THE APPLICATION OF COMPUTATIONAL MODELING TO DATA VISUALIZATION

BY

Daniel S. Pineo.

B.S. University of Massachusetts, Amherst, MA (2000)

DISSERTATION

Submitted to the University of New Hampshire
in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

in

Computer Science

December 2010

This dissertation has been examined and approved.

Dissertation Director, Colin Ware,

Professor of Computer Science

David H. Laidlaw,

Professor of Computer Science

R. Daniel Bergeron,

Professor of Computer Science

Wheeler Ruml,

Assistant Professor of Computer Science

Kurt Schwehr,

Research Assistant Professor of Ocean Engineering

Date

DEDICATION

To my wife Gretchen

ACKNOWLEDGEMENTS

I would like to foremost express my gratitude to my advisor, Colin Ware, for his guidance and support throughout my research. His invaluable insight focused me on what was important, and his imagination on what was possible. Many of the major ideas embodied in this work originated from discussions with him. This truly would not have been possible without him.

I would also like to thank my committee for their many valuable comments throughout the development of this work, and in particular David Laidlaw for providing the images used in his flow visualization evaluation paper.

Finally, I would like to thank my family. The encouragement, inspiration, and love of my wife, Gretchen, always gave me confidence and motivation when I needed it the most. I thank my parents, Carol and Craig, for their support and inspiration.

Funding for this work was provided by NOAA grant NA05NOS4001153.

TABLE OF CONTENTS

DEDICATION **iii**

ACKNOWLEDGEMENTS **iv**

LIST OF FIGURES **x**

ABSTRACT **xi**

1 INTRODUCTION **1**

1.1 Problem Statement 1

1.2 A Computational Approach 3

1.3 Original Contributions 8

2 BACKGROUND **10**

2.1 Perception in Data Visualization 10

2.1.1 Edge and Contour Perception 10

2.2 Computational Models of Visual Perception Applied to Data Visualization 13

2.3 The Neurophysiology of Perception 15

2.3.1 Neurons 15

2.3.2 The Eye and Retina 18

2.3.3 The Lateral Geniculate Nucleus (LGN) 19

2.3.4 V1 19

2.4 Neurophysiological Computational Models of Perception 21

2.4.1 Models of Retinal Center Surround Response 21
2.4.2 Models of V1 Edge Response 23
2.4.3 Models of V1 Edge Enhancement 23
2.4.4 Models of Color Perception 25
2.5 Conclusion 25

3 NEURAL MODELING OF FLOW RENDERING EFFECTIVENESS 27

3.1 Introduction 27
3.2 The Computational Model - CortexSim 1.0 28
3.2.1 The Edge Detection Stage 29
3.2.2 The Contour Enhancement Stage 29
3.2.3 The Streamline Tracing Stage 33
3.3 Qualitative Evaluation 36
Regular Arrows 38
Jittered Arrows 39
LIC 40
Aligned Streaklets 41
3.4 Evaluation 41
3.4.1 Participants 42
3.4.2 Vector Field Generation 42
3.4.3 Instructions 44
3.4.4 Procedure 44
3.4.5 Computer Trials 45
3.5 Results 45
3.6 Discussion and Conclusion 48

4 FLOW VISUALIZATION OPTIMIZATION VIA COMPUTATIONAL

MODELING OF PERCEPTION 51

4.1 Introduction 51

4.1.1 High-level Perception 53

4.2 The Computational Model - CortexSim 2.0 54

4.2.1 The Image 55

4.2.2 The Retina 56

4.2.3 V1 Edge Detection 57

4.2.4 V1 Edge Enhancement 59

4.2.5 GPU Model Implementation 60

4.3 Approach to Optimization 60

4.3.1 Graphical Primitives 61

Streaklet Primitives 61

Pixel Primitives 62

4.3.2 The Hill Climbing Algorithm 62

4.3.3 Evaluation Metric 64

4.3.4 Orientation perception 65

4.3.5 Speed Perception 66

4.4 Results 67

4.5 Discussion 71

5 COMPUTATIONAL MODELING OF NODE-LINK GRAPH PERCEPTION 75

5.1 Introduction 75

5.2 Background 76

5.2.1 Layered and Two-layer Node-link Graph Diagrams 77

5.2.2 Perceptual Theory and Node-link Diagram Aesthetics 79

5.2.3 Roelfsema's Curve Tracing Experiment 79

5.2.4 Roelfsema's Curve Tracing Model 80

5.3 The Computational Model - CortexSim 3.0 81

5.3.1 The Label Cortex 81

5.3.2 Grossberg's Shunting Neural Dynamics 84

5.4 Model Evaluation 87

5.4.1 Node-link Graph Diagram Generation 87

5.4.2 Graph Data Generation 88

5.4.3 Instructions 89

5.4.4 Procedure 89

5.4.5 Participants 91

5.4.6 Computer Trials 91

5.4.7 Node-link Graph Modeling Results 91

5.5 Node-link Graph Diagram Optimization 93

5.5.1 Optimization Evaluation 95

5.6 Discussion 95

6 CONCLUSION 98

BIBLIOGRAPHY 106

APPENDICES 115

A CORTEXSIM 2.0 GPU KERNELS 116

B CORTEXSIM 3.0 GPU KERNELS 121

LIST OF FIGURES

1-1 The Three Pillars of Modern Science 4

1-2 Scramjet CFD Model 5

1-3 Andromeda CFD Model 6

2-1 The Perception of Contours 11

2-2 The Perceptual Enhancement and Inhibition of Edges 12

2-3 The Early Human Visual System 16

2-4 A Neuron 17

2-5 V1 visual processing 20

2-6 The Difference-of-Gaussians Receptive Field 22

2-7 The Gabor Response Function 24

3-1 Zhaoping's Model of V1. 30

3-2 Regular Arrows 38

3-3 Jittered Arrows 39

3-4 Line Integral Convolution 40

3-5 Aligned Streaklets 42

3-6 Jittered Arrows Closeup 43

3-7 Aligned Arrows Closeup 44

3-8 Human Error Distribution 46

3-9 Model Error Distribution 47

3-10 Log Transformed Histogram 47

3-11 Visualization Mean Error 48

4-1 The Neural Network Architecture 56
4-2 V1 Gabor Kernels 58
4-3 V1 Edge Enhancement Kernels 59
4-4 Optimization Block Diagram 63
4-5 Jobard & Lefer Visualization 68
4-6 Optimized Flow Visualization 68
4-7 Optimized Jet Stream Visualization 69
4-8 Optimized Color Jet Stream Visualization 69
4-9 Pixel Parameterized Visualization 70
4-10 Line Integral Convolution Visualization 70
4-11 Closeup of the discontinuities 71

5-1 Node-link Diagram Example 77
5-2 Layered Node-link Diagram Example 78
5-3 Roelfsema's Experiment 80
5-4 Roelfsema's Contour Tracing Operator 82
5-5 Labeling in CortexSim 83
5-6 Labeling Spread with Competitive Dynamics 86
5-7 Two-layer Node-link Graph Diagram Example 88
5-8 A Hermite spline 89
5-9 The Link Tracing Experiment 90
5-10 Model Iterations vs. Human Response Time 92
5-11 Model Error vs. Human Error 93
5-12 Sequence of Graph Optimizations 94
5-13 Original vs. Optimized Response Time 96

6-1 The General Design of the CortexSim Models 99

ABSTRACT

THE APPLICATION OF COMPUTATIONAL MODELING TO DATA VISUALIZATION

by

Daniel S. Pineo.
University of New Hampshire, December, 2010

Researchers have argued that perceptual issues are important in determining what makes an effective visualization, but generally only provide descriptive guidelines for transforming perceptual theory into practical designs. In order to bridge the gap between theory and practice in a more rigorous way, a computational model of the primary visual cortex is used to explore the perception of data visualizations.

A method is presented for automatically evaluating and optimizing data visualizations for an analytical task using a computational model of human vision. The method relies on a neural network simulation of early perceptual processing in the retina and visual cortex. The neural activity resulting from viewing an information visualization is simulated and evaluated to produce metrics of visualization effectiveness for analytical tasks. Visualization optimization is achieved by applying these effectiveness metrics as the utility function in a hill-climbing algorithm. This method is applied to the evaluation and optimization of two visualization types: 2D flow visualizations and node-link graph visualizations.

The computational perceptual model is applied to various visual representations of flow fields evaluated using the advection task of Laidlaw et al. [1]. The predictive power of the model is examined by comparing its performance to that of human subjects on the advection task using four flow visualization types. The results show the same overall pattern for humans and the model. In both cases, the best performance was obtained from visualizations containing aligned visual edges. Flow visualization optimization is done using both streaklet-based and pixel-based visualization parameterizations. An emergent property of the streaklet-based optimization is head-to-tail streaklet alignment, the pixel-based parameterization results in a LIC-like result.

The model is also applied to node-link graph diagram visualizations for a node connectivity task using two-layer node-link diagrams. The model evaluation of node-link graph visualizations correlates with human performance, in terms of both accuracy and response time. Node-link graph visualizations are optimized using the perceptual model. The optimized node-link diagrams exhibit the aesthetic properties associated with good node-link diagram design, such as straight edges, minimal edge crossings, and maximimal crossing angles, and yields empirically better performance on the node connectivity task.

CHAPTER 1

INTRODUCTION

Perception is the means through which all data visualizations are processed, and the characteristics of successful data visualizations can often be traced back to perceptual mechanisms of the human visual system. It is becoming increasingly recognized that the properties of human perception play a vital role in determining the effectiveness of data visualizations [2]. For example, the most successful flow visualizations contain contours tangential to the flow field. These visualizations take advantage of the edge detection mechanisms of human vision, which respond strongly to continuous contours. As a second example, visualizations that must convey fine spatial details are found to be more effective when they are rendered in greyscale then in a red-green scale or yellow-blue scale. This is because luminance contrast processing mechanisms of the visual system can convey far more detail than the chromatic contrast processing mechanisms.

1.1 Problem Statement

Despite the obvious relevance, perceptual theories have been of limited use to data visualization designers because they are typically expressed in imprecise and descriptive forms that are not readily applicable. For example, the theory of contour perception states that two nearby visual objects that exhibit edges oriented close to the collinear axis between them will be perceived as a continuous contour [3]. While this theory suggests that we should limit our flow visualization designs to those with aligned arrows, it still leaves and huge number of degrees of freedom unspecified. What about the myriad of other design choices? A data visualization designer would likely be interested in understanding the perceptual effects of the lengths of the arrows, the widths, their spacings, and the presence or lack of arrowheads. Certainly these all have some effect on the perceptual experience, but what is that effect? The space of possible visualization choices is enormous, and descriptive theories are not comprehensive enough to describe it. As a result, creating effective data visualizations continues to be more of an art than a science.

Seminal flow visualization papers that introduced the most widely used flow visualization methods, such those of Cabral and Leedom [4], Jobard and Lefer[5], Turk and Banks [6], and van Wijk [7], give no indication of using perceptual theory in their development. Thus, there is clearly a gap between perceptual theory and its application to data visualization, which is likely due to the difficulty in applying such a descriptive theory in practice. In particular, computers are not capable of interpreting a descriptive theory, thus limiting their applicability. Interpretations by humans are often imprecise, inconsistent, and subjective.

Furthermore, these seminal flow papers lack psychophysical studies into the effectiveness of the visualization methods they introduce, although more recently Laidlaw et al. [1] conducted a study that showed large differences in the effectiveness of different methods for a variety of visual tasks. Psychophysics complements perceptual theory, allowing designers to experimentally quantify the effectiveness of data visualizations techniques by testing the performance of human test subjects on tasks requiring accurate interpretation of visualizations, but are often costly and time-consuming. Laidlaw et al. only examined a tiny subset of the possible flow renderings, when all the parametric variations are considered. The combinatoric problem means that human testing is not a viable technique for evaluating the space of design alternatives for any reasonably complex visualization.

1.2 A Computational Approach

An alternative method of quantifying the effectiveness of a data visualization, one that addresses the shortcomings of descriptive theories and psychophysical tests, is highly desirable. What is needed is not just a perceptual *theory*, but a perceptual *computational model* that allows data visualization designers to explore in detail how the perceptual mechanisms of the human visual system respond to a data visualization.

Computational modeling has now emerged as the third pillar of modern science (Figure 1-1), a peer alongside theory and physical experiment [8]. The basis for this claim is the enormous impact that computational modeling has had in the scientific disciplines where it has been successfully applied. Computational modeling has a history of tying together theory and experiment by showing how small scale mechanics produce large scale emergent behavior.

For example, computational modeling has become indispensable addition to the field of fluid dynamics. While closed form theories of fluid dynamics are capable of describing idealized systems, in even simple real world systems the behavior can quickly become too complex for them to be directly applied in practice. For example, the problem of designing of supersonic blunt nose aircraft confounded the fluid dynamics community throughout the 1950's and early 1960's [9]. The complexity of the mathematical analysis of this seemingly simple problem proved to be essentially intractable. Experimental approaches to fluid dynamics via wind tunnels have the limitations of being slow and expensive, and only yield a few variables such as lift and drag. These limitations have a striking resemblance to the current limitations of perceptual theory and psychophysical testing. In both cases, the theory is difficult

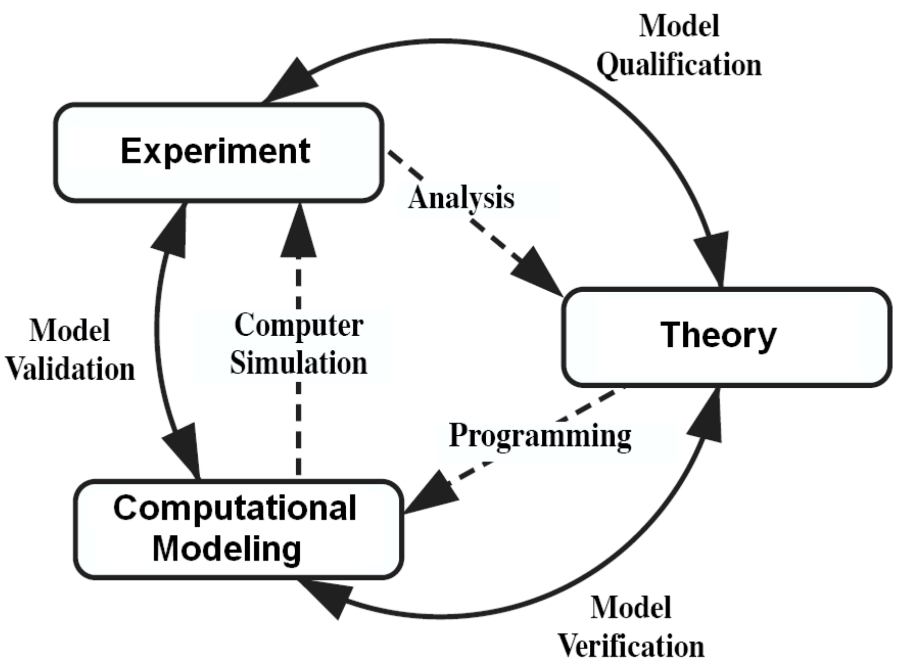
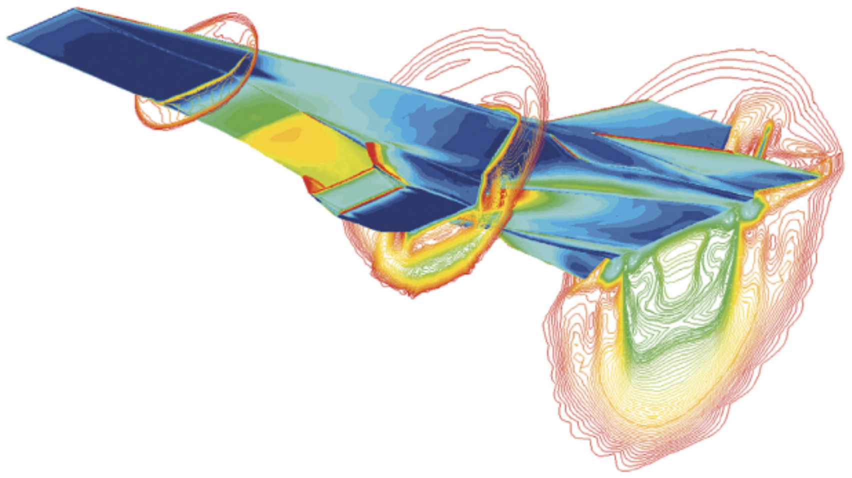


Figure 1-1: *The three pillars of modern science: experiment, theory, and computational modeling*

to apply, with the experimental approach being slow, costly, and providing only a partial understanding of the system.

For the field of fluid dynamics, the solution to these problems arrived with the application of computational models. These computational fluid dynamics (CFD) models calculate the behavior of small units of volume each behaving according to the Naviar-Stokes equation (Figure 1-2), which describe the pressures and velocities of the fluid in the volume. The generalization afforded by this approach allows the model to be reused for arbitrary designs, there is no longer the need to create a new model for each design. The models allow designers to easily determine the pressures and velocities at any point. Thus, designers are able to inexpensively, interactively, and with great detail, explore the behavior of the system. Ultimately, the application of computational modeling profoundly changed the field of fluid dynamics.



$$\rho(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \nabla \cdot \mathbb{T} + \mathbf{f}$$

Figure 1-2: *The CFD model of the Hyper-X Scramjet that enabled NASA engineers to study its behavior while operating at Mach 7, the Naviar-Stokes equation on which the model is based [10].*

Another example of the benefits of applying computational science can be found in cosmology. While the gravitational mechanics underlying the interaction of pairs of stars was well known, how these interactions produced the spirals and arms found in galaxies was an unsettled question. The number of stars involved was too large to be calculated analytically, and experimentation was clearly not possible. Finally, with the application of computational science, cosmologists were able to model galaxy behavior (Figure 1-3) and settle the question once and for all.

In many ways, data visualization designers find themselves in the same predicament. The complexity of real-world data visualizations interacting with the complexities of the human perceptual process makes the direct application of perceptual theories difficult. As with wind tunnels, psychophysical experiments allow the study of complex data visualization designs experimentally, but this approach is slow and

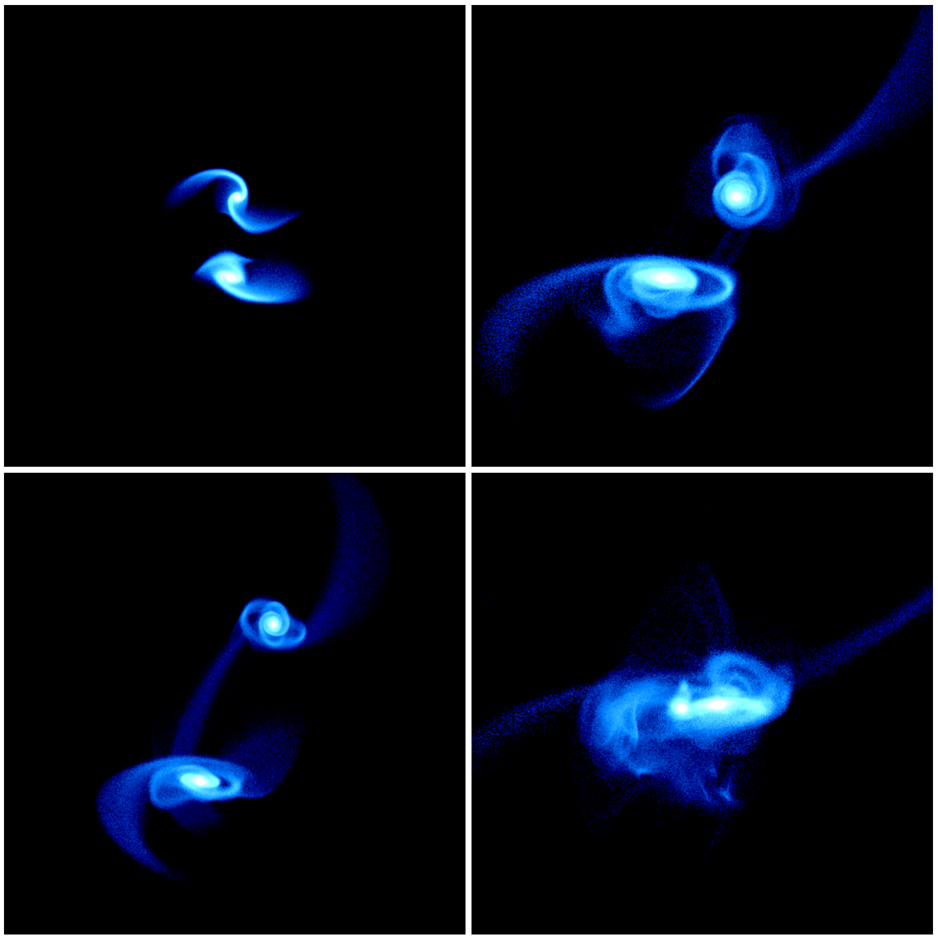


Figure 1-3: *An N-body simulation of the Milky Way and Andromeda galaxy collision that will take place in 3 billion years [11].*

expensive, and the data obtained for one design is not always generalizable to others. What data visualization designers lack is a computational model of perception analogous to the CFD models that have benefited aeronautical designers. Such a model would not only provide a cheaper alternative to human psychophysical studies, but yield more detailed insight of the visual system's response to those visualizations.

