesy of Siemens Medical Systems, Inc., Iselin, NJ.), from left to right: *grayscale map, rainbow map* and *heat map*. ................................. 7

2.3 Isolines example image. ....................................................... 8

2.4 Two direct volume renders of an MRI scan of a human head (data set courtesy of Siemens Medical Systems, Inc., Iselin, NJ.) with clipped geometries visualized together in one image. One of the images shows skin, while the other shows brain tissue. .................................................. 9

2.5 An example 2D vector field visualization using an arrow plot. Vector magnitude is represented by colour. ...................................................... 10

2.6 An example vector field visualization using streamlines. .......................... 11

2.7 An example spot noise visualization. 10000 spots were distributed to visualize an underlying synthetic vector field. .............................................. 12
2.8 Top left: vector field, top right: input texture (white noise), bottom left: output pixel, bottom right: example of a LIC visualization. To compute the output pixel value of the pixel indicated by a blue outline, we use the vector field to determine a streamline with a kernel size of, in this case, 8 pixels. All pixels of the input texture that lie under this kernel are convoluted - in the simplest case by a weighted addition of the pixels under the streamline - resulting in the colour as shown in the output pixel. Vector magnitude can be represented as well, either by colour mapping or by varying the kernel size.

2.9 Oriented sliver textures visualization: eight sliver layers oriented at intervals of 15° each. Image courtesy of Weigle et al. [2000].

2.10 Schema of the oriented sliver textures concept. (a,b) two scalar fields represented with 0° and 90°, respectively; (c) both fields displayed in a single image, overlapping values show as elements that look like plus signs. Image courtesy of Weigle et al. [2000].

2.11 Perceptual brush strokes visualization representing four scalar fields in a single image: wind speed by spatial density, temperature by colour, precipitation by orientation (tilted strokes mean heavy rain) and pressure by stroke size.

2.12 A representation of two vector fields in which streamlines are placed using the improved quality metric [Urness and Interrante, 2008].

2.13 The perceptual process. Image courtesy of Goldstein [2010].

2.14 The law of pragnanz: in this image we see a series of three shapes (rectangle, circle and triangle), rather than many more complicated shapes.

3.1 Different pairs of 2D vector fields visualized simultaneously using the blend method (left), the overlay method (middle), and the smooth bump map method (right).

3.2 Four visualizations showing two vector fields simultaneously using different pattern masks. From the top left image, in clockwise order: a fine checkerboard pattern, a gradual shadow weaving pattern, a solid shadow weaving pattern, and a fine diagonal pattern.

3.3 Web interface used in the user study. The screen shot shows a critical point in the yellow vector field (indicated as transparent red circle) located by clicking on the canvas. The click has triggered a context menu enabling the user to select the type of critical point. After locating and classifying all identified critical points clicking the button at the bottom displays the next experiment.

3.4 Mean and 95% confidence interval for the F-score. Both bump mapping techniques are shown as their performance differs significantly. For the masking technique, only the result for the best performing masking pattern (coarse diagonal) is shown.

3.5 Mean and 95% confidence interval of the time taken per experiment in seconds. For the masking technique, only the results for the best performing pattern (coarse diagonal) are shown.
4.1 Production rules for a context-sensitive grammar that produces the language $L$ as shown in equation 4.1. .......................... 34

4.2 Three examples of the application of the production rules of grammar $G$ as shown in equation 4.1, resulting in three words that are part of the language $L$ as defined by $G$. The notation $X \rightarrow Y$ reads “String $X$ produces string $Y$ by means of production rule $i$”. .......................... 35

4.3 An example of a set of production rules for a texture grammar. The first (starting) rule is adjusted based on the data fields provided to the module - in this case, one scalar field and three vector fields. Rule number 2 and 3 make sure that if we have several vector fields in a row, we don’t use the same technique for each. Rule 7 does the same for scalar fields. Rules 13 to 18 trigger for combinations of isolines, arrowplots and streamlines with any other terminal. Rule 19 is in place to make sure we don’t end up with more than one terminal. .......................... 39

4.4 Result of applying a texture grammar to three vector fields and one scalar field; one vector field is visualized using LIC and consequently alpha blended with a colour map visualization of the scalar field. A second vector field is visualized using streamlines, which are applied as a bumpmap over the image, and the third vector field is rendered as an arrow plot which is overlaid on top of the image. .......................... 40

4.5 Result of applying a texture grammar to three vector fields and one scalar field, resulting in a composition of four visualizations in a single image, very similar to the configuration after step 7 in table 4.3. a) a blue-to-red colourmap is applied to the scalar field. b) LIC is applied to vector field 1, and the result is blended with the result of step a. c) a streamline visualization is generated for vector field 2, and is subsequently bump-mapped over the result of step b. d) finally, an arrow plot is generated for vector field 3, and is overlaid on top of the result so far, to get our final composition. .......................... 41

4.6 The top two rows demonstrate user interaction by means of a magic lens after visualizing four fields of data using the texture grammar. In the top row we can see the arrowplot of a vector field (left) and the colour map of a scalar field (right) in focus. In the middle row, the focus is on the LIC render of a vector field (left) and the streamlines render of another vector field (right). The bottom image, finally, shows the resulting visualization without any user interaction. .......................... 43

4.7 An example of a production rule using a conditional and a variable replacement. The rule only triggers if the given condition $V_x.length \geq 0.1$ evaluates true ($V_x.length$ refers to the vector length which is normalized in the range $[0, 1]$). In this example, LIC($V_x, k$) stands for a LIC operation on vector field $V_x$ with kernel size $k$; this example rule illustrates how we could use field properties to manipulate parameters locally, using kernel sizes in the range $[5, 50]$ depending on the vector length. .......................... 44

B.1 An example of a Voreen data flow network for the visualization of anatomical structures. Image courtesy of The Voreen Team. .......................... 57
B.2 The Voreen module \textit{MFTGSliceRenderer} takes two inputs: first, a collection of co-located scalar or vector data fields (these can be volumes, in which case a slice needs to be selected). Second, a texture grammar specified in a text file. This can be a default texture grammar or a customized, user-defined version. The parser/interpreter takes this texture grammar and takes note of the input data fields and their types - this information will be added in the starting production rule. The texture grammar runs, and the resulting combination of visualizations and compositing methods are sent to the data preparation part of the module. The data is uploaded to the fragment shader which also receives information from the texture grammar, and the rendered image is finally rendered to a display window.

B.3 Heatmap colour scale, to map colours to normalized values in the range $[0, 1]$. 

Chapter 1

Introduction

In this chapter, we introduce the field of scientific visualization in section 1.1 and motivate the need for techniques to visualize multiple fields of data in section 1.2. We then present our research questions in section 1.3 and finally list the main contributions presented in this work in section 1.4.

1.1 Context

Technological advances over the past decade have enabled scientists to create ever larger and more complex scientific data sets. Examples are advanced numerical simulations, remote sensing data, medical imaging modalities, and applications utilizing multivariate data, e.g. in population health and homeland security. The increased complexity of data has made it more difficult to understand, analyze and communicate it. Scientific Visualization is an attempt to simplify these tasks according to the motto “A picture says more than a thousand words”. Representing scientific data as images and interactive 3D models improves the perception of features and patterns in the data, facilitates the navigation through and interaction with complex and disparate sets of data, and improves the communication of scientific results with peers and the wider community.

The main challenge in Scientific Visualization is mapping high-dimensional data onto a 2D display with only 3 perceptual dimensions (RGB colors). In order to improve perception many visualization techniques utilize glyphs which encode data parameters using geometric properties such as glyph shape, size, and orientation. While glyphs have been shown to be effective in a wide variety of applications, they usually only show information at a single point and it is difficult to visualize multiple data fields at once in order to discover relationships between them. Multi-field visualization is a central issue of current research [Moorhead et al., 2006]. Novel image acquisition and simulation techniques have made it possible to record a large number of co-located data fields. Visualizing them simultaneously facilitates the investigation of correlations and relationships between the fields [Urness et al., 2006]. An example application is the visualization of multi-field cardiac imaging data (MRI, PET, CT, DTI, tagged MRI, molecular imaging) in order to analyse
1.2 Motivation

The term multi-field visualization deserves some clarification. When discussing complex scientific datasets, we can discern several ways for data to have multiple attributes. Fuchs and Hauser [2009] describe the terms multi-variate and multi-dimensional as related to the structure of the data and the relation of the data to the physical world, and summarize the terms multi-channel, multi-modal, multi-field and multi-valued as related to the method of data acquisition and data representation. They state multi-field data contains multiple attributes grouped into physical fields such that within a group, attributes need the others to be interpretable. For the sake of clarity, we will use the term multi-field throughout this thesis to describe data that contains multiple fields sampled over the same physical space. The values stored in these fields can either be recorded using different input modalities or by having derived fields.

1.2 Motivation

The goal of visualizing data is to understand its structure, or identify characteristics such as patterns, anomalies or extreme values. As we mentioned in the previous section, multi-field visualization is currently an important issue in many application areas - our original motivation to develop a novel texture-based visualization method for multivariate, multi-field data sets comes from cardiac imaging where the improved diagnosis and understanding of diseases requires the simultaneous visualization of different imaging data sets. However, we are confident that the concept presented in this thesis will prove useful in many other application fields. An example is computational fluid dynamics, which naturally generates a set of related measurements (e.g., pressure, temperature, velocity) [Batchelor, 2000]. In applications where only a single field of data is generated, it can often be beneficial to investigate derived fields in conjunction with the original data, to identify important relationships.

A popular class of methods to visualize multi-field data is multiple views - information on a single conceptual entity is presented using various visualizations in several different windows (views), to support the comparison of data [Wang Baldonado et al., 2000; Roberts, 2007]. To enhance consistency in the data, views can be linked. For example, when we display multiple views in which direct volume rendering is applied using different transfer functions. Linking these views means that when the camera angle on one view is changed, it is changed accordingly on the other (linked) views.

Some research has been done in visualizing multi-field data in a single image, but this research area is still in its infancy. An example to visualize multiple data attributes is by using glyphs, where different attributes of a glyph (e.g. the orientation, size and colour of shapes) represent different data attributes. A shortcoming of this method is that it is difficult to see relations, as glyphs represent individual points of data. Geometric glyphs can carry more information than a single pixel; however, they take up much more screen space.
We wish to investigate a more spatially dense way to visualize multi-field data in a single image. The human brain has evolved to efficiently process complex textures. We want to explore perceptually motivated textures for improving perception and understanding of complex data sets. Relevant research includes the work of Shenas and Interrante [2005] who combine colour and texture to represent multiple values at a single location; Urness et al. [2006] who explore strategies for the visualization of multiple coincident two dimensional vector fields; and Urness and Interrante [2008], who investigate the visualization of multiple two dimensional vector fields in a single image using streamlines.

User studies are not utilized often enough to evaluate visualization methods. It is difficult to design a good experiment, but a well conducted study is usually worth the effort. The results can have a considerable impact and can potentially contribute to the scientific foundations of the visualization discipline [Christopher et al., 2003]. We have conducted a user study on the effectiveness of several methods that visualize two vector fields in a single image, and presented the results at the 27th Image and Vision Computing New Zealand conference (IVCNZ'12). An extended version of the publication of these results can be found in chapter 3.

In this thesis we present a novel method for the visualization of multi-field data. We introduce a new generative formal grammar, which takes a list of input field types, and outputs a combination of visualization techniques that are to be applied to these fields. The grammar also defines which image processing techniques will be used to combine the different visualizations in one image.

1.3 Research Question

The main research question that is examined in this thesis is as follows:

*How can we make use of generative formal grammars to combine data visualization and image processing methods to synthesize textures that facilitate the exploration of multiple two-dimensional fields of data in a single image?*

In order to answer this question, the following subquestions will be studied:

1. Which data visualization techniques can be effectively combined without cluttering or obscuring information?

2. Which methods of image composition will be most effective to combine several visualizations in a single image?

3. How can we map visualization techniques and image composition methods to symbols in a formal grammar, and how can automatically manipulate visualization parameters?
4. Can we integrate any methods of user interaction to increase the effectiveness of data exploration?

To implement a prototype module that can take a set of data fields, parse and run texture grammars and render an output texture, we will either need to use a framework with existing implementations of the techniques as determined by subquestions 1 and 2, or implement the techniques ourself. Another challenge is, if we do find it desirable to provide means of user interaction as per subquestion 4, can we ensure this interaction happens in real-time by maintaining interactive frame-rates?

1.4 Contributions

The main contribution presented in this thesis is a new automated approach to the visualization of multiple two-dimensional, spatial data fields in a single image. We introduce the texture grammar, a new formal grammar that combines implementations of existing dense visualization techniques (such as Line Integral Convolution and colourmapping) to automatically produce visualizations, depending on the input fields. The grammar attempts to find the most effective visualization technique for each field, as well as selecting image compositing techniques that ensure minimal information loss when combining visualizations of multiple fields.

- A novel solution for the visualization of multi-field data, where visualization operators are selected by running a generative formal grammar.
- We present the formal notion of the texture grammar.
- We present a user study that investigates the effectiveness of four image composition techniques in the light of multi-field data visualization.

1.5 Thesis Structure

In chapter 2 we introduce concepts from the literature that are most important to this thesis. Three main topics are discerned: first we discuss advancements in scientific visualization and relevant visualization methods, followed by an explanation of formal grammars and finally we consider several important concepts from perception and the human visual system.