Computational models describe a system by calculating the behavior of a large number of simple, interacting elements. The complex behavior of the system emerges from the interaction of the simple elements. CFD models, for example, show how complex behaviors such as waves, vorticies, and turbulence emerge from volume elements (voxels) behaving according to the Naviar-Stokes formulas that describe fluid flow. For a computational neural model of visual perception, the simple elements are the neurons of the human visual system, and the emergent behaviors are the perceptual mechanisms. There are many potential advantages to applying a computational modeling approach to data visualization:

- Simplicity - Computational models express complex behavior in terms of a few fundamental behavioral rules, enabling the behavior of the system to be more easily comprehended. By applying a computational model of visual perception to data visualization, a visualization practitioner could observe the effects that a visualization design have on the neural dynamics of the visual system, thus gaining insight into the reason for its effectiveness.
- Utility - Computational models produce numerical values that are suitable for use within a larger system. We explore this possibility in this thesis work by encompassing a computational model of perception within an optimization loop.
- Visibility - The numerical values with which computational models are expressed afford the creation of visualizations that show the effects that the input design has on the model.
- Extensibility - Computational models can be easily integrated with other computational models. For example, a computational model of the early visual system might be integrated with a computational model of working memory, allowing designers to explore the cognitive load associated with a visualization.
- Generality - While descriptive models often describe very specific phenomena, a computational model of perception may be made to process arbitrary visual image inputs.

- Validation - The neural and psychophysical behavior predicted by computational models can be compared with reality.

- Verification - A computational model provides a test of theoretical models. If the computational model yields the behavior predicted by the theoretical model, then this provides evidence that both the computational and the theoretical models are correct.

There are good reasons to believe that the ingredients are present for computational modeling to make a similar contribution to the science of data visualization.

- The primary visual cortex (also known as area V1) of the brain contains a very large number of computational elements all performing the same task in parallel on different parts of an image. This means that high performance parallel processing can be applied.

- V1 has been extensively studied by vision researchers for several decades and its mechanisms are quite well understood.

- V1 is a critical area for pattern perception. It is here that the elements of form are processed. It is also here that the computations are made regarding which parts of an image will be visually salient. Although our understanding of the brain is too limited and computers are still insufficiently powerful to simulate the entire human brain we believe that useful progress can be made by simulating this critical brain area for visual processing.

1.3 Original Contributions

The contribution of this work is the application of the computational modeling paradigm to the problem of data visualization, and the demonstration that such an approach can yield tangible results. Given the tremendous impact that computational science has had on other scientific disciplines, it seems clear that exploring the viability of its application to data visualization is a worthwhile endeavor.

My contribution has five components:

1. Development of a computational model of human perception that simulates the visual mechanisms critical to the perception of data visualizations.

2. Comparison of the model evaluations with human performance on psychophysical tasks

3. Automatic optimization of data visualizations based on the effectiveness predicted by the model.

4. Demonstration of the generality of the approach by the application to two distinct classes of visualizations: 2D vector fields and node-link diagrams.

5. Establishment of computational modeling of perception as a "third pillar of science" in data visualization by developing its relationship with perceptual theory and psychophysical experimentation.

CHAPTER 2

BACKGROUND

The approach of using a perceptual model to optimize and evaluate data visualizations as described in chapter 1 touches upon a diverse range of scientific disciplines, including data visualization, perception, neurophysiology, and computational modeling. In this chapter, we review the relevant work within these fields as they pertain to this approach. In particular, we develop the relationship between these disciplines and how they relate to the end problem of data visualization.

2.1 Perception in Data Visualization

Perceptual theory is perhaps the most directly related to the field of data visualization, describing the visual mechanisms triggered when one views a data visualization. Such descriptions are useful for understanding the implications that the choice of visual variables has on the perception of a data visualization. Perceptual theory has been applied to a number of data visualization design problems, such as color sequences for color maps [12], node-link diagram layout [13] [14], symbol color and shape distinctiveness[15] , and texture coding for scalar maps [16].

2.1.1 Edge and Contour Perception

Of particular relevance to this thesis work is the perception of edges and contours in data visualizations. The flow visualizations and node-link graph diagrams that we model in this thesis work rely heavily on the use of edges and contours to convey

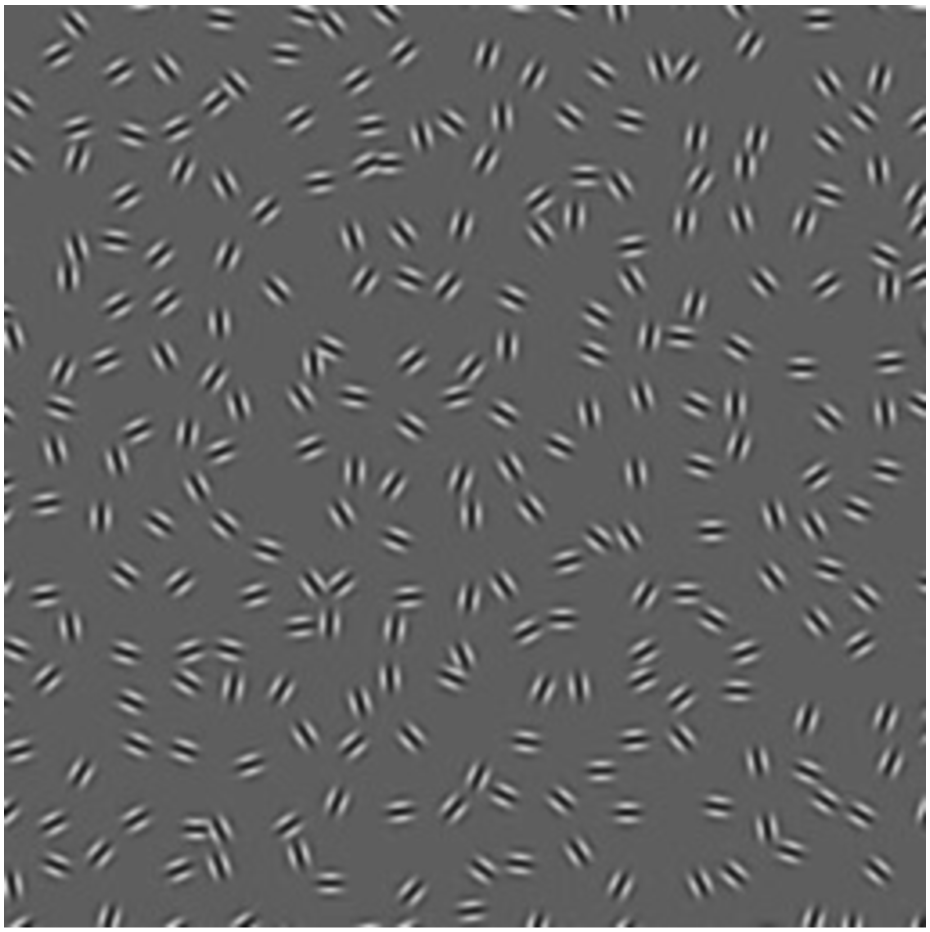


Figure 2-1: *The perception of contours (from Field et al. [3]). The contour composed of aligned edges is easily perceived among a field of distractors.*

information. Perceptual processing mechanisms enable a viewer to rapidly detect contours. Psychophysical research by Field et al. [3] has suggested that test subjects were able to identify a path of edge elements among a field of distractor elements (Figure 2-1). They found this to be reliable even as the elements were oriented up to $60°$ relative to each other. This suggests that there exists a perceptual mechanism in the visual system that is specialized in detecting continuous contours, and has given rise to the perceptual theory that two nearby visual objects that exhibit edges oriented close to the colinear axis between them will be perceived as a continuous contour.

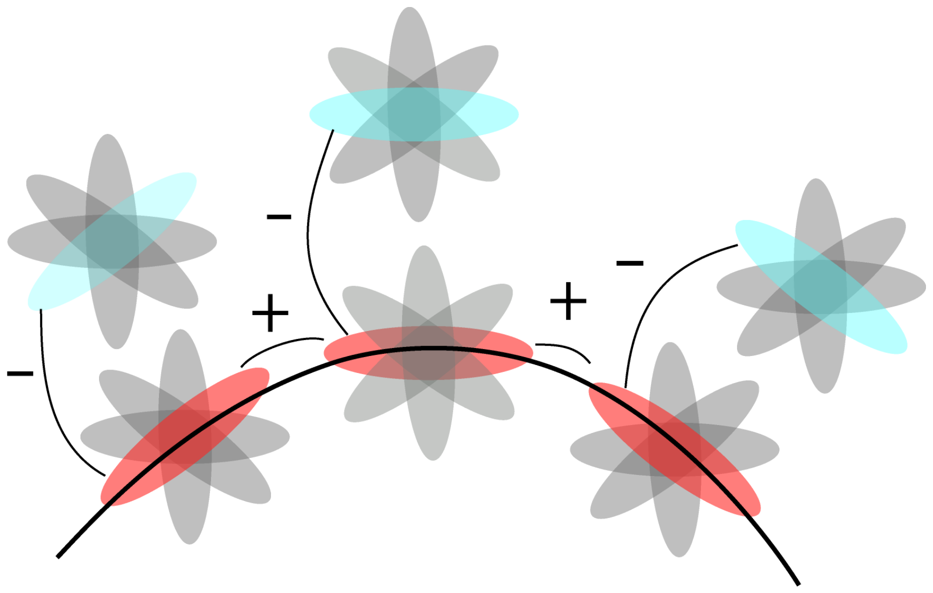


Figure 2-2: *The perceptual enhancement and inhibition of edges. The perception of nearby aligned edges are enhanced, while the perception of unaligned edges is inhibited. This mechanism enables continuous contours to be easily perceived.*

This research on edge perception, along with neurophysiological evidence described later in section 2.3.4, suggests that there is some manner of lateral coupling among the visual elements involved in detecting the path of edge elements. They and other researchers have suggested that similarly oriented aligned contours mutually excite one another, whereas they inhibit other neurons that are nearby (Figure 2-2).

Ware [17] proposed that the effectiveness of flow visualizations can be linked to these perceptual processing mechanisms. Since conveying the path of flow advection is one of the primary goals of flow visualization, the most effective flow visualizations use contours to visually express the vector field. In effective flow visualization techniques, contours within the visualizations are evident and are tangential to the direction of flow. Laidlaw et al. [1] showed how to test the effectiveness of vector visualizations by tasking human subjects with estimating where the particle advecting within the flow field exits a bounding circle. They found that visualizations containing strong perceptual contours along the flow direction tended to perform best.

The effectiveness of node-link graph diagrams can also be understood in terms of this contour perception mechanism. Node-link graph diagrams use visual contours connecting nodes to convey information. Smooth contours, lacking sharp angles or crossings, are perceived most readily by the visual system, and are thus most effective. This behavior is applied in data visualization in the form of graph layout aesthetics [18] [19], which provide guidelines for effective layout of node-link graph diagrams.

It is important to note, however, that perceptual theories of the kind discussed above are descriptive, rather than computational. As a result, they are capable of only providing general guidelines for data visualization design. Such descriptive theories cannot be used to quantitatively evaluate data visualizations.

2.2 Computational Models of Visual Perception Applied to Data Visualization

In contrast to descriptive perceptual theories, computational models of perception enable the analytical evaluation of data visualizations. A significant advantage of such computational models is that they can be applied to a visual image using computer algorithms, unlike descriptive theories that require human interpretation. These models have been used to automatically evaluate the perceptual properties of visual popout, saliency, and clutter in data visualizations.

Visual popout is the mechanism of visual perception whereby conspicuous features in a visual scene can be easily found. Theories of popout suggest that visual features are processed in parallel to create a set of "feature maps" [20] [21]. Each feature map expresses the presence of some visual variable, such as size, orientation, or color, at all points in the visual space. Based on these theories, computational models of visual popout have been developed to predict the saliency of visual objects within a scene [22], and such models have been applied to data visualization to predict how a viewer's attention will be directed to relevant areas of a chart visualization [23] [24] [25].

Clutter is a concept closely related to visual popout, but describes factors that make targets harder, rather than easier, to see. Rosenholtz et al. [26] defined clutter as "the state in which excess items, or their representation or organization, lead to a degradation of performance at some task", and offered two computational methods for measuring clutter.

The first method treats clutter from a purely information theoretic standpoint, by relating clutter to the number of bits required to encode the scene. While this method provides an elegant measure of clutter, it is not clear how this method relates to perceptual processes involved in the perception of clutter. The second method, which is based on a statistical measure of saliency, is more closely related to perceptual processing. This method calculates the ensemble of feature vectors in a scene. These feature vectors correspond to the local features detected in the early visual system, such as edges and color. The measure of clutter is derived from the volume of the feature space spanned by the ensemble. Both measures correlate well with human response time on search tasks using data visualizations, with no significant difference between the measures.

While these models seek to computationally express perceptual behavior when viewing data visualizations, they do not attempt to do so in a manner that is physiologically plausible. These models describe very specific perceptual behavior and are not straightforwardly extended to include additional behaviors. Nevertheless, this work represents the state of the art in applying models of perception to the problem of data visualization. Computational models based on the neurophysiology of the visual system have the potential to express not just one specific perceptual behavior, but possibly many behaviors that may be relevant to the perception of a data visualization.

2.3 The Neurophysiology of Perception

Having argued the benefit of a computational neural model of perception for data visualization, and noting its potential for application in data visualization, we now review the architecture of the human visual system and explore the feasibility of constructing a computational model that is capable of expressing the visual mechanisms observed in the perception of data visualizations.

The early stages of the visual system are critical to the perceptual processing of data visualizations. When a visual image projects onto the retina at the back of the eye, receptors in the eye convert the image into electrical signals. These signals pass through the midbrain lateral geniculate nucleus (LGN), before proceeding to the primary visual cortex (V1) at the posterior of the brain (Figure 2-3). These early stages of visual processing provide the foundation of pattern perception, color perception, texture and visual salience. Understanding how a visualization is processed in these stages can tell us a lot about how effective it will be. In the following brief review of relevant neurophysiology, we concentrate on aspects of the visual system that are relevant to the modeling effort. Specific neurophysiological measurements are given so they may be later incorporated into the computational model.

2.3.1 Neurons

The most basic building block of the visual system is the neuron. Neurons contain many branches, called *dendrites*, that allow them to receive signals from other neurons (Figure 2-4). When a particular pattern of signals is received, the neuron may become excited and begin signaling other neurons along a branch called an *axon*. Neurons

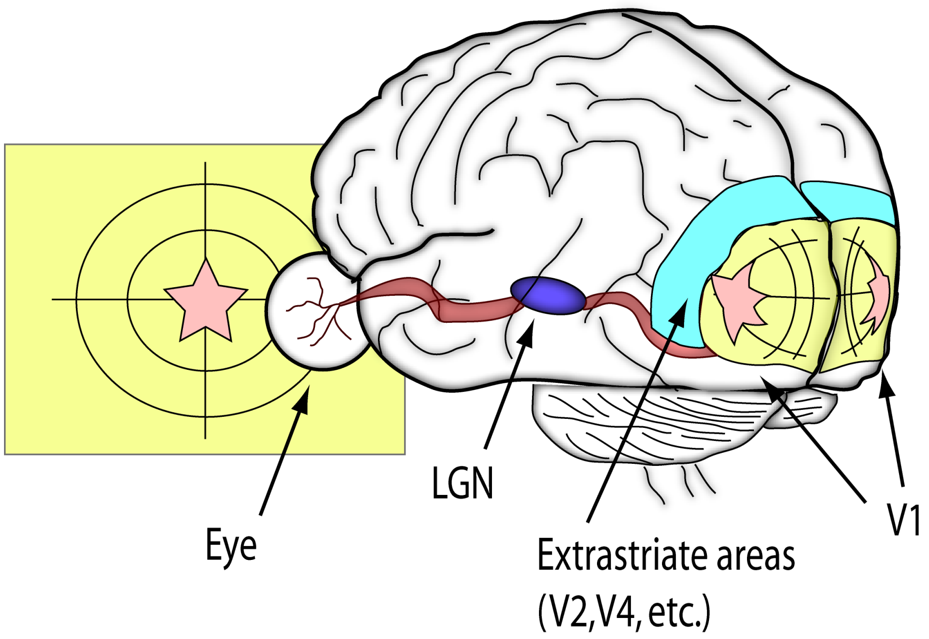


Figure 2-3: *The early stages of human vision. Light enters the eye and falls on the retina. Photoreceptors in the eye then send signals along the optic nerve, through the LGN, and finally to V1. The mapping of locations in the visual scene to neurons in V1 is depicted.*

make on the order of 10,000 connections, called *synapses*, to other neurons. Depending on the neurotransmitter used by the neuron, its signals may have either excitatory or inhibitory effects on the recipient neuron. Signals from an excitatory neuron makes the neurons it signals more likely to become excited, and signals from inhibitory neurons make the neurons it signals less likely to become excited.

The current across the neural membrane was first expressed by Hodgkin and Huxley [28] as:

$$I = C_M \frac{dV}{dt} + \bar{g}_K n^4 (V - V_K) + \bar{g}_{Na} m^3 h (V - V_{Na}) + \bar{g}_l (V - V_l)$$

where I is the total current through the surface membrane of a neuron. It is dependent on both the conductivity of the synaptic connection and the activity of the source

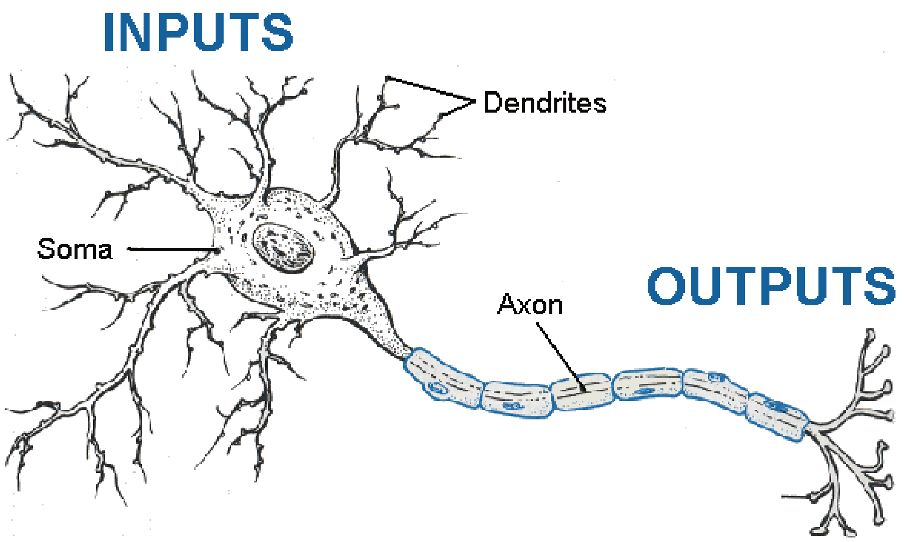


Figure 2-4: *An example of a neuron [27]. The neuron receives input signals along its dendrites, and sends an output signal along its axon.*

neuron. In this equation, $C_M \frac{dV}{dt}$ is the capacitive current, $\bar{g}_K n^4 (V - V_K)$ is an ionic current of Potassium ions, $\bar{g}_{Na} m^3 h (V - V_{Na})$ is an ionic current of Sodium ions, and $\bar{g}_l (V - V_l)$ is a leakage current.

In artificial neural network models, the membrane conductances are often combined and simplified into a single weight parameter, w_{ij}. This parameter specifies the ability of a signal to propagate from the presynaptic neuron i, to the postsynaptic neuron j. The overall activity of the postsynaptic neuron, y_i, is then calculated with the function [29]:

$$y_j = \phi \left(\sum_i w_{ij} x_i \right)$$

Where, x_i is the activity of the presynaptic neuron, and ϕ is a sigmoid saturation function. Thus, w_{ij} describes the pattern that the neuron responds to. In the early visual system, these are localized visual patterns such as edges. The local visual area corresponding to the pattern is known as a receptive field.