Then, in chapter 3 we provide a copy of a published user study that evaluated several multi-field visualization techniques, which we performed as a part of the research that was done for this thesis. In chapter 4 we present the concept of our main contribution: the texture grammar, a novel method to visualize multi-field data, as well as details on the implementation.

Finally, in chapter 5 we discuss the texture grammar approach to visualizing multiple datafields, draw conclusions and present recommendations for future work.
Chapter 2

Background and Related Work

In this chapter we introduce the reader to various concepts that are important in the light of this thesis. In section 2.1, several concepts from the field of scientific visualization\(^1\) are discussed, and section 2.2 is devoted to important concepts from cognitive science that apply to computer graphics and visualization.

2.1 Advancements in Scientific Visualization

In various fields of science, scientists have to deal with and interpret datasets of ever increasing complexity, mainly due to technological advances in instrumentation, simulation and data storage facilities. This gives rise to an important challenge: how can we successfully represent the relevant pieces of information from such datasets, to enable humans to perceive them efficiently and accurately? The field of scientific visualization is dedicated to finding solutions to this problem. A visualization is an attempt to convey insight and understanding about otherwise complex numerical data, by having the underlying dataset represented by one or more images. Successful visualization techniques make effective use of visual attributes to exploit the unique capabilities of the human visual system to detect trends and patterns in images. Figure 2.1 shows a typical data visualization pipeline. In the history of scientific visualization, myriads of techniques have been developed to this end, dealing with different types of data, exploiting various properties of human perception.

There are two ways to characterise the dimensionality of a scientific dataset. First, by its spatial (and temporal) dimensions, for example two dimensional (data on a Euclidian surface) or three dimensional (volumetric data), where time serves as an extra dimension in the case of dynamic data. Second, there is the dimensionality of the individual recorded data values, which can be for example a scalar, vector or tensor value, or multiple values at each point. In the following sections several major visualization techniques are discussed

---

\(^1\) Scientific visualization aims at creating insightful images or animations of numerical datasets obtained from scientific measurements or simulations. It is a specialization within the broader field of data visualization, just as information visualization, which contrastingly deals with abstract and non-spatial data. In this thesis we will be focusing exclusively on scientific visualization of spatial data.
2.1 Advancements in Scientific Visualization

Background and Related Work

Figure 2.1: Various stages of a typical data visualization pipeline. Data acquisition: collect data from resources such as temperature sensors, MRI or CT scanners, or generate data synthetically. In the pre-processing stage, the data is prepared for visualization, for example by filtering, or extracting interesting features from the data. In the mapping stage, visualization techniques are employed to map the data to a displayable form such as images or geometry. The rendering stage finally renders these to a display device. It should be noted that user interaction is not mentioned in this pipeline, but depending on the application can be present at all of the stages in the pipeline (e.g. changing visualization parameters in the mapping stage, or dragging and re-orienting the camera in the rendering stage).

and compared, and they are categorised by their individual data type (scalar, vector, or multiple values). We discuss techniques that deal with data in the form of scalar fields (section 2.1.1), vector fields (section 2.1.2) and multi-field datasets (section 2.1.3). In each section an overview of all discussed techniques and some of their properties is listed in a table.

The properties used to classify the techniques are the following: the visual attributes used by a method to encode data are represented by icons, as shown in table 2.1. These are based on a framework as suggested by Wünsche [2004]. Visual attributes are attributes that can be perceived by the human visual system (HVS). These attributes can be used in different ways by the HVS to discriminate between values, and to perceive patterns and relations in the data. The spatial density refers to whether a visualization is discrete (data is represented by icons that are sparsely distributed over the output image, where ’empty’ room can exist in between icons which does not provide information about the underlying field) or continuous (the visualization covers the entire output image and is spatially dense). The information content describes the dimensionality of the data that is represented by a single pixel in the visualization image: how much information is used to calculate the value at one pixel? This value ranges from point (for a single data point), line and surface to volume (respectively data points are used in one, two or three dimensions). The generating function is a mathematical function to provide some insight in how the values are manipulated to obtain the needed visual attributes.

2.1.1 Visualizing scalar fields

Scalar fields are the most elementary type of dataset, associating a scalar value to every point in space (which is typically two or three dimensional Euclidian space, sometimes having time as another dimension). Typical application examples include geospatial data such as temperature or pressure distribution, and imaging in medicine where magnetic resonance imaging (MRI) and computed tomography (CT) scanners generate spatial volumes of scalar data [Udupa and Herman, 1991]. Table 2.2 shows an overview of some of the most popular
Background and Related Work

2.1 Advancements in Scientific Visualization

Table 2.1: Visual attributes.

- Position
- Size
- Orientation
- Volume
- Colour
- Shape
- Texture

Techniques used to visualize spatial scalar data sets. The rest of this section briefly discusses these techniques.

**Colourmap**  Colourmapping is a technique that assigns a colour to every possible field value. It is generally used to visualize a two-dimensional field of data (or slices of volume data), where the colour at each point is meant to clearly hint at the value (intensity) recorded at that point. In this light, it is important that the colourmap clearly conveys the order of the scalar values, and that the perceived difference in colours agree as closely as possible to the difference in scalar value. For example, a rainbow colour map uses hue to indicate order and is much less useful than a grayscale map, since even though the colours are ordered by wavelength, the map is not actually perceptually isolinear. Still, according to Borland and Taylor II [2007], the rainbow colour map is one of the most popular colour maps in the visualization community, and in their paper they propose to stop using this approach in visualization. Figure 2.2 provides examples of three different colour maps.

Figure 2.2: Three examples of popular colour maps applied to an MRI scan of a human head (data set courtesy of Siemens Medical Systems, Inc., Iselin, NJ.), from left to right: grayscale map, rainbow map and heat map.

**Isoline and isosurface**  An isoline (also referred to as contour) is the line containing the complete set of points in a two-dimensional field for which the scalar value is equal to some value \( c \) (the isovalue). This line divides the field into regions where the scalar values are
either all higher or all lower than the iso-value. Rendering an image with multiple isolines at different values results in a topographical image (see figure 2.3). The isosurface is an extension of this concept to a third dimension, where the division is a surface containing all the data points equal to the iso-value. The most popular isosurface extraction algorithm is Marching Cubes [Lorensen and Cline, 1987], which extracts a polygonal mesh from a scalar volume. The input volume is represented as a regular grid of sampled field values, and each grid cell has eight sampled values at its corners. The algorithm ”marches” through the volume, determining a polygon configuration for each cell. This configuration is stored in eight bits, using the field values stored at the eight cell corners. If a corner value is higher than the iso-value, the corresponding bit is set to 1, otherwise it is set to 0. The resulting eight-bit configuration is mapped to a polygon. Each vertex of this polygon is then placed on the cells edge at the intersection point calculated by linearly interpolating the values at both ends of the cell edge.

Heightmap

The heightmap is a technique in which a two dimensional field of scalars is visualized by displacing a flat surface in one direction (usually vertical). The surface will be offset at each point with an amount relative to the scalar value at that point. A flat base surface at ”0 height” can be used as a reference. The main advantage of the heightmap is its accuracy in representing quantitative information. A drawback is that a two dimensional scalar field has to be represented in a three dimensional image to perceive the different height values.

Direct Volume Rendering

Direct volume rendering (DVR) can transform scalar volume data into an image by mapping scalar values to colour and transparency values using transfer functions. The image is built pixel by pixel by shooting a ray from each pixel location through the volume, accumulating the colour and transparency information encountered along this ray [Sobierajski and Kaufman, 1994; Purcell et al., 2002; Levoy, 1990]. The simplest form of this algorithm is ray casting, which is non-recursive and casts only primary rays. Methods employing ray tracing, in contrast, recursively cast additional rays, to simu-
late caustics [Wald et al., 2002; Levoy, 1990]. An example of DVR can be found in figure 2.4.

In recent years, using realism to enhance DVR visualizations has been explored. Ropinski et al. developed a realistic lighting model for volume rendering and demonstrate with a user study that realism helps observers to assess depth more accurately and efficiently [Ropinski et al., 2010]. Kroes et al. apply Monte Carlo ray tracing to direct volume rendering, in an interactive framework that demonstrates several realistic effects [Kroes et al., 2012].

Figure 2.4: Two direct volume renders of an MRI scan of a human head (data set courtesy of Siemens Medical Systems, Inc., Iselin, NJ.) with clipped geometries visualized together in one image. One of the images shows skin, while the other shows brain tissue.

2.1.2 Visualizing vector fields

Vector fields associate a vector (usually containing two or three scalar values) to every point in space. The interesting properties of vectors are its orientation and magnitude. Examples of vector fields are the velocity and direction of flowing wind from a wind tunnel, or fluid flows from computational fluid dynamics (CFD). Vector fields are often referred to by the term flow fields as well. Table 2.3 shows an overview of the techniques presented in this section.

Arrow plot An arrow is perhaps the most obvious icon to represent a vector - it can easily be used to show direction in two or three dimensions. An arrow plot (sometimes referred to as 'hedgehogs') places arrows as glyphs at each data point (or at a selection of points) to represent individual vectors. The orientation of the icon corresponds to the two or three dimensions present in the vector. By varying the colour, size or length of the arrow glyph, we can represent the vector magnitude. In two dimensions, an effective and complete overview of a vector field can be conveyed; in three dimensions, however, there are several drawbacks, such as directional ambiguity, cluttering and a poor spatial effect [Post and van Wijk, 1994]. An example of an arrow plot visualization is shown in figure 2.5.
2.1 Advancements in Scientific Visualization

Table 2.2: Techniques to visualize scalar fields. For all generating functions, \( \rho(u,v) \in \mathbb{R} \) represents the field value at location \((u,v)\). \( f(x,y) \) denotes the output pixel value at location \((x,y)\), \( c \) denotes the iso-value for the isoline and isosurface techniques, and \( z \) is the offset in one dimension for the heightmap, where the other two dimensions remain the same. For DVR, \( r \) denotes the collection of points along the cast ray.

<table>
<thead>
<tr>
<th>Visual Attributes</th>
<th>Spatial Density</th>
<th>Information Content</th>
<th>Generating Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Colourmap</strong></td>
<td>Continuous</td>
<td>Point</td>
<td>( f : \rho(u,v) \rightarrow (r,g,b) )</td>
</tr>
<tr>
<td><strong>2D Contour: Isoline</strong></td>
<td>Discrete</td>
<td>Point</td>
<td>( f(x,y) = \begin{cases} 1 &amp; \text{if } \rho(x,y) = c \ 0 &amp; \text{otherwise} \end{cases} )</td>
</tr>
<tr>
<td><strong>3D Contour: Isosurface</strong></td>
<td>Discrete</td>
<td>Point</td>
<td>( f(x,y,z) = \begin{cases} 1 &amp; \text{if } \rho(x,y,z) = c \ 0 &amp; \text{otherwise} \end{cases} )</td>
</tr>
<tr>
<td><strong>Heightmap</strong></td>
<td>Continuous</td>
<td>Point</td>
<td>( f : \rho(u,v) \rightarrow (x,y,z) )</td>
</tr>
<tr>
<td><strong>DVR (Ray Casting)</strong></td>
<td>Continuous</td>
<td>Line</td>
<td>( f(x,y) = \sum r \rho(r) )</td>
</tr>
</tbody>
</table>

Figure 2.5: An example 2D vector field visualization using an arrow plot. Vector magnitude is represented by colour.

**Particle Advection** Particle advection is a technique that visualizes vector fields by means of a dynamic image; it animates a set of particles which are released at various points in the vector field, and subsequently advected by the vector field. This technique can be used for a realistic visualization of fluid flow; velocity can be directly modelled by the speed with which particles move from frame to frame [Post and van Wijk, 1994]. Therefore, this method is intuitive and easily understood in two dimensional visualizations. In three dimensional visualizations it suffers from similar drawbacks as the arrow plot; directional ambiguity and cluttering. The interactivity of the method is also dependent on the amount of particles that are visualized.
Streamlines A streamline is a curve that is tangent to the given vector field at every location. The technique can be applied to two dimensional as well as three dimensional vector fields. Usually, the streamlines are "grown" from selected seed points, and there are several strategies available to select these seed points (e.g. using regular grids, random sampling, or letting the user select the seed points). Turk and Banks [1996] introduce a seeding technique that uses an energy function to guide the placement of streamlines at a specified density. Mathematically we can describe a streamline as an integral curve \( x(s) \) which satisfies equation 2.1,

\[
\frac{dx}{ds} = v(x(s)), x(0) = x_0
\]  

(2.1)

where \( v(x) \) is a vector field and the starting point of the streamline \( x_0 \) is defined by the initial condition \( x(0) \). The main advantage of streamlines is their intuitive representation. By using a single line to represent multiple values it also reduces the visual cluttering that can be caused by multiple glyphs such as vector arrows [Wünsche, 2004]. Vector magnitude can be encoded in the colour of the streamline. One problem with streamlines is the difficulty in finding good seed points. To produce effective visualizations, it is important to choose a proper streamline density. When they are placed too close together, streamlines merge and directional information is obscured whereas if they are placed too far apart, information is missing for large areas in between. When streamlines are used in multi-field visualization this problem becomes even more important, as now other fields of data are dependent on the density of streamlines, and dense placement means obscuring more information in other fields. See figure 2.6 for a streamlines visualization of a vector field.