2.3.2 The Eye and Retina

The visual image is projected by the lens of the eye and focused onto the retinal lining at the back of the eye. The light travels through the layers of the retina and falls upon the light-sensitive photoreceptor neurons. There exist approximately 6 million color-sensitive photoreceptors, known as cones, and approximately 100 million color-insensitive photoreceptors, known as rods. Cones are subtended by an angle of approximately .006 degrees and are the principle photoreceptors responsible for processing high-detail at normal light levels. Cones are most sensitive to a specific frequency of light, such as red, green, or blue, but all cones are capable of detecting white light. Thus, the retina is capable of processing black and white patterns at a higher level of detail than it can process blue-yellow or red-green patterns.

The signals produced by these photoreceptors are then processed by several layers of neurons in the retina. The receptive field surround signal is computed by horizontal [30] and amacrine cells that are connected laterally to nearby areas of the retina. Bipolar cells combine this surround signal with a center signal of an opposing polarity to produce what is known as a *center-surround receptive field*. Such a receptive field responds most strongly to light falling either in center of the field, or in the annulus surrounding center, but not both. Bipolar cells exist in both on-sensitive and off-sensitive forms, thus producing receptive fields that are sensitive to light in the center (on-center/off-surround) and as well as fields that are sensitive to light in the annulus (off-center/on-surround). Bipolar cells forward this signal to the retinal ganglion cells, which send the center surround signal along the optic nerve, through the lateral geniculate nucleus (LGN), and to the V1.

The central 2° of vision is processed by a $500\mu m$ diameter rod-free area of the retina known as the *fovea* [31]. It is this area that is responsible for processing fine detail. On average, there are approximately two retinal ganglion cells for each cone in this area, one on-center and one off-center [32]. Away from the fovea, the cone spacing and the ratio of photoreceptors to retinal ganglion cells increases quickly, causing a drop in visual acuity in the periphery.

2.3.3 The Lateral Geniculate Nucleus (LGN)

The LGN is an area where the optic nerves from the two eyes connect in the mid-brain. It appears that one of the primary functions of the LGN is to physically reorganize the signal paths. For example, signals from the two eyes are brought into close proximity. Also, the signals from the rods and cones are separated so they may progress down different processing paths. The signal processing performed by LGN remains incompletely understood, but is believed to be involved in stereopsis. Because this functionality is not pertinent to this work, the LGN is not included in the computational model.

2.3.4 V1

V1, also known as the primary visual cortex, is the first cortical processing area of vision, and is largely populated with neurons that are selective to orientated edges. These neurons are capable of detecting the edges of sinusoidal grating patterns with spatial frequencies up to 8 cycles per degree between 2° and 5° eccentricity [33], and presumably higher in the fovea. The receptive fields of V1 neurons are retinotopic, meaning that there is a direct mapping between a neuron's location in V1 and it's receptive field in visual space.

The architecture of V1 can best be understood in terms of columns and hyper-columns. A column is a cortical area of neurons that respond to a specific edge orientation and receptive field location [34] [35] and contains about 200 to 250 neurons [36]. These columns form a repeating pattern across the surface of V1, such that

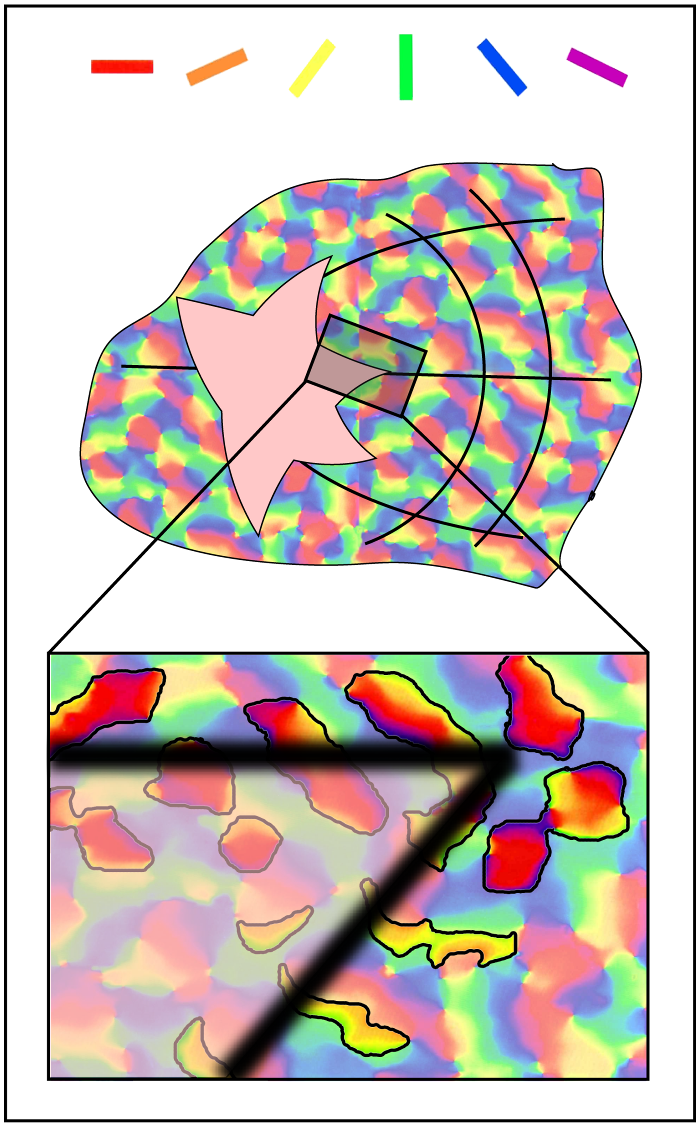


Figure 2-5: *The V1 visual processing of the scene from Figure 2-3. Red patches are cortical columns containing neurons that respond to horizontal edges, yellow patches are columns sensitive to edges angled at* $60°$*. Highlighted areas indicate columns that are active due to the edges of the star viewed in the visual scene.*

within any $500\mu m$ diameter area columns *all orientations are represented* (Figure 2-5). Such an area is known as a hypercolumn. Both columns and hypercolumns lack well-defined borders in the visual cortex, however, their treatment as discrete, well-defined areas provides a useful approximation for understanding and modeling cortical behavior.

Furthermore, there is evidence that the perception of an edge is enhanced by nearby edges with an aligned orientation [3]. Studies of the dendritic trees of V1 neurons suggest that this behavior is achieved via lateral connections, up to 4mm long, between neighboring columns in V1 [37]. V1 is notable for detecting visual feature dimensions including color, orientation, and simple motion, which form the basic primitives that are used in all subsequent processing. Higher level visual areas, such as V2 and V4, are currently not as well understood, and are not modeled in this work.

2.4 Neurophysiological Computational Models of Perception

The neurophysiology described in the previous section is responsible for the perceptual mechanisms activated when viewing data visualizations. By expressing this physiology in a computational form, we can construct a model that exhibits the behavior of the visual system, capable of being used to analyze data visualizations. In this section, we review mathematical models that have been used to describe aspects of the visual system.

2.4.1 Models of Retinal Center Surround Response

The center surround response of retinal ganglion cells can be described mathematically using a Difference-of-Gaussians (DoG) function (Equation 2.1, Figure 2-6) [38]. This function contains a narrow excitatory center, produced by the cones of the retina and forwarded directly to the retinal ganglion cells by the bipolar cells. The excitatory center is encompassed by a larger inhibitory surround, produced by the inhibitory effect of nearby horizontal and amacrine cells.

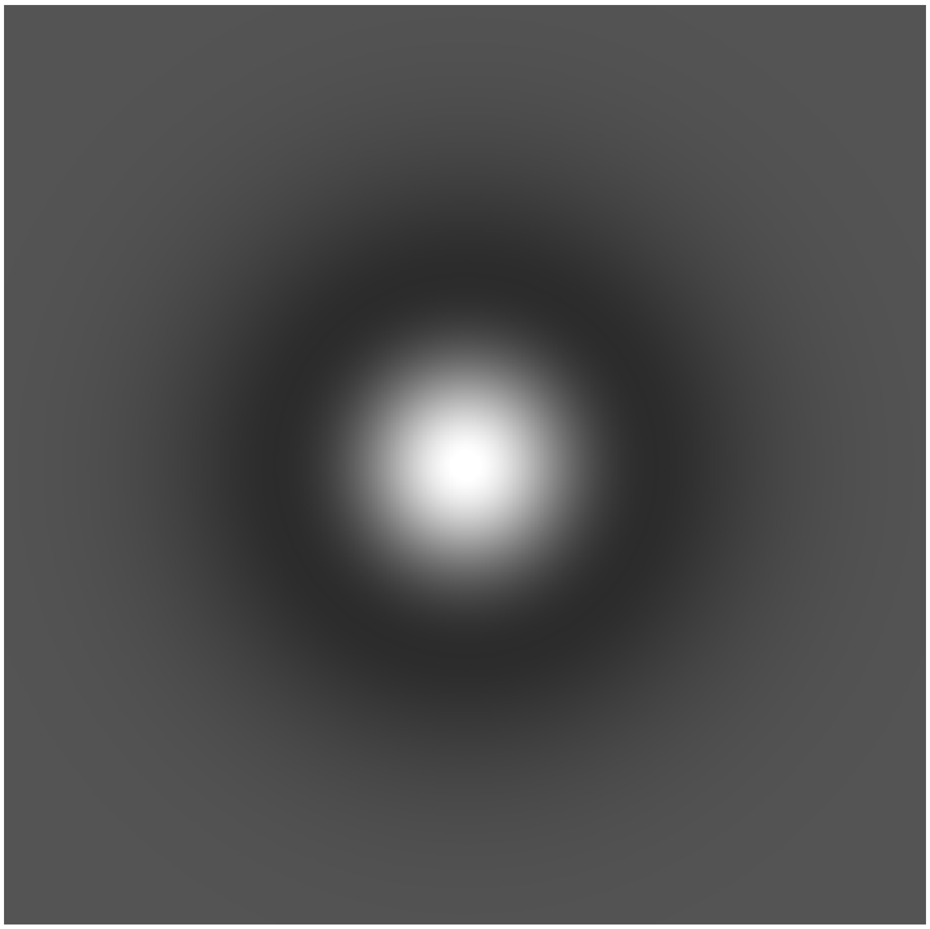


Figure 2-6: *The retinal ganglion Difference-of-Gaussians receptive field, containing an excitatory on-center (light) and an inhibitory off-surround (dark).*

$$DoG_{x,y,\sigma_1,\sigma_2} = \alpha_1 G_{x,y,\sigma_1} - \alpha_2 G_{x,y,\sigma_2} \tag{2.1}$$

Where $G_{x,y,\sigma}$ denotes the normalized two-dimensional Gaussian kernel,

$$G_{x,y,\sigma} = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.2}$$

and σ denotes the standard deviation.

2.4.2 Models of V1 Edge Response

As noted in the previous section, the neurons of area V1 have been observed to respond preferentially to orientated edges [34] [35]. It has been argued that the Gabor function (Figure 2-7, Equation 2.3), defined mathematically as one-dimensional sinusoid encapsulated within a two-dimensional Gaussian envelope (Equation 2.2), is a good approximation for the edge patterns for which V1 neurons are selective [39]. Daugman argued that this function provides an optimal tradeoff between representing spatial position and frequency.

$$Gabor_{x,y,\lambda,\sigma,\phi} = G_{x,y,\sigma} \cos(\frac{2\pi}{\lambda}x + \phi) \tag{2.3}$$

where λ defines the wavelength of the sinusoid, and ϕ defines the sinusoid offset.

2.4.3 Models of V1 Edge Enhancement

The edge enhancement behavior described in section 2.1.1 can be achieved by implementing the pattern of lateral excitation described in section 2.3.4. Such a model of V1 edge enhancement was developed by Zhaoping Li [40]. In this model, excitatory and inhibitory neurons exist in pairs, and together express the saliency of an edge. Li's model focuses entirely on the contour enhancement mechanisms of V1. It does not include retinal processing, or even edge pattern recognition using a Gabor-like pattern matching elements. Thus, the model is not capable of processing an input image directly, but instead requires an array of edge perception strengths as input. The model uses a hexagonal array to represent the hypercolumn structure of V1, with each hexagon containing a set of 12 orientation selective columns. This model is described in more detail in chapter 3, where it is used as the basis for our model of flow visualization perception.

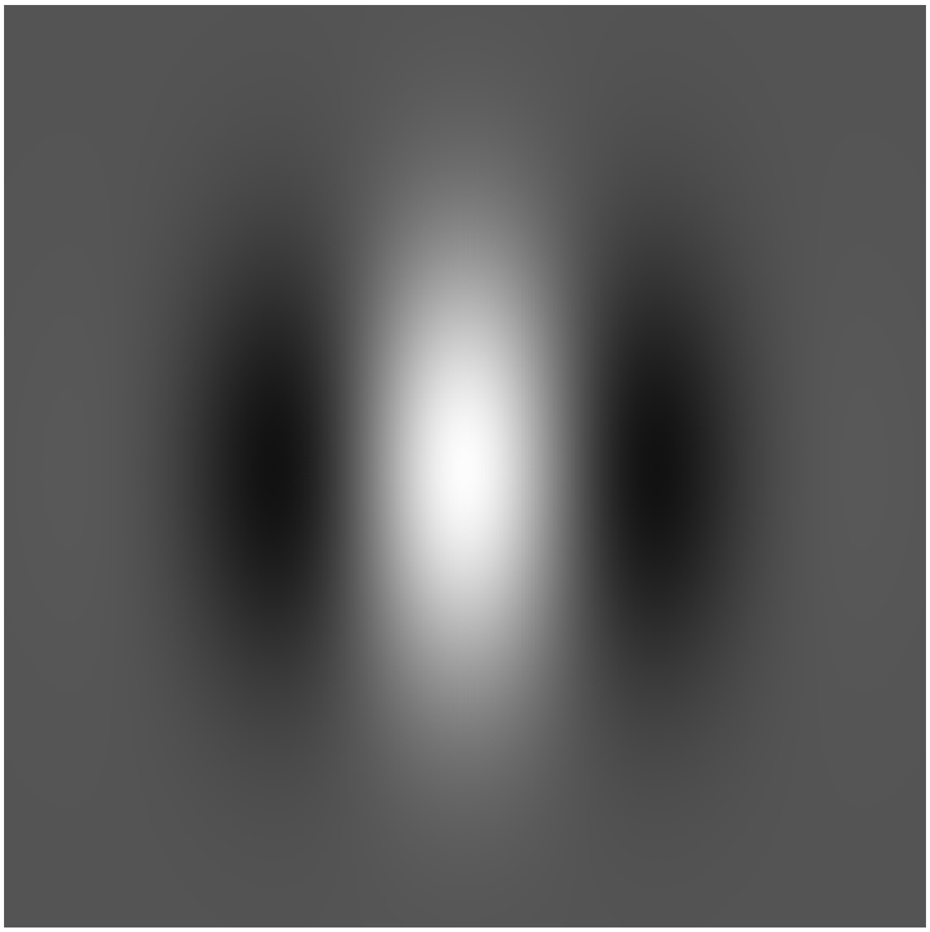


Figure 2-7: *The Gabor response function, also with an excitatory center and inhibitory surround.*

The LAMINART model is a more detailed and physiologically-based model of V1 edge perception developed by Grossberg et al. [41] [42]. It is considerably more detailed than Li's model, including the center-surround processing of the retina and Gabor-like pattern matching of V1 neurons. The model also incorporates the individual layers of neurons found in the neocortex, which are proposed to produce particular behaviors, including contour enhancement and convergence of neural activity. This model uses a regular grid layout and contains only two orientation selective columns per hypercolumn: vertical and horizontally selective columns.

Using the LAMINART model, Grossberg has suggested the functional purposes of many of the architectural feature found in the neocortex. From this model, Grossberg and colleagues demonstrated functionality such as contrast normalization, perceptual grouping [43] [44], texture segregation [45], and 3-D shape perception [46]. However, while these functionalities are of importance to a human's ability to perceive data visualizations, the application of this model to the problem of data visualization has not been explored.

2.4.4 Models of Color Perception

While perhaps not as important as form perception for the visualization problems addressed in this thesis work, the perception of color is also relevant to data visualization design. The first stage of color processing by the cones of the retina can be modeled using the CIEL*a*b* [47] perceptual color space. The Hering opponent-process theory states that humans perceive color along white-black, red-green, and blue-yellow color dimensions. The CIEL*a*b* color appearance model describes this perceptual space using L*, a*, and b* coordinates, and defines the transformation to this color space from the standardized RGB space (sRGB). In the CIEL*a*b* space, the L* coordinate specifies the location of the color along the white-black dimension, a* specifies along the red-green dimension, and b* along the blue-yellow dimension.

2.5 Conclusion

These models demonstrate that is possible to neurophysiologically model the perceptual mechanisms that play an important role in data visualizations. In this thesis work, we build upon these models to create a framework for computationally evaluating and optimizing data visualizations.

The work reviewed provides a powerful foundation for developing a computational modeling approach to data visualization. We've reviewed the relationships between data visualization and perception, between perception and neurophysiology, and between neurophysiology and computational modeling. These relationships transitively form an indirect relationship between data visualization and computational modeling, and suggest computational modeling as a means of exploring the problem of data visualization. The goal of this thesis work is to explore the problem of data visualization in the context of this relationship with computational modeling.

CHAPTER 3

NEURAL MODELING OF FLOW RENDERING EFFECTIVENESS

3.1 Introduction

Many techniques for 2D flow visualization have been developed and applied. These include grids of little arrows, still the most common for many applications, equally spaced streamlines [6] [5], and line integral convolution (LIC) [4]. But which is best and why? Laidlaw et al. [1] showed that the "which is best" question can be answered by means of user studies in which participants are asked to carry out tasks such as tracing advection pathways or finding critical points in the flow field. (Note: an advection pathway is the same as a streamline in a steady flow field.) Ware [17] proposed that the "why" question may be answered through the application of recent theories of the way contours in the environment are processed in the visual cortex of the brain. But Ware only provided a descriptive sketch with minimal detail and no formal expression. In the present chapter, we show, through a numerical model of neural processing in the cortex, how the theory predicts which methods will be best for an advection path tracing task.

Our basic rationale is as follows: tracing an advection pathway for a particle dropped in a flow field is a perceptual task that can be carried out with the aid of a visual representation of the flow. The task requires that an individual attempt to trace a continuous contour from some designated starting point in the flow until some terminating condition is realized. This terminating condition might be the edge of the flow field or the crossing of some designated boundary. If we can produce a neurologically plausible model of contour perception then this may be the basis of a rigorous theory of flow visualization efficiency.

The mechanisms of contour perception have been studied by psychologists for at least 80 years, starting with the Gestalt psychologists. With advances in our knowledge of the physiology of the visual system, neurological theories of contour perception have begun to develop. In the present chapter, we show that a model of neural processing in the visual cortex can be used to predict which flow representation methods will be better. We apply the model to a set of 2D flow visualization methods that were previously studied by Laidlaw et al. [1]. This allows us to carry out a qualitative comparison between the model and how humans actually performed. We evaluated the model against human performance in an experiment in which humans and the model performed the same task.

This chapter is organized as follows. First we describe the CortexSim 1.0 model of visual perception used in this work. Next we show how the perceptual mechanisms of the model differentially process various flow rendering methods. Following this, we show how the model predicts different outcomes for an advection path tracing task based on Laidlaw et al.'s prior work. Finally we discuss how this work relates to other research that has applied perceptual modeling to data visualization and suggest other uses of the general method.

3.2 The Computational Model - CortexSim 1.0

The CortexSim 1.0 model of the visual system processes images in three stages. The first is an edge detection stage, which computes the visual edges in the scene. The second stage, which performs edge enhancement, calculates the enhancing effect that aligned edges has on their detection. The final streamline tracing stage models the higher level cognitive process whereby a pathway is traced.

3.2.1 The Edge Detection Stage

CortexSim 1.0 computes visual edge detection using the Gabor model of V1 edge detection described section 2.4.2 and defined by equation 2.3. The input to this stage is a rasterized data visualization image. For parameters of the Gabor equation, we used $\phi = 0$, causing the Gabor function to respond to lines in the center of the receptive field. We used $\lambda = 21$ pixels and $\sigma = 7$ pixels, producing a Gabor function where the center maximum and the two neighboring minimum are significant, but more distant maxima and minima become negligible. For γ we used a value of 1, resulting in a radially symmetric Gaussian envelope.

3.2.2 The Contour Enhancement Stage

The edge enhancement stage of the CortexSim 1.0 model is based on a neural model of V1 by Li [40], which deals with the neural interactions of V1 responsible for producing the perception of an extended contour from aligned visual edges. The detection of visual edges is not part of Li's model, nor is the high-level perceptual process of streamline tracing. The CortexSim 1.0 model combines these components with Li's V1 contour enhancement model.

Following Li's model, CortexSim 1.0 uses a hexagonal array to represent the hypercolumn structure of V1. Each hex contains a set of 12 orientation selective columns, oriented in 15-degree increments, and each column contains an excitatory and inhibitory neuron, for a total of 24 neurons per hex. The signaling activity of neurons are modeled as floating point values.

Lateral connections between nearby V1 neurons cause their activity to be either

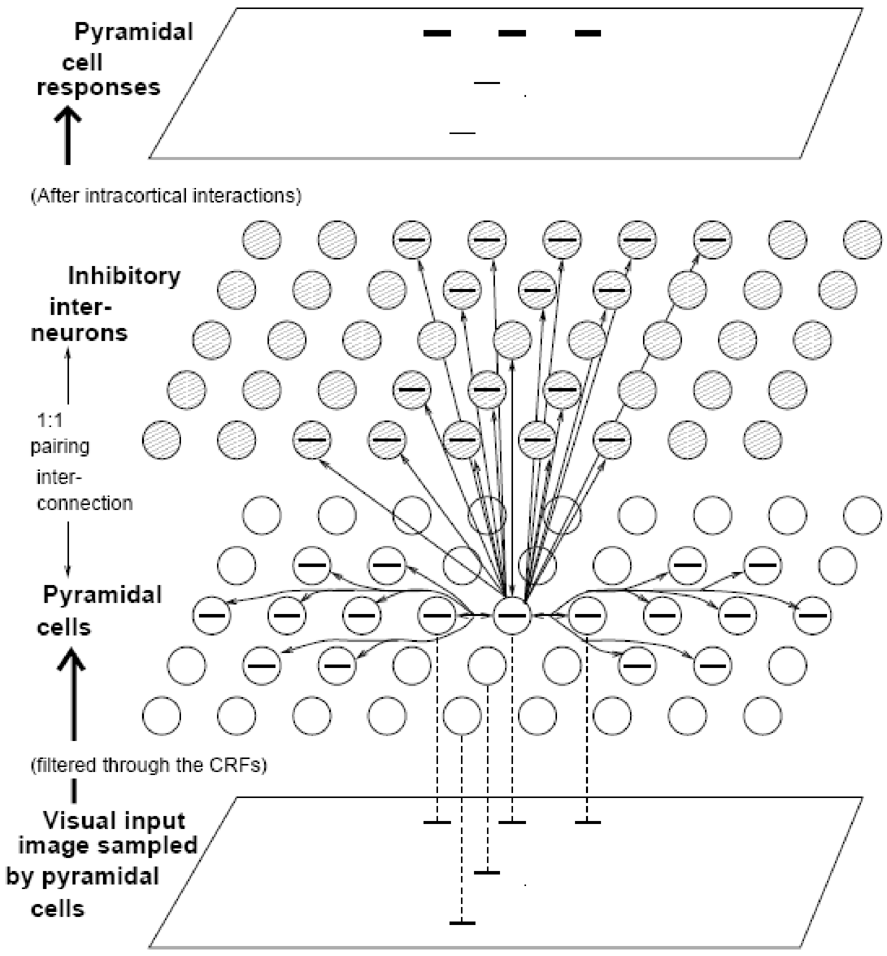

Pyramidal cell responses
(After intracortical interactions)
Inhibitory inter-neurons
1:1 pairing inter-connection
Pyramidal cells
(filtered through the CRFs)
Visual input image sampled by pyramidal cells

Figure 3-1: *Zhaoping's Model of V1 [48]. Neurons with aligned receptive fields mutually excite each other, and neurons with unaligned receptive fields inhibit each other.*

enhanced or suppressed depending on the relative arrangement of their receptive fields. These lateral connections cause neurons with aligned receptive fields to interact such that they mutually excite each other. Similarly, neurons with unaligned receptive fields interact such that they mutually inhibit each other (Figure 3-1).

These dynamics are described as follows. The excitation of the excitatory neuron varies according to the equation:

$$\frac{d}{dt}x_{i\theta} = -\alpha_x x_{i\theta} - \sum_{\Delta\theta} \Psi(\Delta\theta) g_y(y_{i,\theta+\Delta\theta}) + J_0 g_x(x_{x\theta}) + \sum_{j \neq i,\theta} J_{i\theta,j\theta} g_x(X_{j\theta}) + I_{i\theta} + I_0 \tag{3.1}$$

The first term describes the voltage decay of the neuron's value back to zero with a time constant of $1/\alpha_x$, for which we use a value of 1. The second term inhibits edges of similar orientation mapping to the same receptive field. This is done by receiving inputs from the inhibitory neurons of the same receptive field and similar orientations. This behavior is produced by the Ψ function, defined by:

$$\Psi(\theta) = \begin{cases} 1 & \text{if } \theta = 0 \\ 0.8 & \text{if } |\theta| = 15^\circ \\ 0.7 & \text{if } |\theta| = 30^\circ \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

The activation function g_y defines the signal produced by an inhibitory neuron as a function of voltage potential. Similarly, the function g_x defines the signal produced by an excitatory neuron. It has the effect of clamping the excitatory signal to between 0 and 1. These functions are defined by:

$$g_x(x) = \begin{cases} 0 & \text{if } x < 1 \\ x - 1 & \text{if } 1 \leq x \leq 2 \\ 1 & \text{if } x > 2 \end{cases} \tag{3.3}$$

$$g_y(y) = \begin{cases} 0 & \text{if } y < 0 \\ 0.21 * y & \text{if } 0 \leq y \leq 1.2 \\ 0.21 * 1.2 + 2.5(y - 1.2) & \text{if } 1.2 < y \end{cases} \tag{3.4}$$

The third term is an excitatory neuron's feedback to itself. J_0 defines the strength of this feedback loop, for which we used a value of 0.8. The fourth term produces the edge enhancement; it models the excitatory signal to neighboring neurons that lie along the orientation direction in the manner depicted by figure 3-1. The function J defines the strength of the excitatory connection to a nearby neuron, and is defined by:

$$J_{i\theta,j\theta'} = \begin{cases} 0.126e^{-(\beta/d)^2 - 2(\beta/d)^7 - d^2/90} \text{ if } 0 < d \leq 10 \\ \text{ and } \beta < \pi/2.69 \text{ or } 0 < d \leq 10 \\ \text{ and } \beta < \pi/1.1 \text{ and } |\theta_1| < \pi/5.9 \text{ and } |\theta_2| > \pi/5.9 \\ 0 \text{ otherwise} \end{cases} \tag{3.5}$$

Where θ_1 is the angle from the neuron's orientation to the line connecting the two neurons, θ_2 is the angle from the neighboring neuron to this line, $\beta = 2|\theta_1| + 2 * sin(|\theta_1 + \theta_2|)$, and d is the distance separating the neuron and its neighbor. Similarly, the function W defines the strength of the inhibitory connections from nearby neurons, and is defined by:

$$W_{i\theta,j\theta'}) = \begin{cases} 0 \text{ if } d = 0 \text{ or } d \geq 10 \text{ or } \beta < \pi/11 \\ \text{ or } |\Delta\theta| \geq \pi/3 \text{ or } |\theta_1| < \pi/11.999 \\ 0.14(1 - e^{-0.4(\beta/d)^{1.5}})e^{-(\Delta\theta/(\pi/4))} \text{ otherwise} \end{cases} \tag{3.6}$$

The fifth term is the input from the receptive field, calculated using the Gabor function described in the previous section. The sixth term describes a background signal that all excitatory neurons receive, and is set such that average neuron voltage in the network remains constant. The inhibitory neurons evolve by:

$$\frac{d}{dt}y_{i\theta} = -\alpha_y y_{i\theta} - g_x(x_{i\theta}) + \sum_{j\neq i,\theta'} W_{i\theta,j\theta'} g_x(X_{j\theta'}) + I_C \tag{3.7}$$

Here the first term again acts to decay the value of the neuron back to zero. The second term is input from the excitatory neuron in the pair. The third term acts to inhibit edges that are of the same direction, but located orthogonally to the edge. This prevents multiple parallel edges from being produced. The last term is a background signal to all inhibitory neurons.

One of the main simplifications embodied in Li's model of contour enhancement is that it fails to incorporate the way the mammalian visual systems scales with respect to the fovea. Real neural architectures have much smaller receptive fields near the fovea at the center of vision than at the edges of the visual field.

3.2.3 The Streamline Tracing Stage

Laidlaw et al. [1] compared the effectiveness of visualization techniques by presenting test subjects with the task of estimating where a particle placed in the center of a flow field would exit a circle. Six different flow field visualization methods were assessed by comparing the difference between the actual exit numerically calculated and the estimation of the exit by the human subjects. Laidlaw's experiment was carried out on humans, but in our work we apply this evaluation technique to humans as well as to our model of the human visual system and use a streamline tracing algorithm to trace the path of the particle.

We use the term *streamline tracing* to describe the higher level process that must exist for people to judge a streamline pathway. We call it streamline tracing because the task requires the user to make a series of judgments, starting at the center, whereby the path of a particle dropped in the center is integrated in a stepwise pattern to the edge of the field. Though algorithms exist in the machine vision literature for contour tracing [49], we found these to be inappropriate for use in this application. Contour tracing algorithms are generally designed to trace out the boundary of a visual shape, but a streamline tracing algorithm must also be able to produce a streamline in a field of disconnected contours, such as is the case with the regular arrows. The streamline to be traced does not necessarily follow a visible contour, but instead be located between contours, and will sometimes pass through areas devoid of visual elements. Thus we developed a specialized algorithm that is capable of tracing streamlines that do not necessarily correspond to the boundary of any shape, but can pass between visual contours.

Perception utilizes a combination of top-down and bottom-up processes. Bottom-up processes are driven by information on the retina and are what is simulated by Li's model [48]. Broadly speaking, top down processes reflect task demands and the bottom up processes reflect environmental information. Top-down processes are much more varied and are driven in the brain by activation from regions in the frontal and temporal cortex that are known to be involved in the control of pattern identification and attention [50]. All of the flow visualizations evaluated by Laidlaw et al. [1], except for LIC [4], contain symbolic information regarding the direction of flow along the contour elements (e.g., an arrowhead). In a perpetual/cognitive process this would be regarded as a top-down influence. At present our model does not deal with symbolic direction information, but it does do streamline tracing once set in the right general direction. In regards to streamline tracing, the bottom-up information is processed from the visual edges in the visualization, while the top-down information represents the cognitive process of streamline pathway tracing.

Contour integration is modeled using the following iterative algorithm:

Algorithm 1 *Contour Integration Algorithm*

```
current_position ← center
current_direction ← up
while current_position is inside circle do
    neighborhood ← all grid hexes within two hexes from current_position
    for all hex in neighborhood do
        for all neuron in hex do
            convert neuron_orientation to vector
            scale vector by neuron_excitation
            vector_sum ← vector_sum + vector
            normalize vector_sum
            current_position ← current_position + vector_sum
            current_direction ← vector_sum
        end for
    end for
end while
return current_position
```

Algorithm 1 maintains a context that contains a current position and direction. Initially, the position is the center, and the direction is set to upward. This context models the higher-order, top-down influence on the algorithm that results from the task requirements (tracing from the center dot) and the directionality which in our experiment was set to be always in an upwardly trending direction.

Algorithm 1 traces the contour by repeatedly estimating the flow direction at the *current_position* and moving the position a small distance (.5 hex radii) in that direction. The flow direction is calculated from the neural responses in the local neighborhood of the *current_position*. The excitation of each neuron is used to generate a vector whose length is proportional to the strength of the response and whose orientation is given by the receptive field orientation. Because receptive field orientations are ambiguous as to direction (for any vector aligned with the receptive field, its negative is similarly aligned). The algorithm chose the vector most closely corresponding to the vector computed on the previous iteration. Vectors are computed for all neurons in hypercolumns within a 2 hexes radius of the current position; they are summed and normalized to generate the next *current_direction*.

Minor modifications changes were made from the method published by Pineo and Ware [51]. Previously, the algorithm considered only a single hex cell at each iteration of the algorithm. This causes undesirable behavior in some cases. For example, on visualizations with arrowheads, the neural network can yield a very strong edge orthogonal to the flow field positioned at the back of an arrowhead. If the algorithm considered only the edges at this point, it can deviate excessively from the correct path, despite the edges in nearby positions indicating the correct flow direction. Averaging over a *neighborhood* is more robust and representative of human visual processing, producing a stronger correlation with human performance.

3.3 Qualitative Evaluation

Four different flow visualization methods were used in our application of the computational model to flow visualization. We used our own implementations of four of the six flow visualization techniques used by Laidlaw et al. [1]. We investigated a regular arrow grid because gridded arrows are still the most commonly used flow visualization technique in practice. We also used grid of jittered arrows because of the conjecture that jittering arrows improves perceptual aliasing problems [2]. We included Line Integral Convolution (LIC) [4] because of its widespread advocation by the visualization community [52] and head-to-tail aligned streaklets because of Laidlaw et al.'s finding that aligned streaklets produces the most accurate perception of advection pathway, and because of the theoretical arguments in support of this method [17]. Note that Laidlaw used Turk and Banks' algorithm [6] to create aligned arrows on equally spaced streamlines whereas we used Jobard and Lefer's [5] method to achieve the same effect without an arrowhead [53].

V1 is known to have detectors at different scales. However, to make the problem computationally tractable we chose only a single scale for the V1, and we compensated for this alteration by designing the data visualizations with elements scaled such that they were effectively detected by the Gabor filter used by the model. The widths of the arrows and streaklets were designed to be smaller than the central excitatory band of the Gabor filter. This permits the edge to be detected even if not precisely centered on the receptive field of the neuron. The spatial frequency of the LIC visualization is defined by the texture over which the vector field is convoluted. Our texture was created by generating a texture of random white noise of one-third the necessary size and scaling it up via interpolation. The resulting spacial frequency of the LIC visualization was of a scale that was effectively detected by the Gabor filters of the model.

Samples of how the algorithm performed with the various visualization methods are shown in Figures 3-2, 3-3, 3-4, and 3-5. For greater clarity we only show a section of each image although the application of the algorithm to the whole image was computed. In each example the original visualization is shown in panel B, in the lower left. Panel A, in the top center, shows the effect of the Li algorithm on the image following ten feedback iterations, at which point the neural excitation values

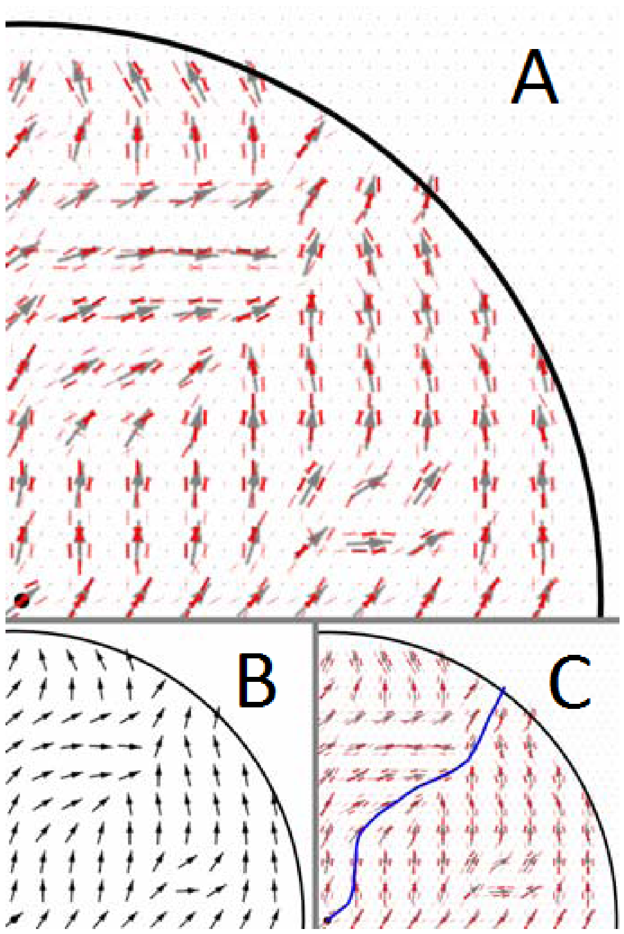


Figure 3-2: *Regular Arrows*

had stabilized. The small bars show how strongly each neuron responds, with redder meaning stronger. Panel C, in the lower right, shows the path traced out by the contour integration algorithm.

Regular Arrows

The regular arrow visualization (Figure 3-2) is produced by placing arrow glyphs with uniform spacings. The magnitude of the vector field is indicated by the arrow length, and the flow direction by the arrow head. The grid underlying the regular arrows is apparent to humans, but the edge weights of the model show no obvious signs of being negatively impacted. In fact, the regularity ensures that the arrows are well

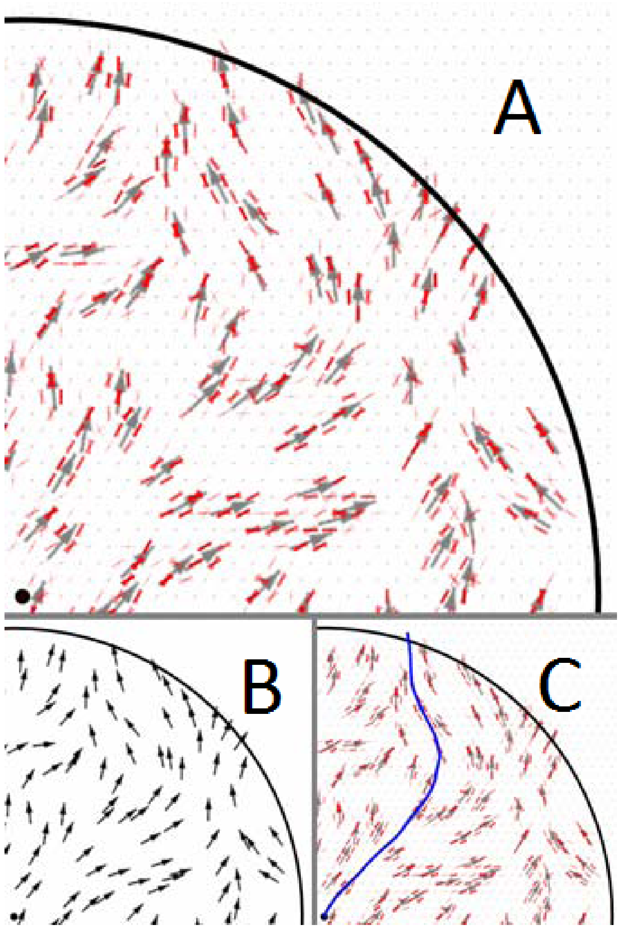


Figure 3-3: *Jittered Arrows*

spaced, thereby preventing any false edge responses produced by the interference of multiple arrows in close proximity. We expect that non-tangential edge responses will be produced by the arrowheads, leading to errors in the streamline advection tracing.