Figure 2.6: An example vector field visualization using streamlines.
2.1 Advancements in Scientific Visualization

Background and Related Work

Spot Noise  This was one of the first dense, texture-based visualization techniques. A large number of spots (intensity functions) are distributed over a vector field, and then transformed based on the underlying vector values, by being stretched out elliptically [van Wijk, 1991]. The author regards spot noise as the spatial analogue of shot noise [Birch, 1973]; it is produced by the successive repetition at random intervals of independent pulses. For spot noise, the pulse \( h(x) \) is considered a spot that is dropped on a plane. Spot noise is defined as in equation 2.2, where \( x_i \) are random positions on the plane and \( a_i \) are scaling factors [van Wijk, 1991]. A spot is a function with unity intensity inside the elliptical shape, and zero everywhere else. The summation shown in eq. 2.2 represents the blending of each spot at random positions. Figure 2.7 shows an example of a spot noise visualization of a synthetic vector field with a singularity in the middle.

\[
f(x) = \sum_i a_i h(x - x_i)
\]  

(2.2)

Figure 2.7: An example spot noise visualization. 10000 spots were distributed to visualize an underlying synthetic vector field.

Line Integral Convolution (LIC)  Arguably the most popular texture-based vector field visualization technique is Line Integral Convolution (LIC), first proposed by Cabral and Leedom [1993]. For each output pixel, the colour value is computed by convolving an input texture (usually containing white noise) along a bi-directional streamline calculated over the underlying vector field, centered around the location of the output pixel. This process is shown in figure 2.8, as well as an example of what an LIC visualization might look like. The calculation of a pixel in the output image at location \((u, v)\) can be modelled with the
following equation:

\[ T_{\text{out}}(u, v) = \sum_{p \in S} T_{\text{in}}(p) h(p) \]  \hspace{0.5cm} (2.3)

where \( S \) is the collection of pixels determined by the streamline originating from \((u, v)\), \( T_{\text{in}}(p) \) is the colour value of the input texture at \( p \), and \( h(p) \) is the convolution filter.

LIC has several advantages such as indicating vector field direction in every point, indicating local information such as convergence, producing 2D or 3D textures which can be visualized using standard rendering techniques, and producing images with a fine spatial resolution. Moreover, unlike streamlines, LIC does not depend on the selection of good seed points. The introduction of LIC has triggered research to a large amount of variations. This research includes extending the concept by, for example, adding flow orientation cues, flow velocity cues, and extending LIC to three dimensions. Laramee et al. [2004a] give an excellent overview of the various LIC algorithms.

De Leeuw and van Liere [1998] give a detailed comparison between spot noise and LIC, and conclude that the spatial resolution for presenting directional information with LIC is higher than for a comparable spot noise texture. Spot noise is more flexible with respect to trading texture quality for computational speed.

### 2.1.3 Visualizing multiple fields

Finally, multi-field datasets are any datasets that comprise a combination of two or more fields of the above types, for example a medical dataset with both a CT scan and an MRI scan of the same patient, or computational fluid dynamics. This type of dataset is the main focus of this thesis. In this section, research related to the visualization of multiple fields of data is discussed.

**Multiple Views** The term "multiple views" has been used in various ways by different authors. Roberts [2007] mentions that it is a general term describing any instance where data is represented in multiple windows. Usually these windows contain different representations and often operations on the views can be linked or coordinated: hence the terms *multiple coordinated views* and *multiple linked views* are used as well. The principle idea of this method is that by using side-by-side views, users can easily compare the data from two or more representations. Wang Baldonado et al. [2000] give a set of guidelines for the use of multiple views in information visualization. A well designed multiple views system can enhance a users ability to compare data and switch context; a disadvantage of the method is the additional display space requirements for each extra view.

**Oriented Sliver Textures** This is a texturing technique that deals with multiple overlapping scalar fields [Weigle et al., 2000]. It depends on orientation and brightness to visualize several fields in one image. It makes use of an exemplar texture called a "sliver" (a simple,
2.1 Advancements in Scientific Visualization

Background and Related Work

Figure 2.8: Top left: vector field, top right: input texture (white noise), bottom left: output pixel, bottom right: example of a LIC visualization. To compute the output pixel value of the pixel indicated by a blue outline, we use the vector field to determine a streamline with a kernel size of, in this case, 8 pixels. All pixels of the input texture that lie under this kernel are convoluted - in the simplest case by a weighted addition of the pixels under the streamline - resulting in the colour as shown in the output pixel. Vector magnitude can be represented as well, either by colour mapping or by varying the kernel size.

straight line - 10% of the texture consists of the sliver). Each scalar field is associated with a unique, easily distinguishable orientation between $0^\circ - 180^\circ$, and for each field, at every data point the base sliver texture is placed and rotated according to the orientation associated with that field. The brightness of the sliver is a function of the field value (e.g. the maximum value in the particular scalar field is white or full luminance, while the minimum value corresponds to black or zero luminance). In the resulting image, the sliver textures for each scalar field are layered on top of each other, and patterns in the data become visible. The authors conducted psychophysical experiments, which results suggest that up to 15 orientations can be rapidly and accurately differentiated. An example visualization is shown in figure 2.9, the concept of this method is explained in figure 2.10.

This technique makes use of the orientation and luminance of simple icons to represent information. Orientation is used to discriminate between different fields of data, while luminance is used to encode the magnitude of the (scalar) field values. An advantage of this
Table 2.3: Techniques to visualize vector fields. $x \in X$ is a data-point, and $x \in \mathbb{R}^{2/3}$. $\rho(x)$ is the field value vector at $x$. $\Delta_x$ is the physical distance on the $x$-axis that a vector is drawn. In the case of particle advection we show the function to update the location of particles; $P_i(t)$ denotes the location of particle $i$ at time $t$.

<table>
<thead>
<tr>
<th>Visual Attributes</th>
<th>Spatial Density</th>
<th>Information Content</th>
<th>Generating Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arrow Plots</strong></td>
<td>Discrete</td>
<td>Point</td>
<td>$f : \rho(x) \to (\Delta_x, \Delta_y, \Delta_z)$</td>
</tr>
<tr>
<td><strong>Particle advection</strong></td>
<td>Discrete</td>
<td>Point</td>
<td>$f : P_i(t + 1) = P_i(t) + \rho(x_i, y_i)$</td>
</tr>
<tr>
<td><strong>Streamlines</strong></td>
<td>Discrete</td>
<td>Line</td>
<td>$\frac{dx}{dt} = v(x(t)), x(0) = x_0$</td>
</tr>
<tr>
<td><strong>Spot noise</strong></td>
<td>Continuous</td>
<td>Surface</td>
<td>$f(x) = \sum \alpha_i h(x - x_i)$</td>
</tr>
<tr>
<td><strong>LIC</strong></td>
<td>Surface</td>
<td>Line</td>
<td>$T_{out}(u, v) = \sum_{p \in S} T_{in}(p) h(p)$</td>
</tr>
</tbody>
</table>

Figure 2.9: Oriented sliver textures visualization: eight sliver layers oriented at intervals of $15^\circ$ each. Image courtesy of Weigle et al. [2000].

is that it leaves other colour attributes (such as hue and saturation) open for other uses, as well as other attributes such as size and density of slivers. A disadvantage of this method is that it is designed for the visualization of scalar fields only; another is that it is difficult to intuitively map data fields to an orientation. This is particularly true in the case of many different fields, where orientation angles are close together. Even though the angles may be perceivably different in the image, it is a difficult task to remember which data field is associated with which orientation - the use of hue to enhance the distinguishability of
important fields can help solve this problem.

**Perceptually Based Brush Strokes for Nonphotorealistic Visualization** In their paper, Healey et al. [2004] present the use of impressionist style brush strokes to visualize a multi-field dataset. After segmenting the dataset into a number of non-overlapping, spatially connected regions with region growing, a region-global stroke coverage is determined for each region. This coverage represents the attribute field on which the original segmentation was done. Then strokes are painted at randomly selected positions within the region. The properties of each stroke (their colour, size and orientation) are determined by the data element that is spatially closest to the location of the stroke; up to three more data fields can be represented by these different stroke properties. By segmenting the dataset into regions, the authors find an analogy to painting different objects on a canvas separately - for each "object" there is an appropriate stroke coverage. The result of their algorithm is a painted image, where the color and texture patterns can be used to explore and analyze information stored in a multi-field dataset. An example visualization presented in their paper is shown in figure 2.11.
2.2 Visual Perception

In this section we discuss relevant concepts from cognitive science relating to perception of the human visual system (HVS). Visual perception is the ability of the HVS to organize, identify, process and interpret information contained in visible light that is perceived by the visual senses, in order to understand the surrounding environment. Understanding visual
2.2 Visual Perception Background and Related Work

Perception is clearly very important in the field of data visualization, as the interpretation of a visualization depends on the visual perception of the image. An effective visualization should therefore exploit the capabilities of the HVS. Dastani [2002] formulate the effectiveness of a visualization as follows: "a visualization presents the input data effectively if the intended structure of the data and the perceptual structure of the visualization coincide".

Perception is a result of complex processes. In figure 2.13 the perceptual process is shown. This process is divided in four categories: stimulus refers to what stimulates our receptors; in the case of visual perception this is visual light. Electricity refers to the signals created by the receptors and transmitted to the brain. Experience and action refers to perceiving, recognizing and reacting to stimuli, and knowledge refers to the knowledge we have about the perceptual situation. This model of the perceptual process is explained in detail in [Goldstein, 2010].

2.2.1 Gestalt

Gestalt is a German word for "form" or "shape". In the English language, it is used referring to the concept of "wholeness". Gestalt psychology is a theory of mind and brain which originates from the Berlin School of Experimental Psychology. The theory states that the brain is holistic (i.e. it sees natural systems as a whole, as opposed to a collection of parts) and that it has self-organizing tendencies. The human visual system can visually recognize shapes, forms and whole figures instead of a collection of lines and curves. Koffka [1935] explains gestalt by the phrase "the whole is other than the sum of the parts", which can often be found in literature as the mistranslation "the whole is greater than the sum of the
The gestalt laws of grouping describe the way the human mind structures and organizes parts of external stimuli as a whole. The fundamental principle is the law of prägnanz (German for "concise": clear and succinct): it states that our brain organizes or reduces reality to the simplest form possible. An example of this principle is shown in figure 2.14. Other laws of grouping are the following:

- The law of similarity: elements that are similar tend to be grouped together, regardless of whether or not the relationship actually exists.
- The law of proximity: elements that are near each other tend to be grouped together regardless of the existence of a relationship.
- The law of continuity: elements that follow an established direction tend to be grouped together, as opposed to elements that are defined by sharp and abrupt changes in direction. When there is an intersection between two or more objects, we tend to perceive the objects as individual uninterrupted entities.
- The law of closure: the mind tends to see complete forms even if an image is incomplete or partially hidden. If part of a shape’s border is missing, for example, we tend to ignore the gaps and perceive the complete shape.
- The law of common fate: elements that are seen moving in the same direction and at the same speed, are perceived as a group. An example is a flock of birds moving uniformly. When two flocks of birds cross each other in a human’s visual field, they will still be perceived as two separate flocks as each bird has a direction common to its flock.
Figure 2.14: The law of prägnanz: in this image we see a series of three shapes (rectangle, circle and triangle), rather than many more complicated shapes.
Chapter 3

Design and evaluation of multi-field visualization techniques

This chapter contains an extension of the contents of a paper that was published in the proceedings of the 27th Image and Vision Computing New Zealand conference (IVCNZ’12). The paper describes a user study which was carried out as part of the research for this thesis. The goal of the user study was to evaluate four image compositing techniques to visualize two 2D vector fields in a single image [van Egmond et al., 2012]. The sections in this chapter follow the outline of the publication, starting with the abstract.

3.1 Abstract

The visualization of vector fields is essential for many applications in science, engineering and biomedicine. A large number of vector icons has been developed, but little research has been done on their effectiveness, especially when visualizing multiple vector fields simultaneously. We apply research in visualization and cognitive science to identify four classes of post-processing techniques for visualizing two 2D vector fields simultaneously: blending, overlay, bump mapping, and masking. We apply these four post-processing methods to Line Integral Convolution (LIC) textures and thus develop several novel multi-field visualization techniques. We evaluate their effectiveness with a user study requiring participants to locate and classify singularities, and to rate each method on its effectiveness and aesthetic value. The results of the study suggest that blending is the most effective technique to combine multiple vector field visualization textures, while masking performs worst. There is some evidence that visualizations with smooth colour changes are perceived as visually more attractive, and that aesthetics increases the perceived effectiveness of a visualization technique.

3.2 Introduction

Vector field visualization is an important task in scientific visualization. A large variety of techniques has been proposed ranging from vector glyphs [Klassen and Harrington, 1991]
3.3 Literature Review

Design and evaluation of multi-field visualization techniques

and textural representations [van Wijk, 1991; Cabral and Leedom, 1993], to the visualization of structural information such as the vector field topology [Reininghaus et al., 2011; Helman and Hesselink, 1989].

One central issue of current research is multi-field visualization [Moorhead et al., 2006]. Novel image acquisition and simulation techniques have made it possible to record a large number of co-located data fields. Visualizing them simultaneously facilitates the investigation of correlations and relationships between the fields [Urness et al., 2006]. An example is the visualization of multi-field cardiac imaging data (MRI, PET, CT, DTI, tagged MRI, molecular imaging) in order to analyse the relationship between cardiac function, structure, anatomical changes, metabolic activity, blood perfusion, and cellular remodelling.

In this paper we investigate texture-based visualization techniques for displaying two 2D vector fields simultaneously. Section 3.3 reviews previous work on vector field visualization and multi-field visualization. Section 3.4 explains the design of our multi-field visualization techniques. Section 3.5 presents an evaluation of the techniques with results being discussed in section 3.6. We conclude this paper and give an outlook on future work in section 3.7.

3.3 Literature Review

Texture-based representations are popular for vector field visualization due to their high spatial resolution (vector information is displayed for many points of the domain with comparatively little screen space) and the ease with which they can be applied to different 2D and 3D geometries [Laramee et al., 2004b].

One of the earliest classes of techniques in this domain is “Spot noise”, which distributes a large number of spots over a surface and transforms them according to the underlying vector field [de Leeuw and van Wijk, 1995]. A contrasting approach is to place streamlines close together until the visualization domain is covered and then separating them [Jobard et al., 2002].

The arguably most popular texture-based vector field visualization technique is Line Integral Convolution (LIC) [Cabral and Leedom, 1993]. The method places short streamlines over each pixel of an input texture (usually white noise), and then applies a convolution kernel to the texels covered by each streamline. LIC has several advantages such as indicating vector field direction in every point, indicating local information such as convergence, producing 2D or 3D textures which can be visualized using standard rendering techniques, and producing images with a fine spatial resolution. Moreover, unlike streamlines, LIC does not depend on the selection of good seed points.

Multi-field datasets contain several fields defined over the same domain. Blaas et al. introduce interaction techniques for selecting and comparing components of interest [Blaas et al., 2007]. Kniss et al. present novel transfer functions for direct volume rendering high-dimensional volume data sets [Kniss et al., 2003]. Weigle uses rectangular texture elements to encode multiple variables [Weigle et al., 2000]. Fuchs and Hauser give an overview of state-of-the-art techniques for visualizing multi-field data [Fuchs and Hauser, 2009].
Much can be learned from observing the occurrence of complex textures in other application fields. Interrante suggests to harness the intricate variety and subtle richness of detail of natural texture patterns [Interrante, 2000]. Using artistic concepts and associating vector information with brush strokes in an oil-painting makes it possible to combine multiple layers of visual elements to a surface texture encoding multiple fields, while still allowing perception of each layer [Kirby et al., 1999].

Little work has been done on comparing the effectiveness of vector field visualization techniques. Laidlaw et al. investigated the effectiveness of six visualization methods for two dimensional vector fields by means of a user study [Laidlaw et al., 2005]. Urness et al. present strategies for visualizing multiple 2D vector fields by overlaying texture and glyph-based techniques, but do not evaluate their algorithms. Our research identifies key concepts for combining texture-based visualizations, presents several new techniques, and evaluates them with a user study.

3.4 Design

We created LIC textures of vector fields using Voreen [Meyer-Spradow et al., 2009] and used concepts from graphics and cognitive psychology to combine them. Except for the bump map, all vector fields were rendered using an implementation of the fast LIC algorithm [Stalling and Hege, 1995] in Voreen (sampling each pixel and using a large convolution kernel $k = 100$). Vector field magnitude is usually encoded by colour, but in order to compare the different techniques we only display vector field direction and use colour to perceptually differentiate multiple fields. The implementation of the following visualization methods is explained in more detail in appendix B.3.

3.4.1 Blend

Alpha blending is frequently used in computer graphics to simulate transparencies or combine different effects such as shading and texture mapping. In order to achieve effective blending we exploit chromatic adaptation, i.e., the human visual system’s ability to adjust
to changes in illumination and preserve the appearance of object colours [Fairchild, 2005].

We hence combine two textures $I_1$ and $I_2$ by encoding them with different hues (gray and yellow) and, as in the original LIC paper, vary the intensity for the white noise texture. The textures are then blended using an alpha value $\alpha \in [0.4, 0.6]$.

$$I_{\text{result}}(x, y) = \alpha I_1(x, y) + (1 - \alpha) I_2(x, y)$$ (3.1)

The left hand image of figure 3.1 shows an example.

### 3.4.2 Overlay

Figure-ground perception describes the observation that an object can be instantly perceptually separated from its background [Schiffman, 1996]. This is due to physically different attributes of a figure and its background but is also influenced by size, angle, and association with meaningful shapes. We employ this concept to overlay two LIC textures by rendering them using different hues (like above) and replacing each pixel of texture $I_1$ with the corresponding pixel of texture $I_2$, where it exceeds a threshold $t$:

$$I_{\text{result}}(x, y) = \begin{cases} I_1(x, y) & \text{if } I_2(x, y) \leq t \\ I_2(x, y) & \text{if } I_2(x, y) > t \end{cases}$$ (3.2)

The image in the middle of figure 3.1 shows an example.

### 3.4.3 Bump Map

Shape perception is dominated by the curvature of the silhouette contour (figure-ground boundary) and 3D surface shading [Humphreys, 1992]. We exploit this property by combining two textures by representing one of them using shading differences which simulate surface deformations. This concept is called bump mapping in computer graphics. In our implementation we create a bump map by rendering the second vector field using a much lower sampling rate of the LIC algorithm, and applying a Roberts’ Cross diagonal gradient filter [Roberts, 1963] $R$ to calculate edges. This gradient image is multiplied with the first image, resulting in the second vector field appearing as “bumps”. In our subsequent experiments we will use a crisp version explained above (equation 3.3), and a smooth version where the bumpmap is first convolved with a 3 by 3 Gaussian kernel $G$ (equation 3.4 and figure 3.1 (right)).

$$I_{\text{result}}(x, y) = I_1(x, y)(I_2 * R)(x, y)$$ (3.3)

$$I_{\text{result}}(x, y) = I_1(x, y)((I_2 * R) * G)(x, y)$$ (3.4)

### 3.4.4 Pattern Mask

The concept of Gestalt originates from the fine arts and expresses the notion that the “whole contains more information than the parts”. An example is the perception of a circular arrangement of symbols as a circle. Perception of Gestalt is influenced by proximity, similarity, continuation, closure, symmetry, and the law of Prägnanz, which states the the eyes tend
3.5 Evaluation

In order to evaluate the above presented visualization techniques we created seven unique pairs of different vector fields. Each vector field was represented by a $600 \times 600$ pixel
3.5 Evaluation

Design and evaluation of multi-field visualization techniques

Figure 3.3: Web interface used in the user study. The screen shot shows a critical point in the yellow vector field (indicated as transparent red circle) located by clicking on the canvas. The click has triggered a context menu enabling the user to select the type of critical point. After locating and classifying all identified critical points clicking the button at the bottom displays the next experiment.

texture, and contained 1 to 3 critical points (nodes, center, focus or saddle). Consequently each simultaneous visualization of two vector fields contained 2 to 6 critical points. The precise location and type of each critical points was stored in a database, and compared with the user input. A total of thirteen different visualization techniques, using the four concepts introduced above, were evaluated.

3.5.1 Methodology

We created 91 visualizations by using seven pairs of vector fields and thirteen visualization techniques employing the four concepts introduced in section 3.4. An online survey was created showing for each of the 13 visualization techniques one image displaying one randomly selected pair of vector fields (each vector field occurred equally often). The first page of the survey contained a quick explanation of the tasks (see subsection 3.5.2).

Removing Bias: The participants were not given any form of training other than an introductory text and example images. In order to counter “learning bias”, the order in which the visualizations were tested was randomised for each participant. Also, to avoid bias based on some vector field potentially being more difficult than others, an image was rendered for each combination of visualization methods and vector field pairs.

Participant Demographics: The user study had 55 participants: 27 participants with
experience in the field of data visualization ("experts") and 28 "non-experts". Most of the "experts" were researchers or students from the computer graphics research groups at the University of Auckland and the University of Technology, Delft. The survey took roughly 15 minutes. Participation was voluntary, anonymous, and no compensation was given.

3.5.2 Required Participant Tasks

To test the effectiveness of each of the visualization methods, participants were required to perform the following tasks for each visualization:

Locate and Classify Critical Points

For each visualization participants were required to locate and classify all critical points by first clicking on the precise location and then selecting the type of critical point (see figure 3.3). The visualizations were presented on a 600 by 600 pixels HTML5 canvas element. A click near a critical point triggered a context menu showing the four singularity types. We chose as admissible target area a circle with a radius of 22 pixels. In a preceding pilot study this provided sufficient distance between features which could be mistaken as critical points, while still taking into account errors due to rushed motions and lack of image information (e.g., when using masking patterns). The process was repeated until a participant clicked the "next experiment" button to indicate that he/she could not find any more singularities. The location of a click, as well as the selected singularity type, was recorded using JavaScript.

User Rating of Perceived Effectiveness and Aesthetic Value

Upon completing the above tasks participants were shown all 13 previous visualizations in identical order on a single page. Users had to rate each visualization both on its perceived effectiveness (ease to locate and classify critical points) and visually attractiveness. The questions were rephrased as statements (e.g., “I found this visualization effective”) and ratings were performed using seven-point Likert scales ranging from −3 (strongly disagree) to 3 (strongly agree).

The user rating was performed to detect any discrepancies between participants’ perception of effectiveness and the actual test results. The aesthetic value of a visualization techniques is of interest since it can effect its usage in commercial (business) applications. In addition studies have shown that the aesthetic value of a visualization is related to users’ ability and willingness to successfully fulfill visualization tasks [Cawthon and Moere, 2007; Moere and Purchase, 2011].

Measured Data

For each visualization the selected location and type of critical point and the user rating was stored. In addition the time period for each visualization task was measured. From this data we derived statistical measures, such as the percentage of found critical points, the percentage of correct classifications, the number of wrongly located or classified critical points.
3.6 Results and Discussion

### 3.6.1 Locating and Classifying Critical Points

For each participant we computed the percentage of correctly located and classified critical points, and the percentage of incorrect choices (i.e., either wrong position or wrong classification). Table 3.1 shows in the first two columns for each of these measures the mean value and standard deviation.

In order to quantify the overall effectiveness and accuracy of each method, we calculated $F_1$-scores [van Rijsbergen, 1979] using equation 3.5. Precision is the number of correctly classified critical points divided by the total number of clicks, and recall is the number of correctly classified critical points divided by the total number of critical points that should have been found.

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (3.5)$$

Figure 3.4 shows a plot of the mean and 95% confidence interval of the $F_1$-score for each visualization technique. The blend method has the highest score, closely followed by the overlay and crisp bump map methods. The masking methods perform far worse than the other three methods; even the best performing masking method (the coarse diagonal pattern) scores lower than 79% accuracy.

<table>
<thead>
<tr>
<th>Visualization Method</th>
<th>$\mu_{\text{Correct (in%)}}$</th>
<th>$\sigma$</th>
<th>$\mu_{\text{False (in%)}}$</th>
<th>$\sigma$</th>
<th>$\mu_{F_1}$</th>
<th>$\sigma$</th>
<th>$\mu_{\text{Time (in s)}}$</th>
<th>$\sigma$</th>
<th>$\mu_{\text{Effectiveness user rating (in range [-3,3])}}$</th>
<th>$\sigma$</th>
<th>$\mu_{\text{Visual Attractiveness user rating (in range [-3,3])}}$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blend</td>
<td>94.6</td>
<td>14.7</td>
<td>1.4</td>
<td>5.7</td>
<td>0.952</td>
<td>0.118</td>
<td>24.89</td>
<td>13.60</td>
<td>1.56</td>
<td>1.44</td>
<td>1.11</td>
<td>1.65</td>
</tr>
<tr>
<td>Overlay</td>
<td>89.5</td>
<td>20.3</td>
<td>0.6</td>
<td>3.1</td>
<td>0.919</td>
<td>0.179</td>
<td>24.62</td>
<td>14.45</td>
<td>1.78</td>
<td>1.36</td>
<td>1.06</td>
<td>1.45</td>
</tr>
<tr>
<td>Crisp bump map</td>
<td>88.7</td>
<td>19.7</td>
<td>1.4</td>
<td>5.7</td>
<td>0.917</td>
<td>0.173</td>
<td>25.33</td>
<td>15.58</td>
<td>1.43</td>
<td>1.50</td>
<td>0.87</td>
<td>1.61</td>
</tr>
<tr>
<td>Smooth bump map</td>
<td>84.7</td>
<td>21.9</td>
<td>4.0</td>
<td>11.2</td>
<td>0.870</td>
<td>0.192</td>
<td>24.46</td>
<td>12.28</td>
<td>1.43</td>
<td>1.53</td>
<td>0.94</td>
<td>1.67</td>
</tr>
<tr>
<td>Mask: f. diagonal</td>
<td>80.1</td>
<td>24.5</td>
<td>6.3</td>
<td>16.0</td>
<td>0.832</td>
<td>0.228</td>
<td>32.07</td>
<td>17.06</td>
<td>0.02</td>
<td>1.67</td>
<td>-0.41</td>
<td>1.55</td>
</tr>
<tr>
<td>Mask: c. diagonal</td>
<td>81.0</td>
<td>22.7</td>
<td>2.6</td>
<td>8.2</td>
<td>0.864</td>
<td>0.183</td>
<td>27.02</td>
<td>13.86</td>
<td>0.11</td>
<td>1.58</td>
<td>-0.48</td>
<td>1.52</td>
</tr>
<tr>
<td>Mask: vertical</td>
<td>81.5</td>
<td>24.1</td>
<td>3.0</td>
<td>11.1</td>
<td>0.858</td>
<td>0.201</td>
<td>30.49</td>
<td>21.21</td>
<td>0.04</td>
<td>1.53</td>
<td>-0.41</td>
<td>1.64</td>
</tr>
<tr>
<td>Mask: horizontal</td>
<td>80.3</td>
<td>23.8</td>
<td>2.9</td>
<td>9.6</td>
<td>0.855</td>
<td>0.188</td>
<td>31.58</td>
<td>41.47</td>
<td>0.11</td>
<td>1.56</td>
<td>-0.39</td>
<td>1.45</td>
</tr>
<tr>
<td>Mask: f. checker</td>
<td>79.3</td>
<td>24.8</td>
<td>1.5</td>
<td>6.4</td>
<td>0.842</td>
<td>0.199</td>
<td>32.66</td>
<td>22.05</td>
<td>-0.56</td>
<td>1.57</td>
<td>-0.83</td>
<td>1.54</td>
</tr>
<tr>
<td>Mask: c. checker</td>
<td>77.6</td>
<td>26.7</td>
<td>5.6</td>
<td>15.8</td>
<td>0.822</td>
<td>0.240</td>
<td>33.47</td>
<td>22.14</td>
<td>-0.20</td>
<td>1.55</td>
<td>-0.57</td>
<td>1.45</td>
</tr>
<tr>
<td>Mask: long weave</td>
<td>78.2</td>
<td>25.8</td>
<td>4.4</td>
<td>16.1</td>
<td>0.835</td>
<td>0.230</td>
<td>31.84</td>
<td>14.57</td>
<td>1.22</td>
<td>1.46</td>
<td>-1.06</td>
<td>1.55</td>
</tr>
<tr>
<td>Mask: solid weave</td>
<td>77.6</td>
<td>25.2</td>
<td>4.2</td>
<td>15.0</td>
<td>0.830</td>
<td>0.212</td>
<td>33.69</td>
<td>18.18</td>
<td>1.09</td>
<td>1.43</td>
<td>-1.13</td>
<td>1.48</td>
</tr>
<tr>
<td>Mask: grad. weave</td>
<td>77.4</td>
<td>27.4</td>
<td>2.7</td>
<td>10.4</td>
<td>0.825</td>
<td>0.234</td>
<td>36.26</td>
<td>24.26</td>
<td>-1.37</td>
<td>1.41</td>
<td>-1.00</td>
<td>1.67</td>
</tr>
</tbody>
</table>