Jittered Arrows

The jittered arrow visualization (Figure 3-3) is derived from the regular arrows method, but the arrows are displaced a small random distance from the regular locations. While composed of the same basic elements as the regular grid, we observe instances where nearby arrows interfere with each other and produce edge responses non-tangential to the flow direction. As with gridded arrows, the arrowheads excite

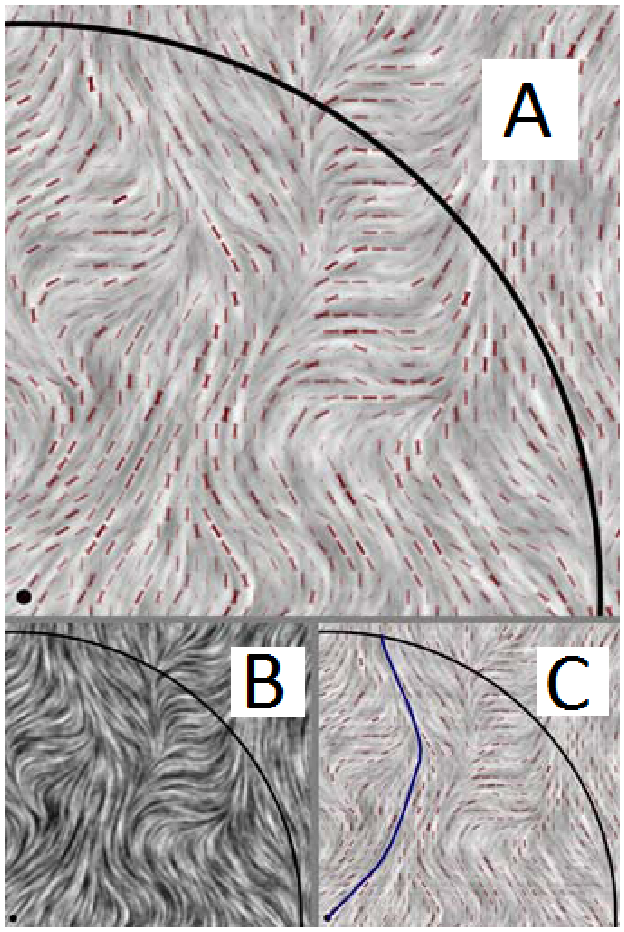


Figure 3-4: *Line Integral Convolution*

neurons with orientation selectivity non-tangential to the flow. This can be seen in Figure 3-6, where the back edge of the arrowhead causes orthogonal neural excitation to each side of the upper arrow (Figure 3-6, blue circles). We observe excitation resulting from the interference of two arrows at the bottom right (Figure 3-6, green circle). These non-tangential responses are much stronger than those found in the aligned streaklets visualization (Figure 3-7)

LIC

Line integral convolution (LIC, Figure 3-4) [4] images are formed by integrating a texture of random noise along the flow direction. The neurons of the model are not strongly excited by the LIC visualization. Elongated patches of black or white produce the strongest responses, but these are still weak compared to other visualizations. However, we note the lack of responses that are not tangential to the flow direction. A major shortcoming of LIC is that it is completely ambiguous as to flow direction which could be in either of two directions at any point. In Laidlaw et al.'s experiment [1] this lead to very poor performance for LIC. However, because algorithm 1 did not take symbolic direction into account for any of the visualization methods (which we remedied with an upward bias as described in section 3.4.2), LIC could be expected to perform better in our experiment than it would in real-world practice.

Aligned Streaklets

In the aligned streaklets visualization (Figure 3-5), streaklets are aligned such that the head of one points to the tail of the next. This visualization produced strong neural responses within the model. As Ware argued [17], theory predicts that head to tail placement of arrows should produce good results. Perceptual theory suggests that evenly spaced streamlines should provide the best stimulus for coherent chains of excited neurons to develop.

3.4 Evaluation

By comparing the performance of humans to the model on the streamline advection task, we can attempt to understand in what cases the model adequately describes contour enhancement and extraction, and in what cases other perceptual mechanics may need to be incorporated. In order to compare how well the cortical model predicted human performance, we conducted an experiment where human subjects and the model completed the same task with the same set of flow representations. We chose Laidlaw's streamline advection task [1].

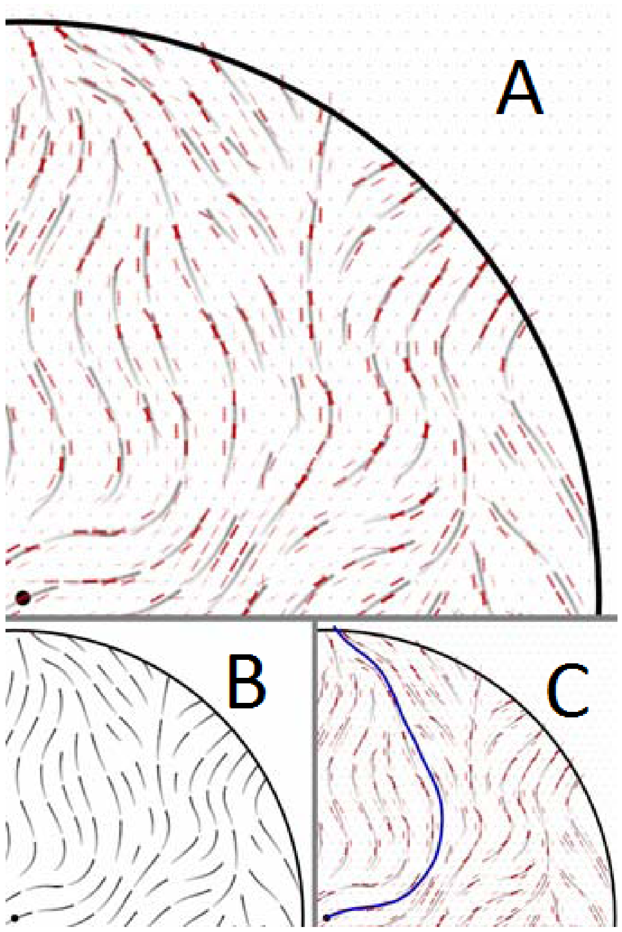


Figure 3-5: *Aligned Streaklets*

3.4.1 Participants

Six human subjects were used in this experiment. They were a mix of volunteers and paid undergraduate students. There were four males and two females.

3.4.2 Vector Field Generation

Artificial flow fields were generated by interpolating a regular 8x8 grid of vectors. Each vector is a pseudo-randomly generated, normalized vector with a positive y component. This produced a vector field with an upward trend. This trend is important for ensuring that all streamline paths eventually exit the circle. To vary the

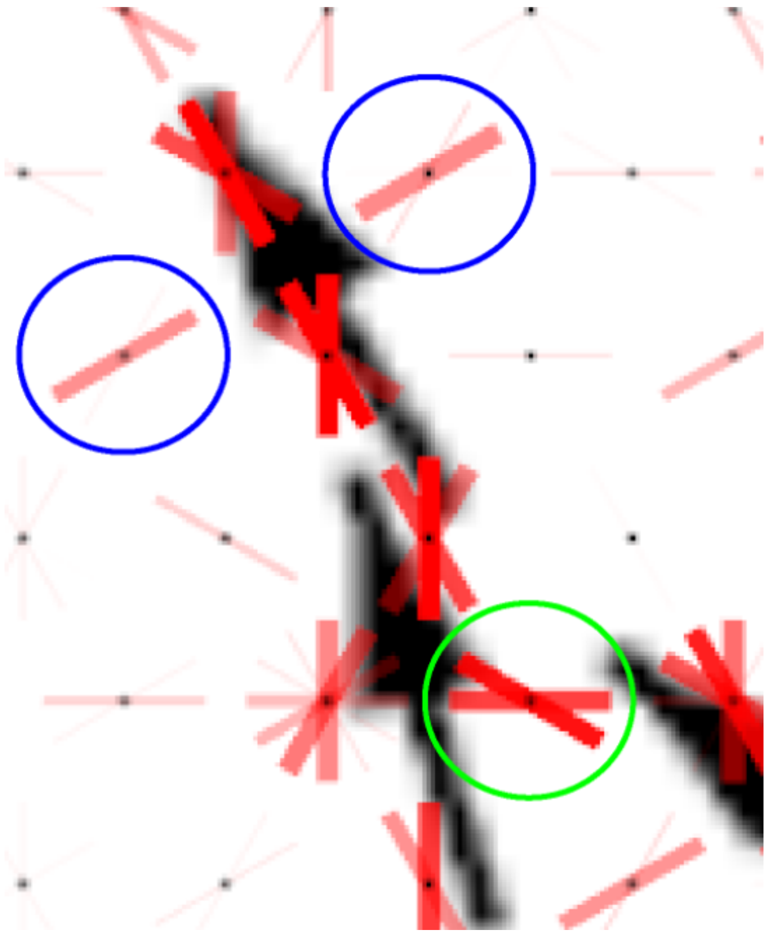


Figure 3-6: *Closeup of neural response to arrowheads. The black dots indicate the center of hypercolumns, red bars indicate the excitation of the orientation sensitive columns within the hypercolumn.*

overall trending direction of the vector field, a pseudorandom angle was produced between 45°, and all vectors were rotated by this value. The resulting vector fields thus trended in the upper 90° quadrant. This was motivated by the fact that the LIC method produces a visualization that is ambiguous; two possible flow directions may be inferred from any standard LIC image. In the experiment, we asked the subject to find pathways that trended upward. Likewise, the algorithm was also set to look for upwardly trending solutions.

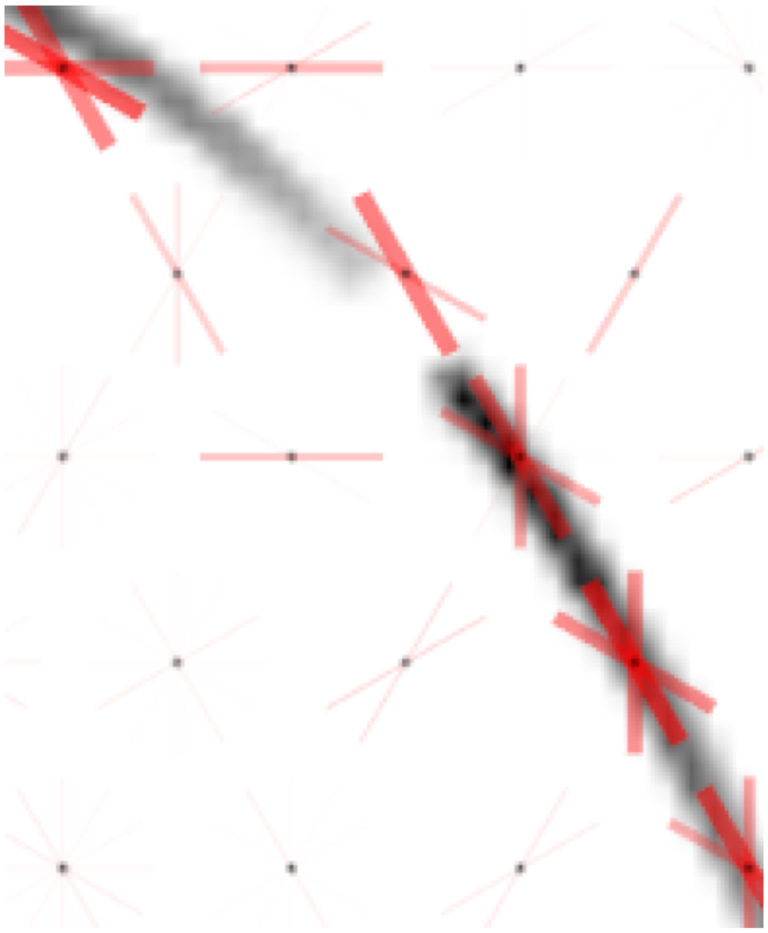


Figure 3-7: *Closeup of neural response to aligned streaklets*

3.4.3 Instructions

Subjects were asked to click where they felt that a particle deposited in the middle of the circle would exit the circle. Due to the directional ambiguity of the LIC visualization, the subjects were informed that the general trend of the flow fields would always be upward.

3.4.4 Procedure

The subjects were presented with each flow field visualization on a 15.1 inch, 133dpi LCD screen. The diameter of the circle was 4.5 inches and the viewing distance was approximately 57 cm. We evaluated four different visualization methods: regular arrows, jittered arrows, LIC, and aligned arrows. The test subjects were given as long as they needed to select with a mouse the point where they estimated that the particle in the center would exit the circle. Each test subject was first allowed to practice the task to minimize learning effects. Following this, they participated in 5 experiment blocks. Each block was constructed by generating 10 random flow fields and rendering them using the four visualization methods. The resulting 40 visualizations were then presented to the test subject in a random order. Following each block, the subjects were allowed to take a short break to minimize fatigue effects. Each test subject performed 200 tests, for a total of 1200 tests for the entire the experiment.

3.4.5 Computer Trials

The computer carried out exactly the same set of trials with exactly the same stimulus pattern generation algorithm. Raster images generated by the same four visualization methods were used as inputs.

3.5 Results

Figures 3-8 and 3-9 show histograms of the errors for the trials of the human participants and computer model, respectively. In both cases, most of the errors are less than three degrees. Because the error data were highly skewed we applied a log transform to the raw data before further analysis. This transform produced a roughly normal distribution (Figure 3-10), allowing the use of analysis of variance techniques. The geometric means for aggregated human performance and model performance are summarized in Figure 3-11. We conducted a two-way analysis of variance (ANOVA) with the two factors being aggregate human data and model output. ANOVA analysis revealed that the model was significantly more accurate than the human participants

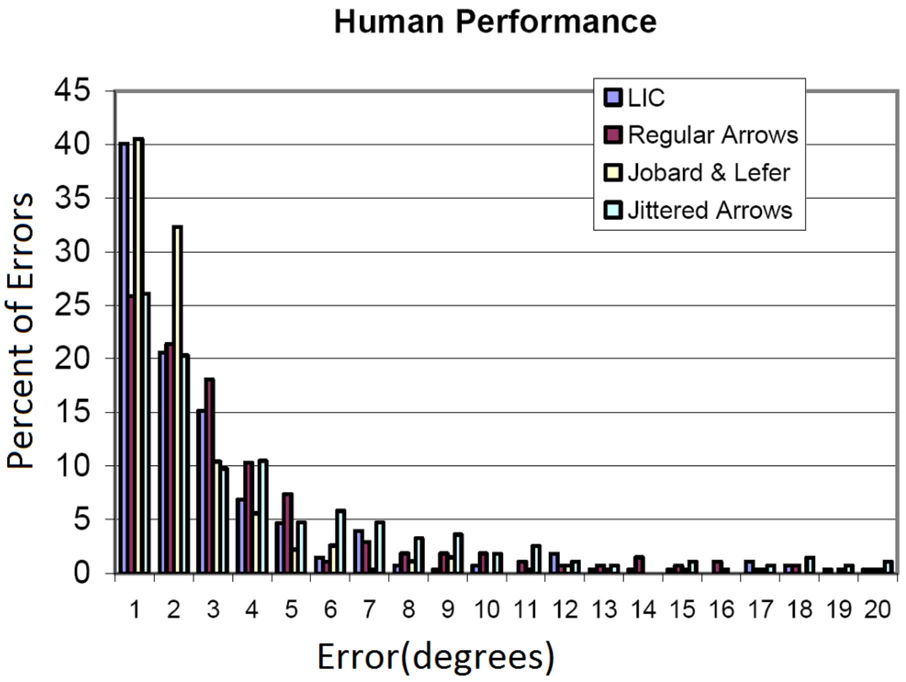


Figure 3-8: *Histogram showing distribution of errors for humans*

$[F(1, 2966) = 128.22,\ p < 0.001]$. The mean error of the CortexSim 1.0 model was .92 degrees, outperforming the human average of 1.5 degrees. There was no interaction between the visualization type and whether the subject was the human or model. There was also a highly significant main effect for the type of visualization $[F(3, 2966) = 23.43,\ p < 0.001]$. A Tukey HSD test indicated that the difference between LIC and aligned streaklets was not significant. However, there were significant differences between all other visualizations. A linear fit of the error over the block number yielded that the block number was not significant on the human trials $[F(1, 1125) = 2.29,\ p = .13]$, indicating that fatigue and learning effects were not significant.

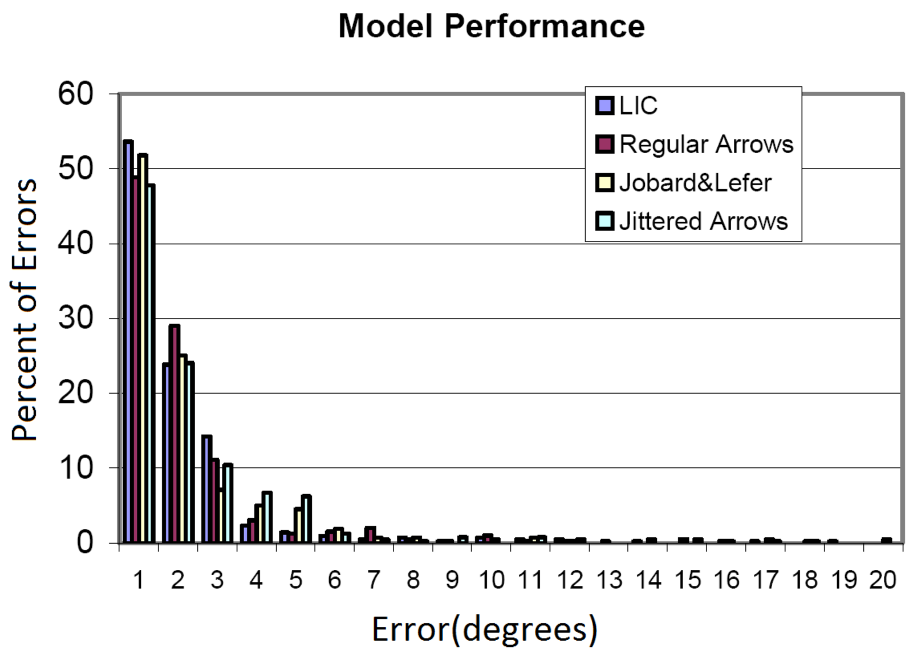


Figure 3-9: *Histogram showing distribution of errors for the computer model*

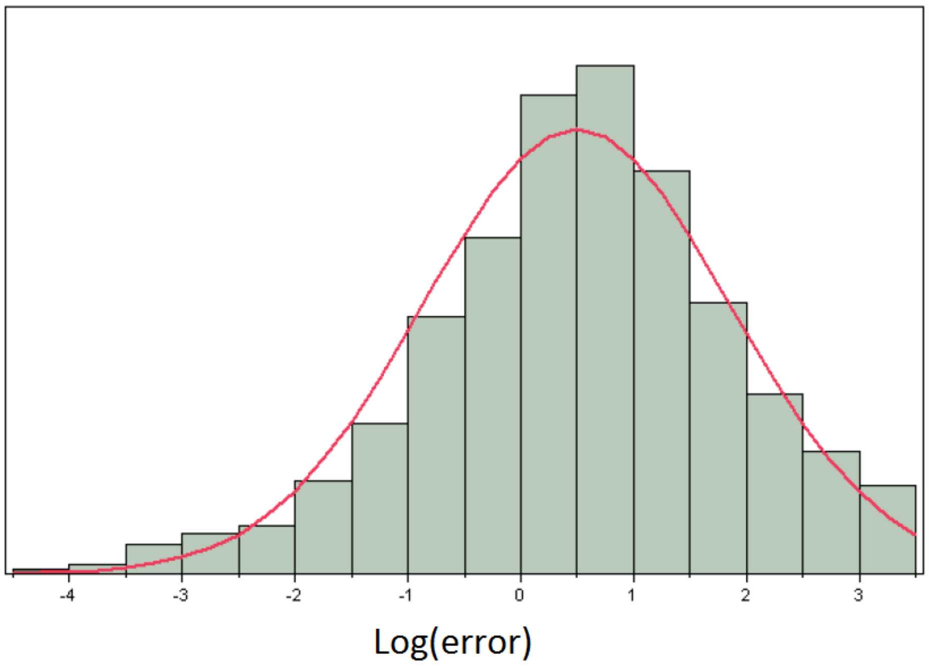


Figure 3-10: *Histogram of the log transformed data with a normal distribution overlay*

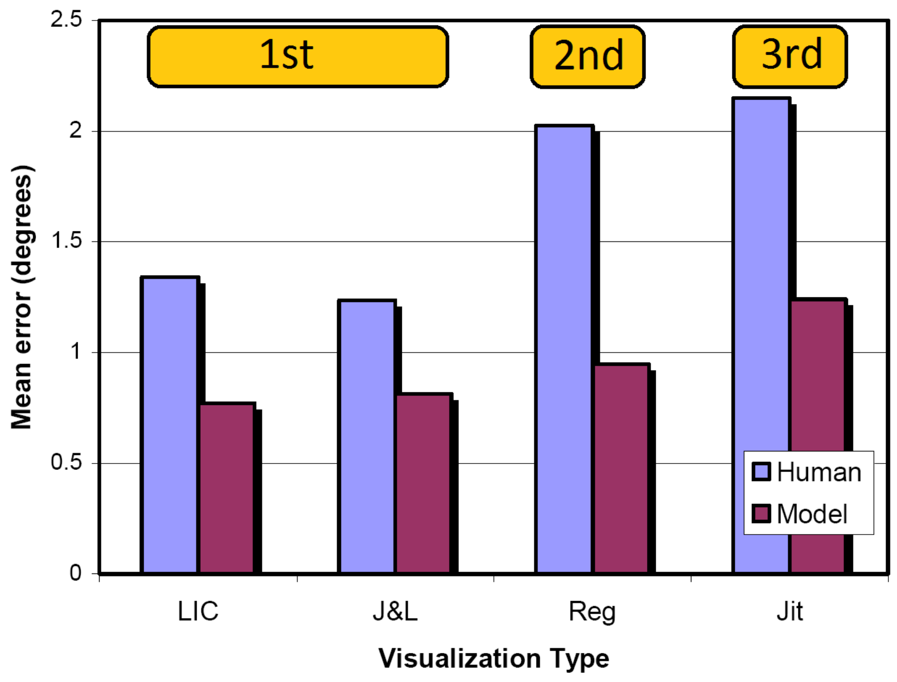


Figure 3-11: *Mean errors for the four visualization methods for human participants and the model. Statistically distinguishable groups, with their performance rank ordering, are indicated with orange boxes.*

3.6 Discussion and Conclusion

The qualitative agreement between the results for human observers and the V1-based model provides strong support of the perceptual theory outlined in the introduction. The aligned arrows style of visualization produced clear chains of mutually reinforcing neurons along the flow path in the representation, making the flow pathway easy to trace as predicted by theory.