Table 3.1: Results of the user study. The rows represent the visualization methods (c. and f. stand for coarse and fine patterns). The columns show the mean and standard deviation of respectively **singularities correct** (fraction of critical points that were correctly classified), **singularities false** (fraction of clicks that were not on a critical point), **$F_1$ score** (the calculated $F$-score), **time** (the time in seconds it took participants to complete a visualization), **effectiveness user rating** (the rating given for effectiveness), and **visual attractiveness user rating** (the rating given for visual attractiveness).
pattern) has a significantly lower $F_1$ than any of the non-masking techniques (the smooth bump map performs only slightly better).

The results indicate that the human visual system’s ability of Gestalt continuation are not sufficient to reliably detect features in a vector field visualization. However, with roughly 80% correctness results are sufficiently good to justify further investigation in selected application cases where the other visualization techniques are not practical (e.g., if additional scalar fields are encoded by colour).

With regard to different masking patterns the coarse diagonal, vertical, and horizontal patterns worked best and also had a high perceived effectiveness. The weaving pattern with gradient and the fine checkerboard pattern rated worst. The results indicate that visual interpolation of missing data is most effective when using relatively large connected regions without boundaries.

We also measured the precision of correctly located critical points. Results were surprisingly similar. The fine diagonal pattern mask performed best with an average location error of 4.61 pixels, followed by overlay (4.65 pixels), blend (4.78 pixels) and smooth bumpmap (4.87 pixels). The worst location error was observed for the gradual weaving pattern (6.20 pixels).

Lastly, we noticed a difference in performance between the group of participants that had indicated to have experience in data visualization, and the group that had indicated not to have such experience. The former group (the experts) have an $F_1 = 0.891$ and an average correctness (correctly locating and classifying critical points) of 84.4%, while the latter group (the non-experts) have an $F_1 = 0.838$ and an average correctness of 78.6%. Per technique the expert group performs better than the non-experts as well, however for the blend technique this difference appears minimal. For the pattern mask methods, the drop in performance between the two groups is larger.

### 3.6.2 Response Time

We evaluated the efficiency of the presented visualization techniques by measuring the time to complete the visualization task (fourth column of table 3.1). The results of one participant were removed since the measured time was more than 2 hours, which suggests that the user interrupted the study at least once. Figure 3.5 demonstrates similar results as for the effectiveness. The response time for the blend, overlay and bump map methods are similar and faster than the best performing masking method (in this case the coarse diagonal pattern). The worst performing method is the weaving pattern with a gradual shadow. The results suggest that vector field perception by mental interpolation of patterns is considerable less efficient than perception using other visual attributes such as shading, chromatic adaptation, and figure-ground perception.

### 3.6.3 Effectiveness and Aesthetics Ratings

The last two columns of table 3.1 show the user ratings for the effectiveness and visual attractiveness of each visualization technique. The ratings that participants gave for the effectiveness are largely in line with the results for the error metric in equation 3.5, except
that users have rated the overlay method highest for effectiveness ($\mu = 1.78$) followed by blend ($\mu = 1.56$), while the test results suggest that blend is the most effective method followed by overlay. The smooth and crisp bump map follow with $\mu = 1.43$. All of the masking methods were rated significantly lower on the effectiveness scale, the horizontal pattern being rated the most effective with $\mu = 0.11$, and the gradual weaving pattern the least with $\mu = -1.37$. Similar results are obtained for the aesthetics ratings.

The smooth bump maps were rated as more aesthetic than the coarse bump maps and as similarly effective, even though their actual performance was inferior to the coarse bump maps. In general methods with smooth colour changes obtained a higher rating for visual attractiveness.

Figure 3.4: Mean and 95% confidence interval for the F-score. Both bump mapping techniques are shown as their performance differs significantly. For the masking technique, only the result for the best performing masking pattern (coarse diagonal) is shown.

Figure 3.5: Mean and 95% confidence interval of the time taken per experiment in seconds. For the masking technique, only the results for the best performing pattern (coarse diagonal) are shown.
A contradiction between user perception and performance was also observed for coarse and fine masking patterns. Whereas the fine checkerboard masking pattern performed better in terms of correctly and falsely classified singularities, users perceived the coarse masking pattern as more effective and visually more attractive.

The results suggest that aesthetics increases the perceived effectiveness of a visualization technique.

### 3.7 Conclusions and Future Work

Based on properties of the human visual system we have developed four classes of techniques for visualizing two 2D vector fields simultaneously: blending (alpha blending of two LIC textures), overlay (replacement of selected pixels in one LIC texture with those from another LIC texture), bump mapping (placement of an LIC texture on top of another LIC texture using shading differences simulating height variations) and pattern masking (division of an image into regions, each of which displays one field exclusively). The use of bump mapping and masking seems to be new in this application domain.

We evaluated the techniques with a user study and showed that all four concepts aid users with finding critical points in a vector field. Blending was most effective, followed by overlay and “crisp” bump maps. Similar results were obtained for efficiency. There were some contradictions between perceived and actual effectiveness. The results suggest that aesthetics increases the perceived effectiveness of a visualization technique, and that visualizations with smooth colour changes are perceived as visually more attractive.

In future work we want to test the presented visualizations using a larger variety of visualization tasks, and we want to investigate in more detail the relationship between aesthetics, perceived effectiveness and actual effectiveness.
Chapter 4

Visualizing multi-field data using texture grammars

In this chapter we present a novel method to visualize multiple two-dimensional fields of data in a single image. The concept is introduced in section 4.1. Then, in section 4.2, we explain the concept of formal grammars and their current role in scientific visualization. In section 4.3 we present a newly developed generative formal grammar which uses production rules to determine a configuration of visualization techniques and image compositing methods: the texture grammar. In section 4.4 we discuss possibilities to incorporate unobtrusive user-interaction to enhance the users ability to see patterns and relations in the data. Finally, in section 4.5 we give a few ideas about future work on the topic of texture grammars. Details on the implementation of the presented solution can be found in appendix B.

4.1 Concept

The concept of our solution is to incorporate knowledge about visualization techniques and image processing techniques in a novel generative grammar (the texture grammar). This generative grammar can then be used to automatically generate single-image visualizations that represent all data fields, based on the number of data fields and their type (in this thesis we focus on scalar and vector fields). To build successful texture grammars, we need to know what visualization techniques work well together, and which image compositing techniques are effective to visualize these fields in one image without losing important information. To encode multiple fields of data in a single image effectively (i.e. the low-level human visual system can process the data and find important features of all data fields efficiently) we should combine techniques that make use of perceptual cues that complement each other and do not interfere [Healey, 1998].
Figure 4.1: Production rules for a context-sensitive grammar that produces the language $L$ as shown in equation 4.1.

### 4.2 Formal Grammars

A formal grammar (a grammar in the context of formal language theory) consists elementarily of an alphabet and a set of production rules for strings in a formal language. The production rules describe how to form strings from the characters in the alphabet. Generative grammars were first formalized by Chomsky [1956]. In his formalization, a generative grammar $G$ is defined by the four elements in the tuple $< V_n, V_t, P, S >$:

- $V_n$ is the set of non terminal symbols
- $V_t$ is the set of terminal symbols
- $P$ is the set of production rules
- $S$ is the unique starting symbol

The set of non-terminal symbols $V_n$ can not appear in any string formed by $G$. The set of terminal symbols $V_t$ is disjoint from $V_n$, and every string in the language defined by $G$ (i.e. all strings that can be formed by the production rules in $G$) consists of only elements from $V_t$. The starting symbol $S$ is a non terminal symbol ($S \in V_n$). Any resulting strings are also called configurations. To illustrate the working of a formal grammar, consider the following example of a context-sensitive grammar, with production rules $P$ as shown in figure 4.1. This grammar has a set of non-terminal symbols $V_n = \{S, B\}$ and a set of terminal symbols $V_t = \{a, b, c\}$, and generates the language $L$ as specified by equation 4.1.

$$L = \{a^n b^n c^n | n \geq 1\}$$

(4.1)

This grammar is classified as context-sensitive, as it operates on production rules where the left hand side contains more than a single non-terminal symbol; replacement rule 4, for example, can only fire if the non-terminal $B$ is preceded by the terminal symbol $b$ (the context). When such a rule fires, all symbols on the left side (non-terminals and terminals) are replaced by all symbols on the right side (i.e. not just the non-terminal symbols get
4.3 Texture Grammar

In this section we formally introduce the texture grammar. First, we explain our reasoning behind the naming of this formal grammar. Then, we give a formal definition of the grammar, followed by a description of the production rules that make up a texture grammar. Finally, we give an example of how the texture grammar derives a configuration of visualization and compositing methods from a starting configuration, and we provide some visualizations generated using such a configuration.
4.3 Texture Grammar

Visualizing multi-field data using texture grammars

4.3.1 Etymology

Existing two-dimensional vector field visualization methods can be categorized into three distinct categories: texture-based, line-based or glyph-based [Urness et al., 2006]. Texture-based methods generate highly detailed and dense representations of the underlying data. Spot noise and LIC are arguably the most widely used texture-based techniques for the visualization of two dimensional vector fields. Line-based methods visualize vector fields using more sparsely distributed streamlines - data existing in the space between streamlines is not represented. These techniques give more of a global sense of the direction of the vector field. Glyph-based methods, finally, display repeatable two or three dimensional icons, using shape, colour or orientational characteristics to encode multivariate data.

As we aim to provide a spatially dense visualization of multiple fields (both scalar and vector) at once, our method can be categorized as "texture-based" - even if we do occasionally use line-based techniques to visualize individual fields. These line-based techniques might be bump-mapped over the visualizations of other fields, resulting in a textured visualization. Therefore, the name of the formal grammar we use in our tool is texture grammar.

4.3.2 Formal Definition of Texture Grammar

In this section we design a grammar that is a type-1 (context-sensitive) grammar in the Chomsky hierarchy [Chomsky, 1956]. In context-sensitive grammars, both the left-hand side as well as the right-hand side of the production rules may consist of a context of terminal and non-terminal symbols. This is more general than type-2 (context-free) grammars where the left-hand side of the production rules consist of only a single non-terminal symbol, but the context-sensitive properties are useful in our case; to determine a visualization technique to be used for a certain field, for example, the texture grammar will need to know the context of other visualization methods used in order to determine the most efficient technique, i.e., a technique that complements the other techniques (by not obscuring information encoded by other methods). Following the formal definition of the grammar given by Chomsky [1956] as explained in section 4.2, we define our texture grammar as follows:

Let $TG = <V_n, V_t, P, S>$ be a texture grammar, where

- $V_n$ is the finite set of non terminal symbols, containing symbols for data fields
- $V_t$ is the finite set of terminal symbols, containing the set of symbols for visualization techniques, as well as the set of symbols for image compositing techniques
- $P$ is the set of production rules
- $S$ is the unique starting symbol

In table 4.1 we show the set of non terminal symbols that we have used in our experimenting with the texture grammar in its current state. Table 4.2 shows the set of terminal symbols. These two sets are limited to the currently implemented visualization and compositing techniques in the prototype Voreen module and accompanying fragment shader;
Table 4.1: Set of non terminal symbols $V_n$ in texture grammar $TG$.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>Unique starting symbol.</td>
</tr>
<tr>
<td>$[S_{x}]$</td>
<td>Scalar field.</td>
</tr>
<tr>
<td>$[V_{x}]$</td>
<td>Vector field.</td>
</tr>
</tbody>
</table>

Table 4.2: Set of terminal symbols $V_t$ in texture grammar $TG$.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bullet_x$</td>
<td>Any terminal symbol.</td>
</tr>
<tr>
<td>$[LIC(V_{x},k)]$</td>
<td>Line Integral Convolution applied to vector field $V_{x}$ with kernel size $k$.</td>
</tr>
<tr>
<td>$[SL(V_{x})]$</td>
<td>Streamlines applied to vector field $V_{x}$.</td>
</tr>
<tr>
<td>$[ARR(V_{x})]$</td>
<td>Arrow Plot applied to vector field $V_{x}$.</td>
</tr>
<tr>
<td>$[COL(S_{x})]$</td>
<td>Colour map applied to scalar field $S_{x}$.</td>
</tr>
<tr>
<td>$[ISO(S_{x},v)]$</td>
<td>Isolines applied to scalar field $S_{x}$. $v$ is an array of iso values.</td>
</tr>
<tr>
<td>$[CWA([\bullet_x],[\bullet_y],w)]$</td>
<td>Weighted addition applied to terminals $\bullet_x$ and $\bullet_y$ with weight $w$.</td>
</tr>
<tr>
<td>$[COV([\bullet_x],[\bullet_y],w)]$</td>
<td>Terminal $\bullet_y$ is overlaid over $\bullet_x$ for values greater than weight $w$.</td>
</tr>
<tr>
<td>$[CBM([\bullet_x],[\bullet_y])]$</td>
<td>Terminal $\bullet_y$ is bumpmapped on top of $\bullet_x$.</td>
</tr>
</tbody>
</table>

however, they can be extended arbitrarily when other techniques are implemented, or when the grammar is used in another visualization framework, depending on the techniques available to the specific framework.

Production rules in $P$ are formed according to the template shown in equation 4.2, where $\alpha, \beta \in (V_n \cup V_t)^*$ (they are strings of non terminals and terminals, including the empty string), $A \in V_n$, and $\gamma \in (V_n \cup V_t)^+$ (a non-empty string of non terminals and terminals). The fact that $\alpha$ and $\beta$ appear on the left side of a rule makes the texture grammar context-sensitive; they represent the context in which $A$ can transform into $\gamma$.

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$ (4.2)

A texture grammar should be so designed that every valid starting string (combination of data fields) leads to a single final configuration; the grammar should be deterministic. In section 4.3.3 below, we will explain how the production rules in a texture grammar are structured, and we will give an example of a texture grammar as it is implemented in our prototype.

### 4.3.3 Production Rules

Using the terminal and non terminal symbols as given in the previous section, we can now establish a set of production rules which should eventually generate a visualization configuration for every set of input fields. Production rules are evaluated in top-to-bottom order,
so we can give rules higher priority by placing them before other rules. An example of a set of 19 production rules is presented in figure 4.3.

To generate a configuration of visualization techniques, we first have to set the starting symbol production rule to match the input field types. Then we run through the texture grammar continuously - each time a rule fires, we update the current configuration, and run through the grammar again starting at the first rule. This goes on until we get a pass in which no more rules are fired. We have now reached a final state in which (if nothing went wrong) we have a configuration of exactly one terminal (which may contain several nested terminals). This terminal represents the final visualized image, built up from composition techniques and visualization techniques linked to each field of data.

4.3.4 Example Visualizations

When we run the texture grammar specified in figure 4.3 on four input fields (one scalar field and three vector fields, as specified in the first production rule), this would result the sequence of events specified in table 4.3. We end up with a configuration where we start with a weighted addition of the LIC texture of vector field $V_1$ and the colour map of scalar field $S_1$. Then an arrow plot representation of vector field $V_2$ is overlaid, and finally the image is bump mapped by a streamline representation of vector field $V_3$. Figure 4.4 shows a render of the four synthetic data fields visualized according to this final configuration.

When we analyse this visualization, we can clearly see distinct features in all four fields of data. By overlaying only those arrows with high luminance (representing the longer vectors), other regions remain unaffected and the important features of the LIC render are still clearly visible. This might be more problematic when both vector fields have important features in the same locations. The bump-mapped streamline visualization hardly obscures any information, and we can clearly discern the scalar values in the colour map. In figure 4.5 we see another example of the texture grammar applied to four data fields. In this image we show step by step which visualization and compositing techniques are applied.

4.4 User Interaction

The purpose of the texture grammar is to automatically evolve a visualization of multiple fields of data in a single image, which attempts to capture the important characteristics of the underlying fields, as well as clearly visualize relations between the fields. While the method does not depend on user interaction, at least not beyond connecting fields of data and selecting (or writing) a texture grammar, there are several techniques that can aid a user in 'browsing' the data more effectively, if needed.

Firstly, integrated in the concept of texture grammars is the possibility for users to customize their own texture grammar. This allows a user to choose parameters and customize the visualization techniques they would like to see applied to the data. Besides this integral
1. $S \rightarrow [S_1][V_1][V_2][V_3]
2. [V_4][V_5][V_6] \rightarrow [LIC(V_x, 50)][ARR(V_y)][SL(V_z)]
3. [V_4][V_5] \rightarrow [LIC(V_x, 50)][ARR(V_y)]
4. [V_4] \rightarrow [LIC(V_x, 50)]
5. [V_4] \rightarrow [ARR(V_y)]
6. [V_4] \rightarrow [SL(V_z)]
7. [S_4][S_5] \rightarrow [COL(S_4)][ISO(S_y, [0.1, 0.2, 0.5, 0.8])]
8. [S_4] \rightarrow [COL(S_4)]
9. [S_4] \rightarrow [ISO(S_x, [0.1, 0.2, 0.5, 0.8])]
10. [COL(S_4)][LIC(V_y, k)] \rightarrow [CWA([COL(S_4), [LIC(V_y, k), 0.5])]
11. [LIC(V_y, k)][COL(S_4)] \rightarrow [CWA([LIC(V_y, k), [COL(S_4), 0.5])]
12. [COL(S_4)][COL(S_4)] \rightarrow [CWA([COL(S_4), [COL(S_4), 0.5])]
13. [\bullet_1][ISO(S_y, \nu)] \rightarrow [C_OV([\bullet_1, [ISO(S_y, \nu)], 0.3])]
14. [ISO(S_x, \nu)][\bullet_2] \rightarrow [C_OV([\bullet_2, [ISO(S_x, \nu)], 0.3])]
15. [\bullet_1][ARR(V_y)] \rightarrow [C_OV([\bullet_1, [ARR(V_y)], 0.3])]
16. [ARR(V_y)][\bullet_2] \rightarrow [C_OV([\bullet_2, [ARR(V_y)], 0.3])]
17. [\bullet_1][SL(V_y)] \rightarrow [C_{BM}([\bullet_1, [SL(V_y)])]
18. [SL(V_x)][\bullet_2] \rightarrow [C_{BM}([\bullet_2, [SL(V_y)])]
19. [\bullet_1][\bullet_2] \rightarrow [C_{WA}([\bullet_1, [\bullet_2], 0.5])]

Figure 4.3: An example of a set of production rules for a texture grammar. The first (starting) rule is adjusted based on the data fields provided to the module - in this case, one scalar field and three vector fields. Rule number 2 and 3 make sure that if we have several vector fields in a row, we don’t use the same technique for each. Rule 7 does the same for scalar fields. Rules 13 to 18 trigger for combinations of isolines, arrowplots and streamlines with any other terminal. Rule 19 is in place to make sure we don’t end up with more than one terminal.
4.4 User Interaction

Visualizing multi-field data using texture grammars

Table 4.3: An example of a sequence of rules fired by the texture grammar as shown in figure 4.3, based on four input fields (one scalar and three vector fields). During the first six passes, a rule fires each time. Through the seventh pass, no rule is fired and we have reached a final configuration of one terminal representing the visualization. N.B. for brevity we have omitted the LIC kernel size parameter of 50.

<table>
<thead>
<tr>
<th>Pass</th>
<th>Current configuration</th>
<th>Rule fired</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$S$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$[S_1][V_1][V_2][V_3]$</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>$[S_1][LIC(V_1)][ARR(V_2)][SL(V_3)]$</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>$[COL(S_1)][LIC(V_1)][ARR(V_2)][SL(V_3)]$</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>$[C_{WA}([COL(S_1)], [LIC(V_1)], 0.5)][ARR(V_2)][SL(V_3)]$</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>$[C_{OV}([C_{WA}([COL(S_1)], [LIC(V_1)], 0.5)], [ARR(V_2), 0.5]), [ARR(V_2), 0.3])][SL(V_3)]$</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>$[C_{BM}([C_{OV}([C_{WA}([COL(S_1)], [LIC(V_1)], 0.5)], [ARR(V_2), 0.5]), [ARR(V_2), 0.3])][SL(V_3)])]$</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 4.4: Result of applying a texture grammar to three vector fields and one scalar field; one vector field is visualized using LIC and consequently alpha blended with a colour map visualization of the scalar field. A second vector field is visualized using streamlines, which are applied as a bumpmap over the image, and the third vector field is rendered as an arrow plot which is overlaid on top of the image.
Figure 4.5: Result of applying a texture grammar to three vector fields and one scalar field, resulting in a composition of four visualizations in a single image, very similar to the configuration after step 7 in table 4.3. a) a blue-to-red colourmap is applied to the scalar field. b) LIC is applied to vector field 1, and the result is blended with the result of step a. c) a streamline visualization is generated for vector field 2, and is subsequently bump-mapped over the result of step b. d) finally, an arrow plot is generated for vector field 3, and is overlaid on top of the result so far, to get our final composition.
4.5 Opportunities

Visualizing multi-field data using texture grammars

piece of user interaction, we discuss an additional method of user interaction that can be applied effectively to work in conjunction with the texture grammar: focus and context.

The principle behind focus and context in visualization is that users, while studying an image, are able to view in detail the object that is of primary interest to them, while still having an overview of the rest of the data presented at the same time. The detailed overview in this case is the focus, while the overview is the context. Focus and context start from three premises [Card et al., 1999]:

1. The user needs both overview (context) and detail information (focus) simultaneously.
2. Information needed in the overview may be different from that needed in detail.
3. These two types of information can be combined within a single (dynamic) display, much as in human vision.

To experiment with focus and context, we have implemented a magic lens [Bier et al., 1993]. The magic lens in our case views only a single visualization in a probed area by disabling the visualizations of the other data fields. This allows users to gain a greater, more detailed view of a selected field while retaining the context of the other data fields around it. This is particularly useful in the case that multiple fields have interesting features in the same location, where some features might be (partly) obscured by other visualizations. Figure 4.6 shows an example of the focus and context principles applied to a texture grammar visualization by means of the magic lens.

4.5 Opportunities

In this section we present some ideas that fell outside of the scope of this project, which we consider relevant and would recommend for further investigation. Currently, our definition and implementation of the texture grammar is a global one, in the sense that the grammar is run only once, before any visualization happens. This results in a single configuration of visualization and composition techniques, which stays the same throughout the complete output image - parameters such as the LIC kernel size are fixed as well. We would like for future research to investigate the possibilities of running a texture grammar for every pixel in the output image; this way, we can incorporate field properties as conditions for certain rules to fire. Karnick et al. [2009]; Li et al. [2011] give examples of how to query parameters of the field in a field-guided shape grammar in order to synthesize geometry on surfaces, as well as to use conditionals. We could also have variable parameters on the right side of production rules, where field values influence the behaviour of the visualization techniques. This would result in local configurations. This opens up more possibilities to emphasize certain fields in certain locations, where the grammar can be customized to determine important regions for each field. An example of such a field-guided production rule is shown in figure 4.7.
Visualizing multi-field data using texture grammars

4.5 Opportunities

Figure 4.6: The top two rows demonstrate user interaction by means of a magic lens after visualizing four fields of data using the texture grammar. In the top row we can see the arrowplot of a vector field (left) and the colour map of a scalar field (right) in focus. In the middle row, the focus is on the LIC render of a vector field (left) and the streamlines render of another vector field (right). The bottom image, finally, shows the resulting visualization without any user interaction.

Another improvement is the expansion of the current implementation with more visualization techniques (e.g. particle advection, spot noise). In order to do this, these techniques need to be implemented on the rendering module or the fragment shader. We also need to reserve a non terminal symbol in the texture grammar definition for each additional tech-
4.6 Results

In this chapter we have introduced the texture grammar, a novel method to visualize multiple data fields in a single image. The texture grammar is a generative formal grammar, that produces visualizations on the basis of a set of production rules that operate on a set of input data fields. The proposed concept allows us to encode general solutions to multi-field visualization problems in a compact format, while at the same time offering the possibility of customization in those cases where a general solution does not suffice. It offers a clear advantage to multiple views solutions in that the resulting visualization is a single image. As the data is spatial and sampled over the same spatial domain, visualizing it on the same image plane makes that trends and correlations in the data can be compared directly and more intuitively.