LIC produced results as good as the equally spaced streamlines, which lends support to its popularity within the visualization community. While LIC did not produce as much neuron excitation as the aligned arrows method, this was offset by the lack of non-tangential edge responses produced by glyph-based visualizations. However, the relatively strong performance observed with the LIC visualization was achieved only because our evaluation method ignored the directional ambiguity inherent in this method by constraining flow to the positive y direction. Laidlaw et al. [1] found LIC method to be the worst and it is probable that had we allowed flow in any direction, up or down, human observers would have found pathways with close to 180 degrees of error half of the time.

The performance of both the CortexSim 1.0 model and the human test subjects is likely to be highly dependent on the underlying vector field used. As described in section 3.4.2, the vector field was generated by interpolating between an 8x8 grid of random, but generally upward pointing vectors. Consequently, when adjacent vectors in this grid point somewhat toward each other, the vector field forms an area of convergence. Convergence areas tend to funnel neighboring streamline paths together, reducing error in streamline tracing (Figure 3-2, the streamline path advects through a convergence area). Overall accuracies of both the model and human subjects may be higher than might be observed using a vector field without such convergence zones.

The computer algorithm achieved a lower mean error than human observers. The poorer performance of humans might be attributable to saccadic eye movements used to trace a path, whereas the computer did not. For the patterns we used, it is likely that the observers fixated on several successive parts of a path, and errors may have accumulated as they resumed a trace from a previous fixation. Nevertheless, we feel that the algorithm could easily be adjusted to make it give results closer to human subjects. A more sophisticated approach would be to simulate eye fixations.

The model we applied is a considerable simplification over what actually occurs in human visual processing. It uses a simple model of orientation sensitive neurons, and fails to include cortical magnification [2], the processing of the center of vision in higher detail than the periphery. Real cortical receptive fields are not arranged in a rigid hexagonal grid as they are in the model. Furthermore, the neurons of V1 respond to many spatial frequencies [54], however our model only uses one in its present form. In addition, besides the so-called *simple* cells, other neurons in V1 and V2 called *complex* and *hypercomplex* cells also have important functions [55]. For example, end-stopped cells respond best to a contour that terminates in the receptive field and understanding these may be important in showing how the direction of flow along a contour can be unambiguously shown. Moreover, visual information is processed through several stages following the primary cortex, including V2, V4 and the IT cortex. Each of these areas appears to abstract more complex, less localized patterns. Researchers are far from having sufficient information to model the operations of these stages, all of which may have a role in tracing contours. Nevertheless, the results are compelling and there are advantages in having a relatively simple model. We plan to add more complex functions in future versions of the model.

The problem of modeling perception at a higher more cognitive level is even more challenging. It is likely that humans adopt different perceptual strategies depending on the visualization. For example, with a regular arrow grid people are likely to rely less on the contour information available in the display, which is degraded due to perceptual aliasing [2], and more on higher-level cognitive strategies, such as looking ahead from a particular arrow, mentally interpolating the flow direction at the look-ahead point and repeating to the edge of the field. Conversely, in the case of aligned streaklets, a simpler contour following finding strategy will be more successful.

The success of even a quite simple artificial V1 model in predicting human performance suggests that our result could have practical as well as theoretical importance. In the following chapter, we describe using a similar model to provide a fitness function for a hill-climbing optimization process with the goal of automatically developing perceptually good visualizations.

CHAPTER 4

FLOW VISUALIZATION OPTIMIZATION VIA COMPUTATIONAL MODELING OF PERCEPTION

4.1 Introduction

Data visualization researchers are largely concerned with algorithms that transform data into a graphical form, but when it comes to deciding on the best form they predominantly resort to their own aesthetic judgment. As we have noted in section 1.1, seminal flow visualization papers that introduced widely used flow visualization methods, such those of Cabral and Leedom[4], Jobard and Lefer[5], Turk and Banks [6], and Wijk [7], give no indication of using perceptual theory in their development. More recently, Laidlaw et al. [1] carried out a psychophysical study to evaluate a number of flow visualization methods, but this too was largely atheoretical. Their study showed which methods were superior for a defined set of tasks, but said little about the reasons for the superiority. There is clearly a gap between perceptual theory and its application to data visualization, which is likely due to the difficulty in applying the theory, provided in a descriptive form, to data visualization algorithm design. This is despite the considerable body of scientific perceptual theory that can be applied to the problem [2].

The effectiveness of data visualizations can be largely attributed to the powerful processing mechanisms of human vision, with the most effective visualizations mapping data to perceptual mechanisms in such a way that the important patterns are easily perceived [2]. For example, the human visual system contains a perceptual mechanism for detecting visual contours [3]. This mechanism causes the perception of nearby, mutually-aligned edges to be enhanced, while the perception of nearby edges that are not aligned are inhibited (Figure 2-2). Successful flow visualizations take advantage of this perceptual mechanism by containing contours tangential to the flow field. We have argued, based on perceptual theory, that head-to-tail aligned contours should be better for the advection pathway perception tasks [56] [51]. Nevertheless, although perceptual theory can provide us with intuitions about the best way of conveying certain kinds of data, perceptual theory is descriptive in nature, and this limits its usefulness in data visualization design [57].

We can say the same for issues such as the saliency of visual symbols used in displays. Various researchers have argued that we should apply the theory of "pre-attentive" processing to data visualization [15] [56], but exactly how this theory should be applied is only given in vague descriptive terms. Thus the designer has no way of knowing how easily a particular symbol in a particular context can be seen.

In the present chapter, we present the case that a computational model of the human visual system can provide a necessary bridge between perceptual theory and its application in data visualization. We use the problem of 2D steady flow visualization as a motivating example throughout. In the previous chapter we showed how a computational model of contour integration in the primary visual cortex could be used to evaluate 2D flow visualization methods. In the present chapter, we show how a much more powerful computational model, one that uses high performance GPU processors, can be used in a perceptual *optimization* process.

4.1.1 High-level Perception

A human's understanding of data visualizations does not occur in the early visual system. The visual patterns detected in the early processing stages are further processed by higher-level areas of cognition enabling viewers to perform cognitive tasks with the visualization.

In perceptual theory, *top-down* influences refer to the way that high-level cognitive processes, such as the task at hand, affect the low-level processing that occurs in areas such as V1. Unfortunately, these top-down processes are not understood in sufficient detail to be amenable to the kind of modeling that can be done for low level processing, and in any case are highly variable, depending on a large number of possible task requirements.

Nevertheless, we can make the reasonable assumption that there is a cognitive mapping from low-level perception to high-level understanding, implying that erroneous low-level perception will have a degrading effect on high-level understanding. This mapping depends on top-down factors, such as the context, task, and low-level factors such as the detection of orientation or more salient features.

In a flow visualization, there is a natural mapping between contour orientation and flow direction. For example, a horizontal edge is readily understood as a left-right flow direction. The interpretation of color is more arbitrary, it depends on factors such as common usage or the comparison to a provided color key. In our data visualization evaluation, we use the natural mapping for orientation, and chose the arbitrary mapping for color with blue mapping to low speed and yellow mapping to high speed.

An effective visualization, in the context of this work, is one where the important patterns in the data visualization are preserved and even enhanced when processed through the low-level mechanisms of the human visual system (the retina and V1), making these patterns amenable to accurate interpretation by higher-level, more cognitive stages of the visual system. Such patterns can represent the streamlines in a flow visualization, symbols on a map, or the lines used to connect nodes in node-link diagrams.

4.2 The Computational Model - CortexSim 2.0

The design of the CortexSim 2.0 computational model used in this chapter is inspired by the models by Li [48] and Grossberg [58] [44] introduced in sections 2.4.3 and 2.4.3, respectively. The CortexSim 2.0 uses 12 orientation columns per hypercolumn like the Li model, but uses a regular layout like the Grossberg model. We also implement a difference-of-Gaussians retinal response and a V1 Gabor response [39], similar to the Grossberg model. Most importantly, while the Li and Grossberg models run until feedback produces a steady state, in contrast, our feedback consists of a single lateral excitation stage. The reasons for this are twofold. First, to model perception in the moments after viewing, we are interested in the feedforward activity before steady-state activity is reached. Second, calculating the neural activity until steady state is reached is computationally time consuming, and is prohibitive in our optimization approach where evaluations must be calculated quickly.

CortexSim 2.0 is based on the known physiology of the human visual system and seeks to produce the behavior found by psychophysical researchers and expressed by perceptual theory. We model the stages of human vision using a multi-layered, partially connected, artificial neural network (See figure 4-1). The layers of the model consists of a 2D array of either neurons (in the retina), or hypercolumns (in V1).

There exists substantial variation in the physiological measurements of the human visual system, between studies and even between individuals of a single study. This variation can often be as large as a factor of two or more. Thus, model parameters derived from physiological measurements are approximated, often to the nearest power of two for computational reasons.

The human visual system is known to process scenes at multiple scales [54] [59]. While it is likely to process at a continuum of scales, our model approximates this by processing the visual image at three resolution scales, using powers of two for reasons of computational efficiency and to minimize scaling aliasing. The finest scale uses a 512x512 array, with each element separated in visual space by .008 degrees of visual space, which corresponds roughly to the .006 degrees subtended by a retinal cone. The two subsequent scales reduce resolution by factors of two. Thus, the medium scale uses a 256x256 array and the coarsest scale uses a 128x128 array, which correspond to .016 and .032 degrees per element, respectively.

Biological neurons operate by transmitting and receiving voltage spikes, with active neurons being considered those that are transmitting voltage spikes most rapidly. We model this behavior with an overall spiking rate. For example, a rate of one corresponds to a neuron transmitting voltage spikes at its maximum rate. A neuron with a rate of zero corresponds to a biological neuron at a minimal, baseline firing rate. Negative rates are interpreted as a response to an inverse pattern. For example, if a neuron that is selective for a white-center, black surround responds negatively, this is interpreted as a response to a black-center, white surround pattern.

Neurons send their signals to other nearby neurons within the layer, as well as to neurons in other layers. The connection strengths between neurons are specified by 16x16 kernel arrays in our model.

4.2.1 The Image

The visualization is rendered to a 512x512 sRGB video buffer array. The values in this array indicate the light illuminating the cone receptors of the retina. We take

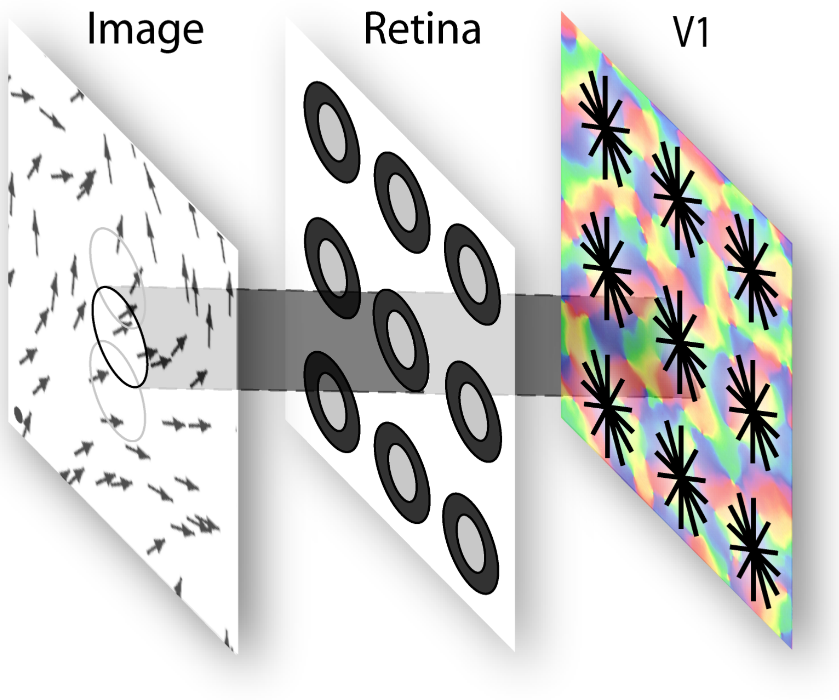


Figure 4-1: *The neural network architecture. The visual image is first processed at each position by the center surround retinal neurons, then in the second stage by the edge selective neurons in the primary visual cortex (V1).*

the red, green, and blue values to correspond to the intensity of light at the three wavelengths to which retinal cones are sensitive. When this array is displayed to a 133dpi monitor, the visual angle of one array element is approximately equal to one foveal cone when viewed at 140cm. One degree of angle corresponds to 128 pixel elements, or .008 degrees/pixel.

4.2.2 The Retina

The first neural layer of the model, the retinal layer, models the perceptual processing of the visual scene done by the retina. The opponent-process mechanism produced by the bipolar cells of the retina is modeled with a conversion to L*a*b* perceptual coordinates, according to the CIEL*a*b* transformation [47]. The black-white luminance dimension, the red-green chromatic dimension, and the yellow-blue chromatic dimension are referred to in the model by I^{L*}, I^{a*}, and I^{b*}, respectively.

The retina also computes a center-surround field (Figure 2-6) by combining the center signal of bipolar cells with a surround signal from horizontal and amacrine cells. This center-surround signal is output by the retinal ganglion cells of the retina to subsequent stages of the visual system. The center surround receptive field output of the retinal ganglion cells is defined in the model as a Difference-of-Gaussians (Equation 2.1, Figure 2-6). The black-white, red-green, and yellow-blue center-surround retinal responses are defined as:

$$R_{i,j}^{w-b} = \sum_{x,y} DoG_{x,y,\sigma_1,\sigma_2}(I^{L*})_{i+x,j+y} \tag{4.1}$$

$$R_{i,j}^{r-g} = \sum_{x,y} DoG_{x,y,\sigma_1,\sigma_2}(I^{a*})_{i+x,j+y} \tag{4.2}$$

$$R_{i,j}^{y-b} = \sum_{x,y} DoG_{x,y,\sigma_1,\sigma_2}(I^{b*})_{i+x,j+y} \tag{4.3}$$

The values of σ_1 and σ_2 specify the size of the center and the surround fields. We use values of 1 and 2 for these parameters, respectively, to produce a center-surround receptive field like the one shown in figure 2-6. For the values of α_1 and α_2 we use 1 and .5, yielding a receptive field with center-surround characteristics, but that produces a positive response to a uniform field.

4.2.3 V1 Edge Detection

Hypercolumns within the V1 component of the model exhibit both edge detection and edge enhancement behavior. As in Li's model [48], we use 12 columns per hy-

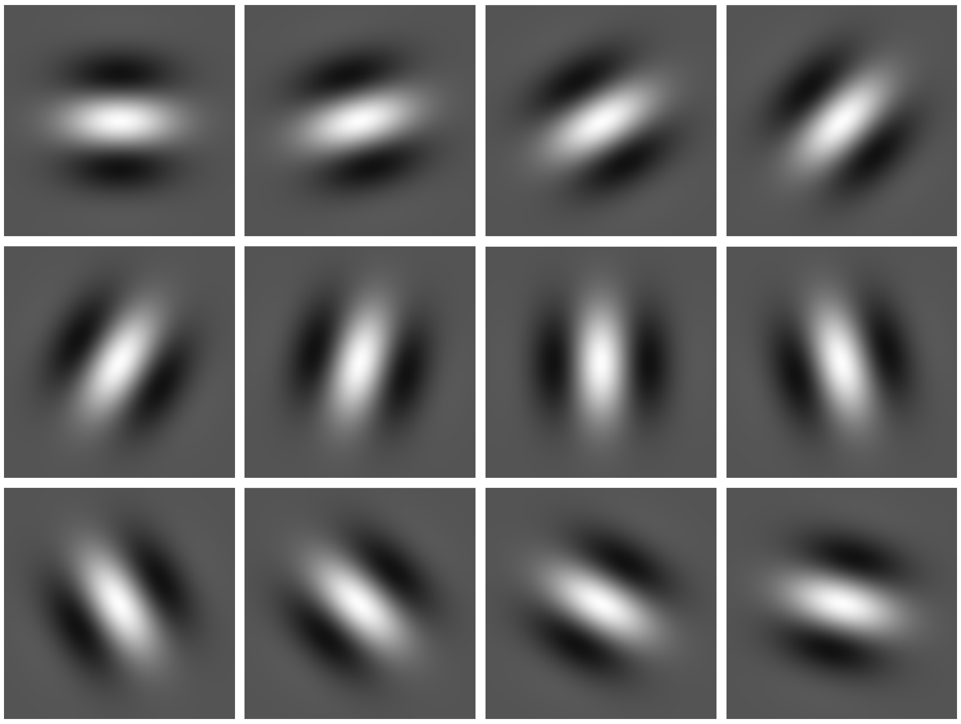


Figure 4-2: *The V1 Gabor kernels [39] used by the model. Neurons in light areas produce an excitatory effect, neurons in dark areas produce an inhibitory effect.*

percolumn, responding to orientations varying by 15° increments (Figure 4-2). Edge detection results from the Gabor pattern [39] of synaptic connections from the retina, defined by:

$$V1_{i,j,\theta} = |\sum_{x,y} Gabor_{x,y,\theta} R^{w-b}_{i+x,j+y}| \tag{4.4}$$

The black-white luminance signal, which is responsible for form perception, is used here. We use a value of 7 for λ (Equation 2.3), which gives a spatial frequency of 8.9 cycles/degree for our medium scale resolution, roughly corresponding to the parafoveal processing in the 2° to 5° eccentricity range found by Foster [33]. At the high and low resolution scales this corresponds to 17.9 and 4.4 cycles/degree, respectively. We use $\sigma = 2$ to define the Gaussian envelope, which was chosen to encapsulate a single cycle of the sinusoid. The absolute value causes the column to respond positively to both light-centered and dark-centered edges. x' and y' are found by rotating x and y by θ degrees:

$$[x', y'] = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{4.5}$$

4.2.4 V1 Edge Enhancement

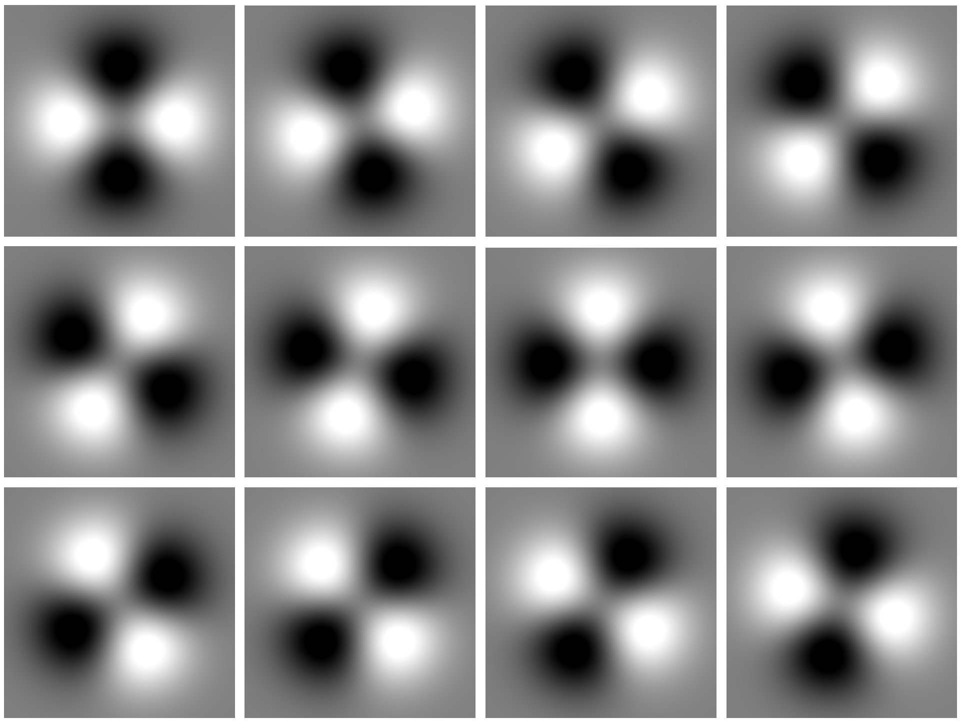


Figure 4-3: *The V1 edge enhancement kernels used by the model. Light areas produce an excitatory effect, dark areas produce an inhibitory effect.*

Columns of V1 exhibit enhanced activity when they correspond to an edge that lies along a continuous contour. The pattern of synaptic connections (Figure 4-3) used in the model to produce this behavior is defined by:

$$E'_{x,y,\theta} = G_{x,y,\sigma}(x'^2 - y'^2) \tag{4.6}$$

Yielding the enhanced V1 column activity:

$$V1'_{i,j,\theta} = \sum_{x,y,\theta} E'_{x,y,\theta} V1_{i+x,j+y,\theta} \tag{4.7}$$

4.2.5 GPU Model Implementation

The model was implemented in nVidia's CUDA GPU programming environment (Appendix A), and run on an nVidia GTX470 graphics processor. The highly parallellizable nature of the neural network model allowed for efficient use of the GPU parallel processing capabilities, yielding performance far beyond what would be possible on conventional CPU processors. The model required approximately 50ms to run, and consumed nearly 300MB of graphics memory. The neuron layers were stored in GPU memory as arrays of single precision floating point numbers, and operated upon by graphics kernels.

4.3 Approach to Optimization

Visualizations encode information with visual variables that correspond to the feature dimensions detected by V1. For example, flow visualizations use the orientation visual variable to express vector flow orientation. Our work begins with the premise that, in effective visualizations, the visual processing of those visual variables closely matches the data being expressed. In visualization where visual variables are interpreted as information, it is clearly desirable that those visual variables are processed in a manner that accurately reflects the data that is being conveyed. If a viewer's processing of orientation within a flow visualization does not correctly correspond to the flow direction, then the viewer's judgment of the flow direction will likely be impaired.

4.3.1 Graphical Primitives

A key issue is the choice of the graphical primitives used in the optimization and their parameterization. For example, we might choose to use a grid of arrows and simply optimize the line with and arrow density. However, because arrow grids are known to be poor [1], even an optimized solution is unlikely to be acceptable. A more sophisticated primitive can take into account prior work in data visualization, suggesting that a particular style of presentation may be more effective. For example, curved streaks that are tangential to the flow pattern and that have more distinct heads than tails are suggested by a number of sources [53] [17].

The parameterization determines the space of visualizations that will be searched by the optimization algorithm, and it is thus important that it be chosen carefully. An excessively liberal parameterization will result in a large search space, yet an overly restrictive parameterization will cause desirable solutions to fall outside the search space.

To better understand these issues, we chose to investigate our optimization approach with two kinds of primitives: one using streaklets, and a second using black or white pixels.

Streaklet Primitives

In the first parameterization, we use a set of graphical streaklet primitives, with each streaklet defined by a center point, a length, and a color. The path of the streaklet is found by advecting from the center point a distance of half the length in both the upstream and downstream directions. Thus we drastically reduce the search space at the cost of not exploring visualizations with graphical elements non-tangential to the flow direction. The color of the streaklets can be used to express speed using an arbitrary color map. In our parameterization we use blue to correspond to low speeds and yellow to correspond to high speeds. In addition, the width of streaklets can be used to express speed with one parameter to represent the gain factor.

Pixel Primitives

In the second parameterization, we define the visualization with a 512x512 array of black and white pixels. This enlarges the search space considerably, allowing the algorithm to search dense flow visualization techniques. We chose to limit the parameterization to black and white pixels, rather than greyscale or color, to limit the search space. Even when limited to black and white pixels, the space has a size of 2^{512^2} visualizations.

4.3.2 The Hill Climbing Algorithm

The visualization optimization is based on the hill climbing search technique (Figure 4-4, Algorithm 2). The algorithm begins with an arbitrary and potentially poor solution. For the streaklet-based parameterization we chose to start with a blank visualization, and for the pixel-based parameterization we chose to start with random white noise. This choice primarily affected the convergence time of the algorithm, and had little effect on the resulting visualization. The visualization is repeatedly modified slightly, and the modifications are kept only if they result in an improved effectiveness. In applying the algorithm to flow visualization optimization using the streaklet primitives, we modify the flow visualization by randomly adding or removing a streaklet, or by slightly modifying the parameters of an existing streaklet. For the visualization using pixel primitives, we invert a random pixel, changing it from white

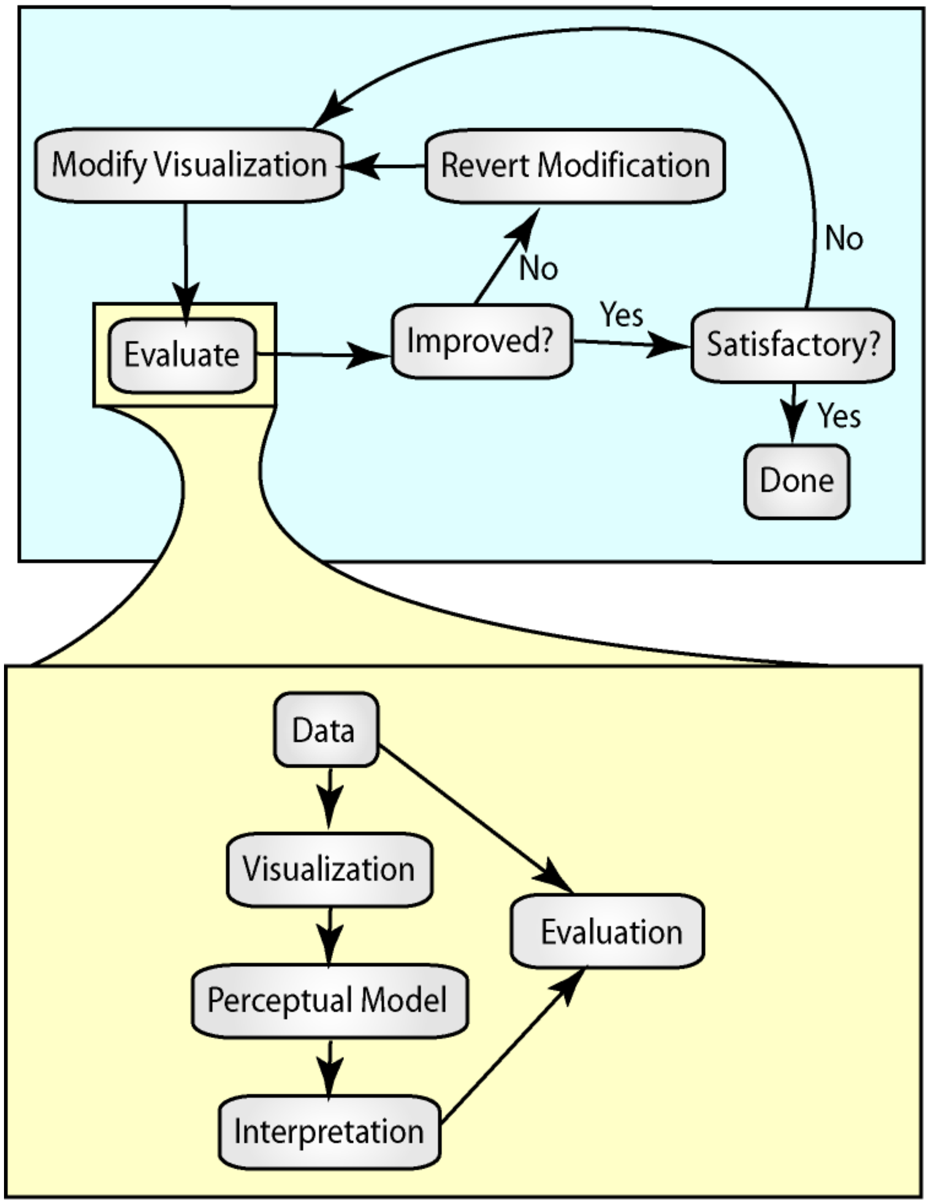


Figure 4-4: *The data visualization optimization block diagram, showing the hillclimbing algorithm (top) and the evaluation function (bottom).*

to black, or vice versa. The process of modifying and keeping improvements continues until some satisfactory solution is reached. While more efficient search strategies may exist, hill climbing's simplicity makes it appropriate for an initial search technique.

Algorithm 2 *Hill Climbing Visualization Optimization Algorithm*

```
$Visualization \leftarrow Blank$ or $Random$
while $Visualization$ is not satisfactory do
  Randomly add, remove, or modify an element of $Visualization$
  $Effectiveness \leftarrow$ effectiveness of $Visualization$ (Algorithm 2)
  if $Effectiveness > BestEffectiveness$ then
    $BestEffectiveness \leftarrow Effectiveness$
  else
    Revert changes to $Visualization$
  end if
end while
```

4.3.3 Evaluation Metric

The evaluation metric is a measure of how accurately the data visualization conveys data to the viewer. This is calculated by interpreting the neural activity of the model as perceived data, and comparing the perceived data to the data that we are attempting to convey. For example, flow visualizations convey flow direction by using perceptual edges tangential to the flow direction. However, it may be possible for non-tangential edges to be detected in the visualization. This can result, for instance, from the shapes of arrowheads, or by the positioning of nearby streaklets such that the arrowheads are aligned perpendicularly to the flow. By comparing the vector data with the predicted perception of edges, we can estimate how accurately the data is being perceived by the viewer. The overall evaluation is calculated as:

$$Eval = \alpha OrientationEval + (1 - \alpha) SpeedEval \tag{4.8}$$

The value of α specifies the relative weight of the perception of orientation and speed in the final evaluation. By setting the value of α to one, the evaluation of speed perception can be optionally ignored. Likewise, the evaluation of orientation perception can be ignored by setting α to zero. The *OrientationEval* and *SpeedEval* terms are described below.

4.3.4 Orientation perception

To allow the analysis of orientation selective neurons in our model, we translate the neural activity into a vector representation. This allows the overall orientation of a set of neurons to be calculated via a vector sum. An orientation can be naturally represented as a vector aligned with the orientation, and is uniquely defined if constrained to the positive y direction. However, this creates complications when performing analyses that incorporate averages or sums of orientations because such calculations result in a strong bias in the positive y direction. Instead, we translate to a vector with double the angle between the orientation angle and the positive x-axis, producing angles that range from 0 to 2π. The orientation vector is calculated as:

$$\vec{O}_{i,j,\theta} = \begin{bmatrix} V1'_{i,j,\theta}\cos(2\theta) \\ V1'_{i,j,\theta}\sin(2\theta) \end{bmatrix} \tag{4.9}$$

From the activation of the edge sensitive neurons in the model, we predict the perception of orientation, $\vec{O}'_{x,y}$, produced by the data visualization. The predicted orientation at a point in visual space is calculated from the activity of edge selective neurons with receptive fields near the point being predicted. Nearby neural activities are converted to vectors in orientation space and summed, weighted by their distance to the visual point being predicted.

$$\vec{O'}_{i,j,\theta} = \sum_{x,y,\theta} G_{x,y}\vec{O}_{i+x,j+y,\theta} \tag{4.10}$$

To evaluate the predicted perceived orientation at a point in the visualization, we calculate the projection of the predicted orientation perception vector onto a normalized orientation vector calculated from the data. This yields values that are largest when the predicted and actual orientations are closely aligned, and smallest when unaligned. The overall orientation evaluation is found by summing the projection over all points at each of the model's three scales.

$$OrientationEval = \sum_{s} \sum_{i,j} \vec{O}'_{i,j} \cdot \frac{\vec{Actual}_{i,j}}{|\vec{Actual}_{i,j}|} \tag{4.11}$$

4.3.5 Speed Perception

Data visualizations frequently use color to encode a data variable (e.g., [12]). In flow visualizations this is typically flow speed. In such cases, the perception of speed may be included by using a non-zero value of α in equation 4.8. Perceived colors must be interpreted as data values with the aid of some mapping from colors to values. The perceived speed is calculated from the blue-yellow chromatic signal produced by the retina, R^{y-b}. A Gaussian blur is applied to model the loss of detail relative to the black-white signal.

We use a linear mapping, with saturation, for the interpretation from colors to speed:

$$S_{i,j} = min(max(\alpha \sum_{x,y} G_{x,y} R^{y-b}_{i+x,j+y} + \beta, 0), 1) \tag{4.12}$$

where α and β define the mapping between the colors and the data values. As such, these values are dependent on the data set used. To evaluate the speed, we sum the absolute difference between the predicted perceived speed and the actual speed over the three model scales:

$$SpeedEval = -\sum_{s}\sum_{i,j}|S_{i,j} - |\vec{Actual_{i,j}}|| \qquad (4.13)$$

This produces evaluations that are largest when the predicted speed closely matches the actual speed, and smallest when there is a large difference.

4.4 Results

Figure 4-5 shows a streaklet-based visualization optimized by the algorithm for orientation perception ($\alpha = 1$, thus ignoring speed evaluation). The data was created from a 5x5 uniform grid of randomly oriented, normalized vectors. First, we notice that the streaklets are spaced out, much like the Jobard & Lefer algorithm (Figure 4-6). It has been conjectured that the effectiveness of aligning streaklets in a head-to-tail fashion, as in the Jobard & Lefer algorithm, is a result of the early perceptual processing of the visual system. To find such streaklet alignment produced as an *emergent* property of our perceptual optimization method lends support to this conjecture.

The head-to-tail alignment may may result from the inhibition of perception of parallel, unaligned edges in V1. In addition, as the spacing between streaklets is reduced, there is an increased likelihood of a edge non-tangential to the flow being perceived between nearby streaklets. Thus, a balance is reached between these factors degrading the perception of direction when streaklets are spaced too closely, and the lack of orientation information in between streaklets when spaced too sparsely.

However, it's worth emphasizing that unlike the Jobard & Lefer algorithm, this spacing is not explicitly specified. The spacing emerges because it is found by the algorithm to be perceptually optimal. One consequence of this is that the spacing is not a strict rule, and is not produced in some areas. We see this in the figure in areas where the flow field diverges or converges. The spacing of the streaklets that develop

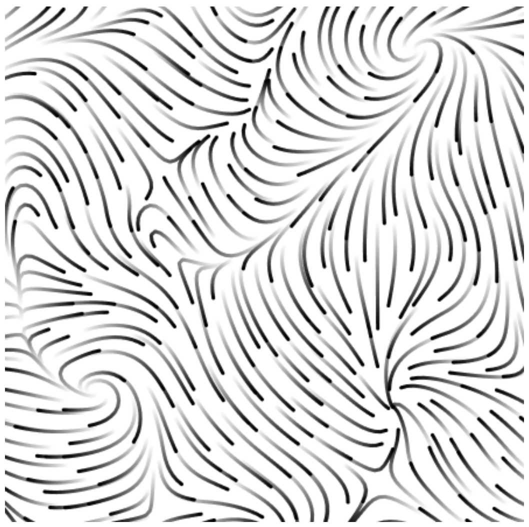


Figure 4-5: ***(Left)*** *Flow visualization optimized by our algorithm using a streaklet parameterization, note the emergent head to tail alignment.*

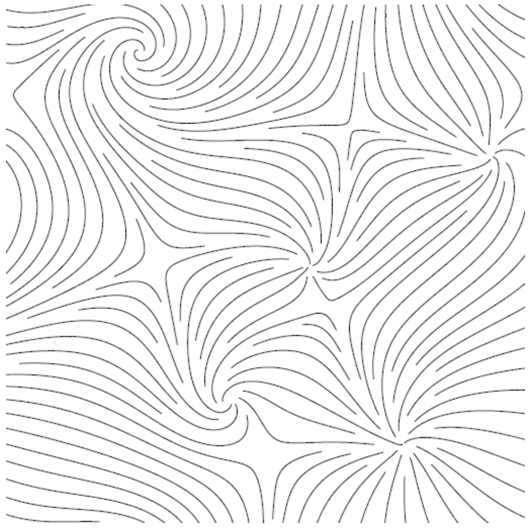


Figure 4-6: ***(Right)*** *A flow visualization created by the Jobard & Lefer algorithm (From Jobard & Lefer [5]).*

elsewhere in the visualization do not emerge in these areas, producing a clear display these converging and diverging areas.

Figures 4-7 and 4-8 demonstrate the algorithm on atmospheric flow data from the U.S. Navy Operational Global Ocean Model (NCOM). The visualized region shows the Jet Stream of the northern hemisphere. Figure 4-7 shows a visualization using black and white streaklets optimized by the algorithm for orientation perception by using $\alpha = 1$. Figure 4-8 shows a visualization using color streaklets, optimized for both orientation and speed perception using $\alpha = .5$. In this case, the blue-yellow perceptual dimension was used to represent flow speed, with the resulting visualization clearly showing the strong jet stream current flowing from left to right across the visualized area.

Using the pixel-based parameterization (Figure 4-9), we see convergence to a vi-

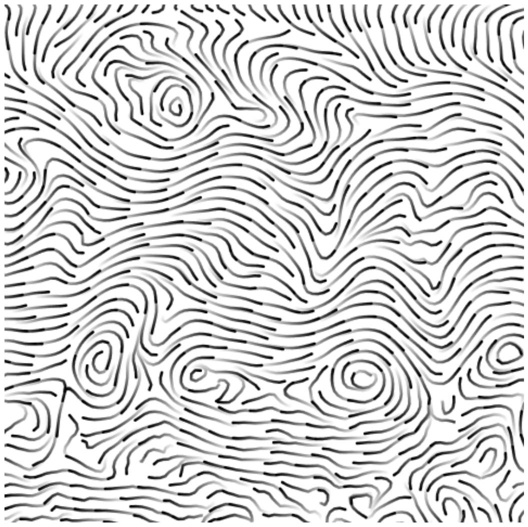


Figure 4-7: ***(Left)*** *Jet stream visualization produced by our technique by optimizing the placement of black and white streaklets.*

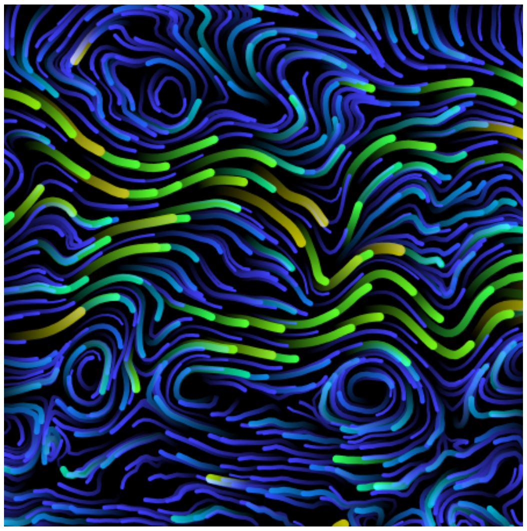
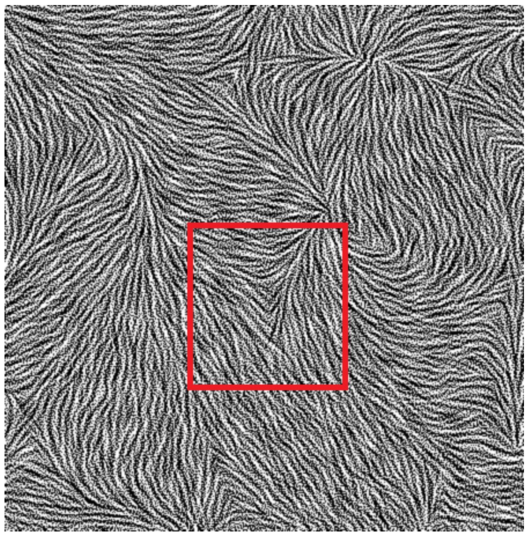


Figure 4-8: ***(Right)*** *Jet stream visualization produced by optimizing a color streaklets with* $\alpha = .5$.

sualization solution that more closely resembles those generated by the line integral convolution (LIC) technique of Cabral and Ledom [4] (Figure 4-10). We do, however, observe important differences. There is a clear difference in the spatial frequency of the texture in the direction perpendicular to the flow direction. The frequency of the LIC visualization results from the underlying noise texture being convolved. In Figure 4-10, the underlying texture is a white noise, which results in frequencies as low as .5 cycles/pixel. This frequency can be adjusted by altering the convolved texture, but such textures are typically not chosen based on the perceptual properties of vision. The perpendicular wavelength generated by our technique is approximately 7 pixels, produced from our choice of λ for the Gabor kernels of V1.