We do not strictly use the concept of a formal grammar in the sense that we do not use it to define a “language” containing a large (or even infinite) amount of words. We use a set of production rules, and adjust the first rule to map the starting symbol $S$ to a string of symbols representing the datasets connected to the texture grammar. In our case, using a formal grammar is a convenient and intuitive choice to map fields of data to configurations of visualizations. It allows us to define rules and constraints for the visualization of multiple fields in a single image, and contain all of those rules, including visualization parameters, in a single location.

---

**Figure 4.7:** An example of a production rule using a conditional and a variable replacement. The rule only triggers if the given condition $V_x.length \geq 0.1$ evaluates true ($V_x.length$ refers to the vector length which is normalized in the range $[0, 1]$). In this example, $LIC(V_x, k)$ stands for a LIC operation on vector field $V_x$ with kernel size $k$; this example rule illustrates how we could use field properties to manipulate parameters locally, using kernel sizes in the range $[5, 50]$ depending on the vector length.

1. $[V_x : V_x.length \geq 0.1 \rightarrow [LIC(V_x, V_x.length * 50)]]$
Chapter 5

Conclusions and Future Work

This chapter gives an overview of the project’s contributions. First, we summarize the presented methods and user study. Then we will discuss and reflect on the results and finally, we provide recommendations for the direction of future research.

5.1 Summary

In this thesis, we have presented a novel method for the visualization of multiple fields of data in a single image. We have developed a generative formal grammar which we have named texture grammar. This grammar takes note of the data fields that are provided as inputs, and generates a combination of visualization techniques for each field. It also determines which image processing techniques should be used to combine the visualizations of individual fields into a final composition in a single image. For this texture grammar a tool has been developed in C++ which takes as input a texture grammar file and a set of input field types, and can be queried on the resulting configuration. We have implemented a module in Voreen (a visualization library and development framework) which uses this tool to render visualizations of multiple fields of data based on a texture grammar.

We have also presented a user study in chapter 3, which was conducted to assess and evaluate the effectiveness of four image processing techniques (alpha blending / weighted addition, overlay, bump mapping, and pattern masking) in multi-field visualization. We applied these techniques in various ways to two 2D vector fields visualized by line integral convolution textures and thus investigated several novel multi-field visualization techniques. Participants of the study were asked to locate and identify critical points in the visualized vector fields, as well as to rate each visualization method on a seven-point Likert scale for both effectiveness as well as aesthetics. We calculated $F_1$-scores to quantify the effectiveness of each method, and we concluded that alpha blending was the most effective, followed closely by overlay and bump mapping. The pattern masking methods performed far worse than the other three methods. We also measured response time (the time needed by the user to complete each visualization task); interestingly, all pattern masking methods showed considerably slower response times than the other three methods, suggesting that the per-
ception of vector fields by mental interpolation of patterns is less efficient than perception using other visual attributes.

5.2 Discussion

The goal of this project was to investigate the possibilities of using a generative grammar to visualize multiple two-dimensional fields of data in a single image. To this end we have developed a novel formal grammar to generate configurations of visualization and image compositing methods based on the input field types. This method is generally applicable to many different scientific fields (e.g. medicine, geographical and oceanic data) where relations between multiple co-located fields of data need to be examined.

Currently, the implementation supports five different visualization techniques (two for scalar field visualization and three for vector field visualization), as well as three image compositing techniques. This can easily be extended by implementing new techniques and defining their symbols in the texture grammar interpreter. We have so far experimented using various synthetic fields of data; visualizations produced show that we can visualize multiple fields of data in a single image based on a texture grammar configuration. However, interesting regions of data fields can still be obscured in some cases. For example, when overlaying an arrow plot over an LIC render, where both vector fields have interesting features in the same location. When this happens, the largest arrows can obscure important information from the field rendered by LIC. In this light, more experimenting should be done particularly to determine important regions of the data fields (e.g. by utilising scalar and vector field topology), and how this information can be encoded in a texture grammar.

Our solution has the potential to be implemented as a module where several default texture grammars can be provided. These grammars can be written by domain experts to ensure domain specific features can be emphasized. End-users and researchers can easily write their own customized texture grammars as well.

5.3 Future work

During this project, we have identified various ideas about improvements that could be investigated and implemented in our solution. A first suggestion is to develop a more robust and scalable implementation of both an application (e.g. a Voreen module, but a choice could also be made for a different framework, or a stand-alone application) and a texture grammar interpreter. The code-base for this project has been developed in an experimental way and has continuously undergone functionality changes and additions as we learned more throughout the project.

The texture grammar presented in chapter 4 is currently implemented to run once, before any visualization happens; this results in a global configuration that is identical throughout
the complete output image. It would be very interesting to investigate a texture grammar that can run separately for every pixel in the output image, where rules can be fired, or parameters of visualization techniques can be adjusted, based on field values. To do this, we should make field values accessible from the texture grammar. These field values can then be used in conditionals added to rules, or in parameters. This is explained in more detail in section 4.5. Related to this, more research should be done to incorporate information on important regions in the data. If we can represent this information in the texture grammar, we can create emphasis on certain fields in certain locations, ensuring that important regions do not get obscured by other fields.

Another recommendation to future work is to expand the set of non terminal symbols of the texture grammar, by implementing more visualization and compositing techniques in the Voreen module, or by seeking a different visualization framework. Examples of potential additions include spot noise, particle advection or heightmaps (in which case a three dimensional view is needed where the user is able to rotate the camera around the visualization to perceive height differences). Another expansion worth investigating is the addition of tensor field visualization as a third field type, to cater for more complex data (example applications include stress and strain measurements in materials, or diffusion MRI in medicine). Also, in future work we would like to apply this technique in practice, e.g. on multi-modal image technologies (PET, MRI, CT, et cetera).

Finally, in chapter 3 we present a user study that investigates combinations of LIC renders of two vector fields (this work is published in [van Egmond et al., 2012]). It would be interesting to perform a similar user study on more than two fields and/or different combinations of visualization techniques and image compositing techniques. We also want to test the presented visualizations using a larger variety of visualization tasks, and we want to investigate in more detail the relationship between aesthetics, perceived effectiveness and actual effectiveness. Following this suggestion, we recommend the development of a method or metric to evaluate the effectiveness of multi-field visualizations as they are rendered using our texture grammar method.
Bibliography


50
BIBLIOGRAPHY


Appendix A

Glossary

In this appendix we give an overview of frequently used terms and abbreviations.

**CPU:** Central Processing Unit

**GLSL:** OpenGL Shading Language: high-level shading language based on the C programming language, used to program the GPU

**GPU:** Graphics Processing Unit

**HVS:** Human Visual System

**LIC:** Line Integral Convolution, a vector field visualization technique

**sampler2D:** A GLSL variable type representing a two dimensional texture
Appendix B

Implementation

This section provides details on the implementation of our novel multi-field visualization method, as conceptualised in chapter 4.

B.1 Development Framework

To implement our solution, we made use of the visualization framework Voreen\(^1\) (volume rendering engine). Voreen is a rapid prototyping framework for the visualization of volumetric data sets. It is implemented as a C++ library using OpenGL and GLSL for GPU-based rendering [Meyer-Spradow et al., 2009]. The interface follows a data-flow concept; visualization processors, data readers and viewports are implemented as modules that form building blocks. The user can connect these building blocks to form a data-flow network. Figure B.1 illustrates this.

![Figure B.1: An example of a Voreen data flow network for the visualization of anatomical structures. Image courtesy of The Voreen Team.](http://www.voreen.org/)

\(^1\)http://www.voreen.org/
Voreen provides clear documentation on extending it and integrating your own functionality by adding modules. We have implemented our solution as a Voreen module, called *MFTGSliceRenderer* (Multi-Field Texture Grammar slice renderer). In addition to this module, we have implemented a texture grammar parser and interpreter library in C++, which is included in our Voreen module. The MFTGSliceRenderer takes an arbitrary number of scalar and vector fields as input, as well as a texture grammar file. It uses the parser library to generate a texture grammar object which has all the symbols and production rules defined, and sets the input field types to the starting symbol $S$. Then it runs the texture grammar, and queries the resulting configuration to determine what visualization methods to apply to which fields. Some of these visualizations need to be calculated in the module itself; for example, the arrow plot shapes are generated using OpenGL commands from the MFTGSliceRenderer. In this case, the generated image is uploaded as a `sampler2D` to the fragment shader, which only needs to sample each pixel. Others, such as LIC, are calculated in our fragment shader: these fields are uploaded as `sampler2D` uniform variables as well, containing the information from the data field itself, and the fragment shader then calls a function that calculates the output pixel according to the specified visualization technique. The module uploads all information on visualization techniques and image compositing techniques to the shader which can then use this information to determine in what order to process the visualizations, and how to combine them. A schematic overview of this process is shown in figure B.2, and the GLSL code of our fragment shader, as well as the GLSL code for the visualization and image compositing techniques, is shown in appendix C.

### B.2 Visualization Techniques

In this section we will discuss the visualization techniques that we have used in our texture grammar. At the start of this project, we experimented with using implementations in the Voreen codebase of several techniques. However, techniques like line integral convolution are computationally expensive; performing all of these computations on the CPU proved to be very time consuming and resulted in far from interactive frame rates. The existing LIC implementation in Voreen also resulted in some artifacts that we could not explain.

For these reasons, we decided to write our own implementations of the techniques that we want to use, attempting to transfer a big chunk of the workload to the GPU by using fragment shaders. A GPU can manipulate graphics much more efficiently than a CPU, due to its highly parallel structure. So far, we have implemented and used the following visualization techniques to work with the texture grammar module (see appendix C for the GLSL code):

- **Colour mapping** is implemented completely in the fragment shader. We map scalar values (normalized to a range of $[0, 1]$) to their representing colours. Several colour scales can be used; to visualize our synthetic data we have used the heatmap scale, as shown in figure B.3. Different scales might be beneficial in different application domains however - it
Figure B.2: The Voreen module *MFTG SliceRenderer* takes two inputs: first, a collection of co-located scalar or vector data fields (these can be volumes, in which case a slice needs to be selected). Second, a texture grammar specified in a text file. This can be a default texture grammar or a customized, user-defined version. The parser/interpreter takes this texture grammar and takes note of the input data fields and their types - this information will be added in the starting production rule. The texture grammar runs, and the resulting combination of visualizations and compositing methods are sent to the data preparation part of the module. The data is uploaded to the fragment shader which also receives information from the texture grammar, and the rendered image is finally rendered to a display window.

Figure B.3: Heatmap colour scale, to map colours to normalized values in the range $[0, 1]$.

It would be wise to incorporate different scales, perhaps mapping different scales to different symbols in the texture grammar.

**Isolines** are implemented in GLSL in a fragment shader. It takes an array of isovalues between 0 and 1. For each pixel, it tests whether the value of the underlying scalar field at that location is an isovalue. If it is, the method returns a white colour, otherwise black.
Line Integral Convolution is implemented in GLSL in a fragment shader. At each output pixel, with kernel size \( k \), we travel a generated noise map forwards and backwards along the underlying vector field for \( \frac{k}{2} \) steps each way. We then set the output pixel value by taking the average value of all the pixels under the calculated streamline kernel.

Arrow plots are implemented on the Voreen module; using OpenGL, we draw arrows on a regular grid. Each arrow takes its colour and size property based on the vector magnitude of the data element nearest to its location.

Streamlines are implemented on the Voreen module as well. We take a number of seed points on a regular grid, and for each seed point we follow the stream by adding the vector value at that point to the positional coordinates. While following the stream, each output pixel we run through is set to white. All output pixels that were not touched by one of the streamlines remain black.

### B.3 Image Compositing Techniques

In our prototype application, we have implemented the three compositing methods that performed best in our user study (see chapter 3). We have decided not to incorporate pattern masking at this time, as masking multiple fields at the same time (as opposed to one field as was tested in the user study) greatly magnifies the amount of data that is obscured by the mask - which is not likely to make the use of this method any more effective than it was found to be in the user study. Below we give a short description of the implementation of the three compositing techniques (see appendix C for the GLSL code):

**Alpha blending** is implemented as a simple weighted addition operator, that takes two colour values \( c_1 \) and \( c_2 \) and a weight \( 0 \leq w \leq 1 \), and returns a colour according to the following formula:

\[
c_{\text{res}} = c_1 \cdot w + c_2 \cdot (1 - w)
\]

**Overlay** takes two colour values \( c_1 \) and \( c_2 \) and a threshold \( 0 \leq t \leq 1 \). It takes the intensity of \( c_1 \) (the maximum value of the red, green and blue channels). If this intensity exceeds the threshold \( t \), \( c_1 \) is returned; else \( c_1 \) is returned. \( c_2 \) is thus the colour of a pixel from the background image, and \( c_1 \) is the colour of a pixel from the foreground image which is only displayed if it exceeds the given intensity threshold.

**Bump mapping** takes a background colour \( c_1 \), a sampler2D of the texture that is to be bumpmapped, and the normalized coordinates in the output image. We then calculate the edge value at the specified location by applying a Roberts’ Cross diagonal gradient filter [Roberts, 1963] on all four channels (red, green, blue and alpha) and adding 1 to get values between 0 and 2. We then multiply these values with the background colour, darkening the image when the gradient values are between 0 and 1, and brightening the colour when
the values are between 1 and 2. This way, the second field appears as 'bumps’ onto the background image.

B.4 User interaction: magic lens

We have implemented a version of the magic lens, as explained earlier in this chapter. To use the magic lens, a user can click in a visualization, and drag the mouse cursor around. The lens appears around the center of the mouse cursor and by scrolling the mouse wheel the user can select which of the data fields should be in focus. Our Voreen module catches the mouse events and sends information about its location, the field in focus and whether or not the left mouse button is pressed down to the shader. The implementation of the lens visualization itself is done in GLSL on the shader: the code can be found in the first code snippet in appendix C. For each pixel we calculate its distance to the location of the mouse cursor. If this distance is smaller than the lens radius, we display only the selected field’s visualization (focus); otherwise we display the “regular” visualization of all fields together (context) (see figure 4.6). At the edges of the lens we display a gradual green border.
Appendix C

Fragment shader source code

In this appendix we show the GLSL source code of the fragment shader that we used to render visualizations as determined by the texture grammar, interpreted from our Voreen module.

```glsl
#include "visualizationtechniques.frag"
#include "compositingtechniques.frag"

uniform vec2 vpdims_;
uniform bool mouseDown_; 
uniform vec2 mouseCoord_; 
uniform int current_field_; 
uniform sampler2D noisemap_; 
uniform sampler2D data1_; 
uniform sampler2D data2_; 
uniform sampler2D data3_; 
uniform sampler2D data4_; 
uniform sampler2D data5_; 
uniform sampler2D data6_; 

// Method definitions, keep in sync w Voreen TextureGrammar parser
const int MFTG_FIELD_SCALAR = 000,
MFTG_FIELD_VECTOR = 001;
```

63
Fragment shader source code

MFTG_VECTOR_LIC = 100,
MFTG_VECTOR_ARROWPLOT = 101,
MFTG_VECTOR_STREAMLINES = 102,
MFTG_SCALAR_COLOURMAP = 200,
MFTG_SCALAR_ISOLINES = 201,
MFTG_COMP_OVERLAY = 400,
MFTG_COMP_BLEND = 401,
MFTG_COMP_BUMPMAP = 402;

// uniforms to determine what methods to use for techniques and blending
// we have (up to) 6 visualization techniques,
// and (up to) 5 compositing methods
uniform int number_of_fields;
uniform int tech1;
uniform int comp1;
uniform int tech2;
uniform int comp2;
uniform int tech3;
uniform int comp3;
uniform int tech4;
uniform int comp4;
uniform int tech5;
uniform int comp5;
uniform int tech6;

/**
 * Applies the selected visualization technique to the data field
 *
 * int visualizationid  The identifier to specify the visualization method
 * sampler2D field  The uploaded scalar field/vector field/texture
 * vec2 coord  The fragment coordinates
 */
vec4 applyVisualization(int visualizationid, sampler2D field, vec2 coord){
    vec4 col = vec4(0.0);
    switch(visualizationid){
        case MFTG_SCALAR_COLOURMAP:
            col = colourmap(field, coord, vpdims_);
            break;
        case MFTG_SCALAR_ISOLINES:
            col = isoline(field, coord, vpdims_);
            break;
        case MFTG_VECTOR_LIC:
            col = LIC(noisemap_, field, coord, vpdims_, 100);
            break;
        case MFTG_VECTOR_ARROWPLOT:
            col = arrowPlot(field, coord, vpdims_);
            break;
        case MFTG_VECTOR_STREAMLINES:
            col = streamlines(field, coord, vpdims_);
            break;
        default:
            break;
    }
    return col;
}
Fragment shader source code

```glsl
/**
 * Applies the selected composition method to two given colours. Note that
 * when bump map is the selected method, it is handled separately, as it
 * needs a sampler2D to compute its result.
 *
 * int blendid The identifier to specify the compositing method
 * vec4 c1 Colour of the first visualization
 * vec4 c2 Colour of the second visualization
 */
vec4 applyBlend(int blendid, vec4 c1, vec4 c2)
{
    vec4 col = vec4(0.0);
    switch(blendid)
    {
        case MFTG_COMP_BLEND:
            col = blend(c1, c2, 0.7);
            break;
        case MFTG_COMP_OVERLAY:
            col = overlay(c1, c2, 0.3);
            break;
        default:
            col = blend(c1, c2, 0.7);
            break;
    }
    return col;
}

float LENS_SIZE = min(vpdims_.x, vpdims_.y) / 4;
float SCALE_FACTOR = 1.00;
float ENLARGED_SIZE = LENS_SIZE * SCALE_FACTOR;
float RIDGE_SIZE = LENS_SIZE * 0.04;
vec4 RIDGE_COLOUR = vec4(0.3, 0.9, 0.6, 1.0);

/**
 * Calculates a nice gradual colour for the magic lens frame
 */
vec4 get_ridge_colour(float distn)
{
    float midridge = RIDGE_SIZE * 0.5;
    float distridge = LENS_SIZE - distn;
    float midridgedist = 1.0 - (abs(midridge - distridge) / midridge);
    return RIDGE_COLOUR * midridgedist;
}

void main()
{
    // stores the accumulated pixel colour
    vec4 col = vec4(0.0);
    vec2 fragcoord = gl_FragCoord.xy;

    // magic lens functionality
    vec2 middle = vec2(mouseCoord_.x, mouseCoord_.y);
    vec2 ptm = middle - fragcoord;
    float dist = sqrt(ptm.x*ptm.x + ptm.y*ptm.y);
```
if (dist < LENS_SIZE && mouseDown_){
   /*******************************
   * Simulate a magic lens if LMB pressed and this pixel *
   * is close enough to the lens center *
   *******************************/
   if (dist >= LENS_SIZE - RIDGE_SIZE){
      col = get_ridge_colour(dist);
   } else{
      // scale the vector pointing to the middle of the lens to
      // match the enlarged area proportions, and translate from
      // the lens centre to obtain coordinates in enlarged area
      ptm *= SCALE_FACTOR;
      vec2 lensCoord = middle - ptm;
      fragcoord = lensCoord;

      switch (current_field_){
         case 1:
            col = applyVisualization(tech1, data1_, fragcoord);
            break;
         case 2:
            col = applyVisualization(tech2, data2_, fragcoord);
            break;
         case 3:
            col = applyVisualization(tech3, data3_, fragcoord);
            break;
         case 4:
            col = applyVisualization(tech4, data4_, fragcoord);
            break;
         case 5:
            col = applyVisualization(tech5, data5_, fragcoord);
            break;
         case 6:
            col = applyVisualization(tech6, data6_, fragcoord);
            break;
         default:
            col = applyVisualization(tech1, data1_, fragcoord);
            break;
      }
   } else{
      // MFTG_COMP_BUMPMAP is an exception and is handled differently,
      // as it needs the sampler2D to calculate the gradient
      if (number_of_fields >= 1){
         col = applyVisualization(tech1, data1_, fragcoord);
      }
      if (number_of_fields >= 2){
         if (comp1 == MFTG_COMP_BUMPMAP){
            col = bumpmap(col, data2_, fragcoord, vpdims_, vec2(-1, -1));
         } else{
            vec4 col2 = applyVisualization(tech2, data2_, fragcoord);
            col = applyBlend(comp1, col2, col);
         }
      }
      if (number_of_fields >= 3){
         if (comp2 == MFTG_COMP_BUMPMAP){
            col = bumpmap(col, data3_, fragcoord, vpdims_, vec2(-1, -1));
         } else{
            vec4 col3 = applyVisualization(tech3, data3_, fragcoord);
            col = applyBlend(comp2, col3, col);
         }
      }
      } }
if (number_of_fields >= 4) {
    if (comp3 == MFTG_COMP_BUMPMAP) {
        col = bumpmap(col, data4_, fragcoord, vpdims_, vec2(-1, -1));
    } else {
        vec4 col4 = applyVisualization(tech4, data4_, fragcoord);
        col = applyBlend(comp3, col4, col);
    }
}
if (number_of_fields >= 5) {
    if (comp4 == MFTG_COMP_BUMPMAP) {
        col = bumpmap(col, data5_, fragcoord, vpdims_, vec2(-1, -1));
    } else {
        vec4 col5 = applyVisualization(tech5, data5_, fragcoord);
        col = applyBlend(comp4, col, col5);
    }
}
if (number_of_fields >= 6) {
    if (comp5 == MFTG_COMP_BUMPMAP) {
        col = bumpmap(col, data6_, fragcoord, vpdims_, vec2(-1, -1));
    } else {
        vec4 col6 = applyVisualization(tech6, data6_, fragcoord);
        col = applyBlend(comp5, col, col6);
    }
}

FragData0 = col;
This fragment shader module contains functions to perform all necessary visualization techniques with the texture grammar shader (texgramshader.frag)

LIC algorithm

```
vec4 LIC(sampler2D tx, sampler2D flow, vec2 coord, vec2 vpdim, int kernel){
    vec2 tc = coord;
    vec4 res = texture(tx, clamp(tc/vpdim, 0.0, 1.0));
    vec2 tempv;
    int counter = 1;
    int k = kernel;
    for(int i=0; i<k; ++i){
        tc += (texture(flow, clamp(tc/vpdim, 0.0, 1.0)).xy * 2) - vec2(1,1);
        if( tc.x < 0 || tc.x > vpdim.x || tc.y < 0 || tc.y > vpdim.y ){ break; }
        res += texture(tx, clamp(tc/vpdim, 0.0, 1.0));
        ++counter;
    }
    tc = coord;
    for(int i=0; i<k; ++i){
        tc -= (texture(flow, clamp(tc/vpdim, 0.0, 1.0)).xy * 2) - vec2(1,1);
        if( tc.x < 0 || tc.x > vpdim.x || tc.y < 0 || tc.y > vpdim.y ){ break; }
        res += texture(tx, clamp(tc/vpdim, 0.0, 1.0));
        ++counter;
    }
    res /= counter;
    return res;
}
```

Isoline algorithm — very crude one, max 40 isovalue

```
vec4 isoline(sampler2D tx, vec2 coord, vec2 dim, float[40] ivals){
    float c = texture(tx, clamp(coord/dim, 0.0, 1.0)).x;
    vec4 colour = vec4(0.0);
    for(int i=0; i<ivals; i<1; ++i){
        if(ivals[i] == 0.0){ break; }
        float sn = c - ivals[i];
        float s1 = texture(tx, clamp((coord-vec2(1.0))/dim, 0.0, 1.0)).x - ivals[i];
        float s2 = texture(tx, clamp((coord-vec2(-1.0))/dim, 0.0, 1.0)).x - ivals[i];
        float s3 = texture(tx, clamp((coord-vec2(0.1))/dim, 0.0, 1.0)).x - ivals[i];
        float s4 = texture(tx, clamp((coord-vec2(0.0))/dim, 0.0, 1.0)).x - ivals[i];
        if( sn * s1 < 0.0 || sn * s2 < 0.0 || sn * s3 < 0.0 || sn * s4 < 0.0 {  
            colour = vec4(1.0); // isoline
            break;
        }
    }
    return colour;
}
```
// Arrowplot and streamlines – these are pre-rendered
// in the cpp processor, so just return the sampled value

vec4 arrowPlot(sampler2D img, vec2 coord, vec2 vdim)
{
    return texture(img, clamp(coord/vdim, 0.0, 1.0));
}

vec4 streamlines(sampler2D img, vec2 coord, vec2 vdim)
{
    return texture(img, clamp(coord/vdim, 0.0, 1.0));
}

// Colourmap

vec4 colourmap(const float m)
{
    return heatmap(m);
}

vec4 heatmap(const float m)
{
    const int colorTableSize = 4;
    vec3 colorTable[colorTableSize];
    colorTable[0] = vec3(0.0, 0.0, 0.0); // black
    colorTable[1] = vec3(1.0, 0.0, 0.0); // red
    colorTable[2] = vec3(1.0, 1.0, 0.0); // yellow
    colorTable[3] = vec3(1.0, 1.0, 1.0); // white

    float numColors = float(colorTableSize - 1);
    float v = clamp(m * numColors, 0.0, numColors);
    ivec2 limits = clamp(ivec2(int(v), int(ceil(v))), 0, colorTableSize);
    vec3 color = mix(colorTable[limits.x], colorTable[limits.y], fract(v));
    return vec4(color, 1.0);
Fragment shader source code

```glsl
// code/blendingtechniques.frag

// Simply returns a colour mix with weight w
vec4 blend(vec4 c1, vec4 c2, float w){
    return mix(c1, c2, w);
}

// Overlays c1 over c2 whereever c1 has an intensity > threshold
vec4 overlay(vec4 c1, vec4 c2, float threshold){
    float intensity = max(c1.r, max(c1.g, c1.b));
    vec4 result = c2;
    if(intensity > threshold){
        result = c1;
    }
    return result;
}

// Calculates a bumpmap with given offset at given coordinates for the
// provided sample field, then applies the bumpmap to the background pixel.
vec4 bumpmap(vec4 backgroundColor, sampler2D bumpmap, vec2 coord, vec2 vpdims, vec2 offset){
    vec4 grad = edge(bumpmap, coord, vpdims, offset) + vec4(1.0);
    return backgroundColor * grad;
}

vec4 edge(sampler2D img, vec2 coord, vec2 vpdims, vec2 offset){
    vec4 col0 = texture(img, clamp(coord/vpdims, 0.0, 1.0));
    vec4 col1 = texture(img, clamp((coord+offset)/vpdims, 0.0, 1.0));
    return col0 - col1;
}
```