One notable limitation of our technique, with respect to LIC, is the ability to visualize areas of high curvature. In such areas, such as indicated by the red box



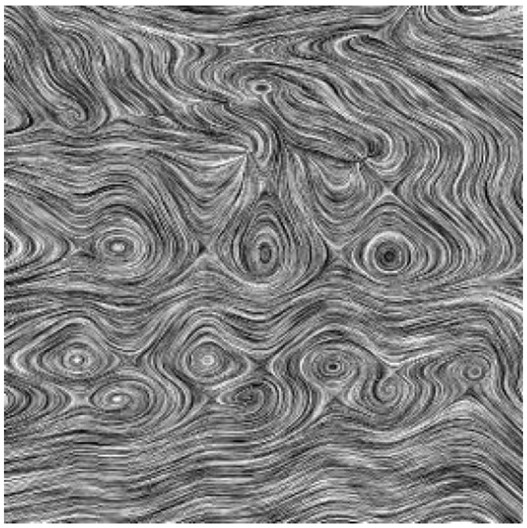


Figure 4-9: ***(Left)*** *The flow visualization produced by our technique using a pixel parameterization. The boxed area indicates discontinuities in the visualization.*

Figure 4-10: ***(Right)*** *A flow visualization created by the Line Integral Convolution method (From Cabral & Ledom [4]).*

in Figure 4-9 with a closeup shown in Figure 4.4, we see the orderly arrangement of the visualization break down and produce sharp discontinuities. We also observe the effects of the large search space resulting from the pixel-based parameterization. In particular, the pixel-based parameterization did not produce extended contours or the spacing observed from the streaklet parameterization. This is due to the large number of pixels that must be inverted to move a pixel-based contour to a more optimal position. Moving a contour requires many steps, and the hill-climbing optimization requires that each intermediate visualization have an improved evaluation.

We found that for the streaklet-based parameterization the search required several minutes to converge on a satisfactory visualization, using an nVidia GTX470 graphics card. The pixel-based parameterization required over 24 hours to obtain the result shown in figure 4-9.

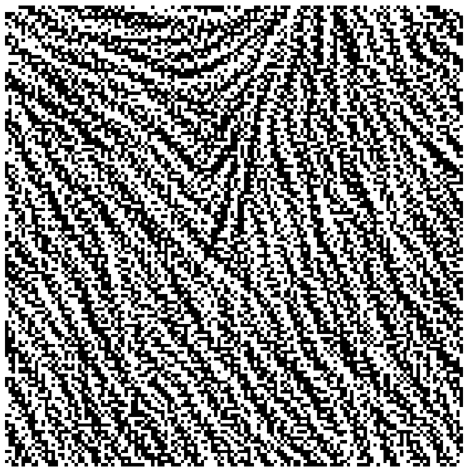


Figure 4-11: *A closeup of the boxed area in figure 4-9 showing the discontinuities produced in areas of high curvature.*

4.5 Discussion

In the introduction, we proposed that a computational model of human vision could be valuable for two reasons: first, it could help bridge the gap between perceptual theory and design guidelines for visualizations; second, it could be useful as a tool to assist in the production of visualizations.

With respect to perceptual theory, we find it significant that the optimization algorithm converged upon visualizations similar to the popular Jobard & Lefer algorithm [5]. This supports the conjecture by Ware [17] that head-to-tail alignment of display elements effective because it better stimulates the contour finding mechanisms of the primary visual cortex.

We also assert that we have provided some support for the claim that an artificial visual system can support design activities, although we recognize that the examples provided here are not yet fully realized solutions, and we do not claim that they are better than visualizations produced by skilled designers. Obviously, the solutions that were produced were highly constrained by the graphical primitives we used.

This suggests that the key to successful application of what lies in the adoption of primitives that reduce the dimensionality of the search space, but not so much that the best solutions are excluded.

Ultimately, an artificial visual system may be used as a design aid. The role of the designer might be to set broad design parameters, choose the primitives, and specify the tasks. The computational system can optimize and evaluate solutions. Such a system could provide rapid feedback to a designer as he or she works to create a visualization. There is also potential for automatic optimizations to find solutions not considered by designers. For instance, overriding the streaklet spacing in areas of high convergence or divergence seems obvious in retrospect, but it was not an expected result. Other uses for an artificial vision approach are quality control for complex displays. It may be possible to use our method to test a large set of symbols against a large set of backgrounds and, by using our evaluation technique, select symbols that are visually compatible. This is not typically done at present because of the difficulty of the task.

Although we believe that using a low-level model can help in ensuring that a design meets goals relating to low-level properties, such as salience and the perception of orientation, the process cannot be fully automatic. It seems inevitable that some element of design must go into the selection of primitives used. These high-level design decisions have parallels in top-down processing, because they directly relate to issues of how a visualization will be used in the visual thinking process. Thus designers still have a crucial role to play in the use of tools such as the one we have devised.

A natural objection to the proposal that a model of human vision can be useful in optimizing visualization is that human subjects will produce better and more reliable results. Human factors experiments are often used to evaluate different displays in order to determine if the display meets operational requirements [60]. Why then do we need a simulation? The problem is that this method is too laborious for any but relatively simple display problems. Complex displays can involve the use of dozens of different symbols, on backgrounds consisting of different textures and colors, together with linear features that use line styles and colors. Using human factors experiments to ensure that all symbols and lines are clearly visible against all possible background combinations is not possible since the number of conditions is an exponential function of the number of factors to be considered leading to an experiment that is too large to be carried out. Our system is capable of executing an experimental "trial" several times a second and it can be run 24 hours a day. It can easily execute a million trials in a single day. Compare this to a human factors experiment where, for example, each trial may last five seconds with a typical session lasting less than an hour. We may succeed in running 5 subjects a day, resulting in 4000 trials a day, although in our experience this is optimistic. Thus, the computer method is over two orders of magnitude faster. This means that evaluations, which would be infeasible using human subjects, become practical.

There is no doubt that the simulation can be optimized for superior speed. There is clear room for improvements in the search algorithm. We use a simple stochastic hill climbing technique, which randomly makes alterations to the visualization. Improvements could be made by making those alteration preferentially in areas where the model has calculated high perceptual error. A well known limitation of hill-climbing search algorithms is the propensity to find local maxima in the search space. While more sophisticated search algorithms may yield visualizations that are calculated to be even more perceptually accurate, we found that a simple hill-climbing approach produced satisfactory solutions in reasonable time.

One can view data visualization as a communication problem. In communication systems, source data is encoded, transmitted across a medium, and then decoded at the destination. The quality of such a system lies in its ability to quickly and faithfully transmit data from the source to the destination. Viewed from this perspective, visualization can be considered the encoding stage of this system. The quality of this system as a whole depends not only on this encoding, but on the decoding done by the human visual system.

While we would not argue that the visualizations we have generated thus far are the best that have been produced for this type of data, we believe that they are at least quite good and certainly better than the arrow grids that appear in many scientific publications and on the web sites of modeling groups. Furthermore, it is encouraging to consider that there is still significant room for improvement using this technique. As we better understand the functioning of the visual system produced by its neurobiology, and more accurately model human visual mechanisms, the perception predicted by the model will become more accurate. Because this prediction is used to search for the optimal data visualization, this will translate to an improvement in data visualizations produced by our method.

We have shown how, by using a computational approach, perceptual theory can be applied to data visualization to produce tangible results. In doing so, we address a fundamental question of data visualization: What does it mean to be a good data visualization? In the context of this work, we give our answer: A good data visualization is one that triggers neural activity that encodes the desired information in the brain.

CHAPTER 5

COMPUTATIONAL MODELING OF NODE-LINK GRAPH PERCEPTION

5.1 Introduction

To demonstrate the generality of a computational approach to the evaluation and optimization of data visualizations, we apply the method to the problem of non-directed node-link graph diagram layout. The high-level information conveyed by node-link diagrams differs vastly from that conveyed by flow visualizations. Typically, flow visualizations are used to express vector direction information, whereas graph-link visualizations express relationship information. However, at the level of early perception, we realize that these visualizations have much in common: both are composed of perceptual edges. Thus, this form of visualization is suitable for the application of our perceptual modeling method. The primary difference in the perceptual processing of the two forms of visualization is how those perceived contours are interpreted as information.

The layout of node-link diagrams is typically accomplished by maximizing aesthetic measures that are known to correspond with what is considered good layout and conducive to high performance on graph-related psychophysical tasks [13] [19]. The alternative approach developed here is to evaluate and optimize a node-link graph layout by using the CortexSim computational model of human perception. The model is used to estimate the speed and accuracy with which the node-link edges are perceived under alternative layout conditions. The approach is to apply the model to predict the V1 perception of contours in the node-link visualization. Next, a perceptual curve tracing operator is applied to trace the contour. This operator describes how visual attention progresses along a contour from some starting location. In areas where the contour is unclear, such as at intersections, the progression of attention slows down. In straight sections of the curve, a stronger perceptual contour is constructed, allowing for a more rapid progression. The accuracy and time required in tracing contours of the node-link diagram using this operator is used as a metric for its quality.

5.2 Background

Graphs are mathematical objects that define a set of relationships, known as edges, among a set of elements known as nodes. Many data sets are naturally expressed as graphs, and visualized using node-link diagrams. Node-link graph diagrams are a form of data visualization designed to convey the presence or absence of graph edges, represented visually by curves or line segments, between the nodes, represented by dots (Figure 5-1). Such a visualization allows viewers to quickly comprehend the relationships between the informational elements, including transitive relationships which may be depicted by dots connected via two or more curves or line segments.

The layout of node-link diagrams is a classical problem in data visualization. There exists quite a bit of freedom in the choice of layout of the node-link graph diagram, and the quality of a layout is typically judged with respect to graph "aesthetics" [62]. These aesthetics include attributes such as the amount of edge curvature and the number of edge crossings. High quality node-link graph layouts tend to have a minimum of curvature and crossings [19], the best layout algorithms aim to minimize these variables. However, there are other factors not generally taken into account,

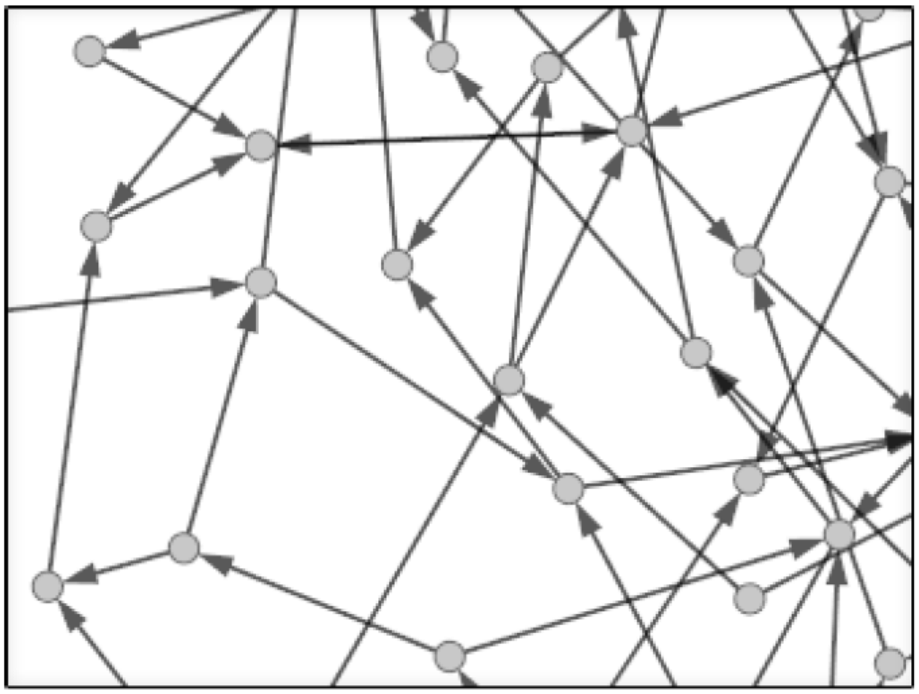


Figure 5-1: *A example of a node-link graph diagram (from Holten [61])*

such as the tradeoff between curvature and crossing angles. A more perceptually based model may be able to take these factors into account to produce a more optimal layout solution.

The perceptual theory of contour perception predicts that performance will be best when the edges connecting the nodes are straightest and have minimal edge crossings. These predictions have been confirmed by experimental evidence [63].

5.2.1 Layered and Two-layer Node-link Graph Diagrams

Layered graph diagrams are a notable subclass of node-link diagrams. The nodes of these diagrams are arranged such that they lie in distinct layers (Figure 5-2). This layout facilitates the understanding of structural attributes of the graph, such hierarchical and sibling relationships. For this reason, layered graph layouts have found use in areas such as dependency diagrams, flow diagrams, and conceptual

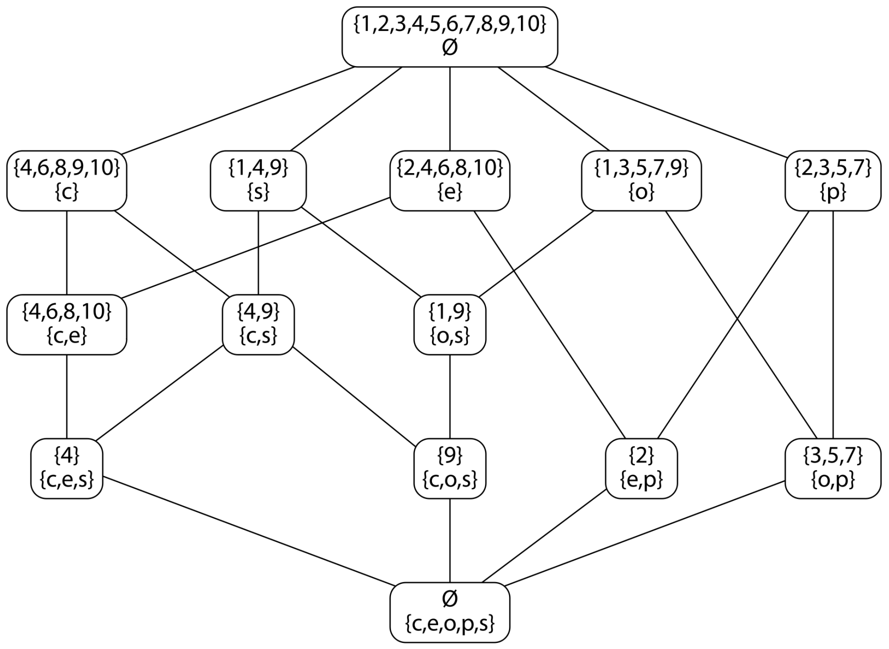


Figure 5-2: *An example of a conceptual lattice visualized as a layered node-link graph diagram [64]. The problem of layered node-link graph diagram layout can be reduced to the problem of two-layer graph diagram layout.*

lattices.

A method commonly used by graph layout systems for producing layered graph diagrams is the two-step approach described by Sugiyama et al. [65]. The first step of the approach is to arrange the nodes of the graph into layers, adding additional nodes if necessary to ensure that no link spans more than one layer. Next, the individual layers of the graph are arranged, starting at one end of the hierarchy and working toward the other. Thus, the Sugiyama method reduces the problem of multi-layer graph layout to the problem of a two-layer graph diagram layout.

5.2.2 Perceptual Theory and Node-link Diagram Aesthetics

The aesthetic principles of minimizing curvature and crossings form the basis of popular graph layout algorithms, such as Dot [66] and Dynadag [67]. These principles are used as metrics for optimization in these algorithms. However, such principles are descriptive, as is often the case with perceptual theories. Thus, applying these aesthetic principles in graph layout algorithms is often done in an *ad hoc* manner. What is needed is a perceptual theory of contour tracing expressed as a computational model.

5.2.3 Roelfsema's Curve Tracing Experiment

How contour tracing occurs at a neurophysiological level can be understood from the experiments of Roelfsema et al. [68]. These experiments investigated the activity of neurons in macaque monkeys during the perception of continuous contours. The macaque subject initially focused vision on a small circle. Then, briefly, a pair of continuous contours were shown: a target contour extending from the circle, and a distractor contour that did not extend from the circle (Figure 5-3). Both contours had a similar shape and were permitted to intersect. The subject's task was to locate the far end of the target contour.

During this task Roelfsema et al. [68]. recorded the activity of V1 neurons with edge sensitive receptive fields along the contours. They found that the activity response of these V1 neurons to the visual contours proceeded in two distinct stages. In the first stage, there is an initial, parallel detection of visual edges, whereby V1 neurons corresponding to an edge in the visual scene become active. This processing occurs simultaneously among neurons all corresponding to a point along either the target or distractor curves. It begins about 40 ms after onset of the stimulus, which is approximately the typical latency of a V1 neuron. After about 150 ms, the second

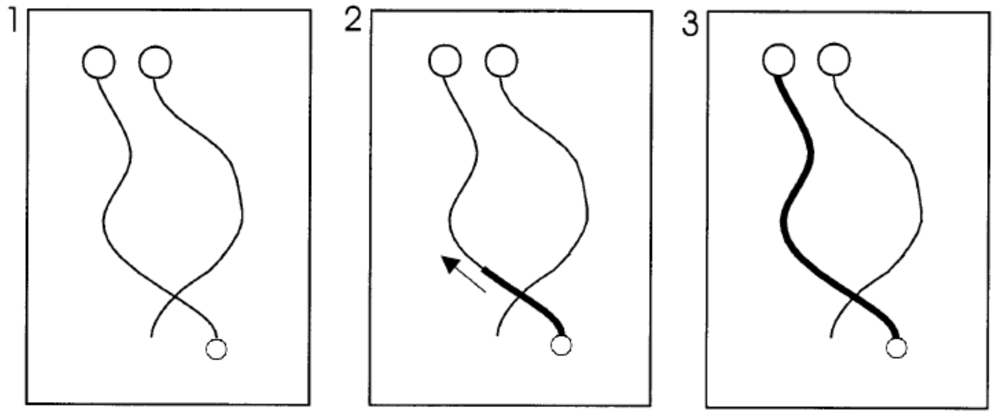


Figure 5-3: *Neurons with receptive fields along the attended contour become excited sequentially. Enhanced activity begins at the center of vision, and target spreads along the target contour. (Modified from Roelfsema et al. [68])*

stage of processing begins. In this second stage of processing, some neurons enter an enhanced state of activity. This enhanced activity spreads sequentially across the contour, beginning at the center of focus, and ending at the far end of the target contour. This enhanced activity spreads only to neurons corresponding to edges that compose the target contour. Neurons corresponding to the distractor contour do not enter the enhanced activity state associated with this second stage of processing, even if the target contour intersects the distractor contour.

5.2.4 Roelfsema's Curve Tracing Model

The Roelfsema curve tracing operator model seeks to describe the physiological behavior observed in the curve tracing experiments (Figure 5-4). Initially, the neurons in this model begin in a "silent" state, corresponding to a baseline level of activity. When the visual contours are presented to the model, feedforward edge detection processing causes all neurons corresponding to the edges of the contours to enter the "active" state, which corresponds to the first parallel stage of activity. The activity of these neurons enable lateral connections between neurons with nearby receptive fields and are tuned to collinear orientations. These connections model the lateral orientation selective connections found in V1 [37]. The second serial stage begins with a neuron corresponding to the initial point of focus on the target contour entering a “labeled” state. This labeled activity spreads through the connections enabled by the first stage of processing.

While the Roelfsema curve tracing model provides many details, it is not a computational model. To apply the curve tracing operator to data visualization, it is necessary to first implement it in a computational manner, and integrate it into the CortexSim computational model of perception.

5.3 The Computational Model - CortexSim 3.0

The CortexSim 3.0 computational model (Appendix B) used to simulate the perception of node-link diagrams is a direct descendant of the CortexSim 2.0 model used for flow visualization optimization. Roelfsema’s “active” neurons are modeled using the edge detection and enhancement processing of the CortexSim 2.0 model. The CortexSim 3.0 model includes an additional cortical area to model the spread of labeling along a perceptual contour as described by Roelfsema.

5.3.1 The Label Cortex

While it is not clear precisely how the tracing described by Roelfsema occurs at the physiological level, we speculate that such processing may occur in the areas of the posterior parietal cortex. Areas of the posterior parietal cortex have been found to be both retinotopic and sensitive to attended location, making them plausible locations for the implementation of such a mechanism. We refer to this speculative cortical area in the CortexSim 3.0 model as the *label cortex*.

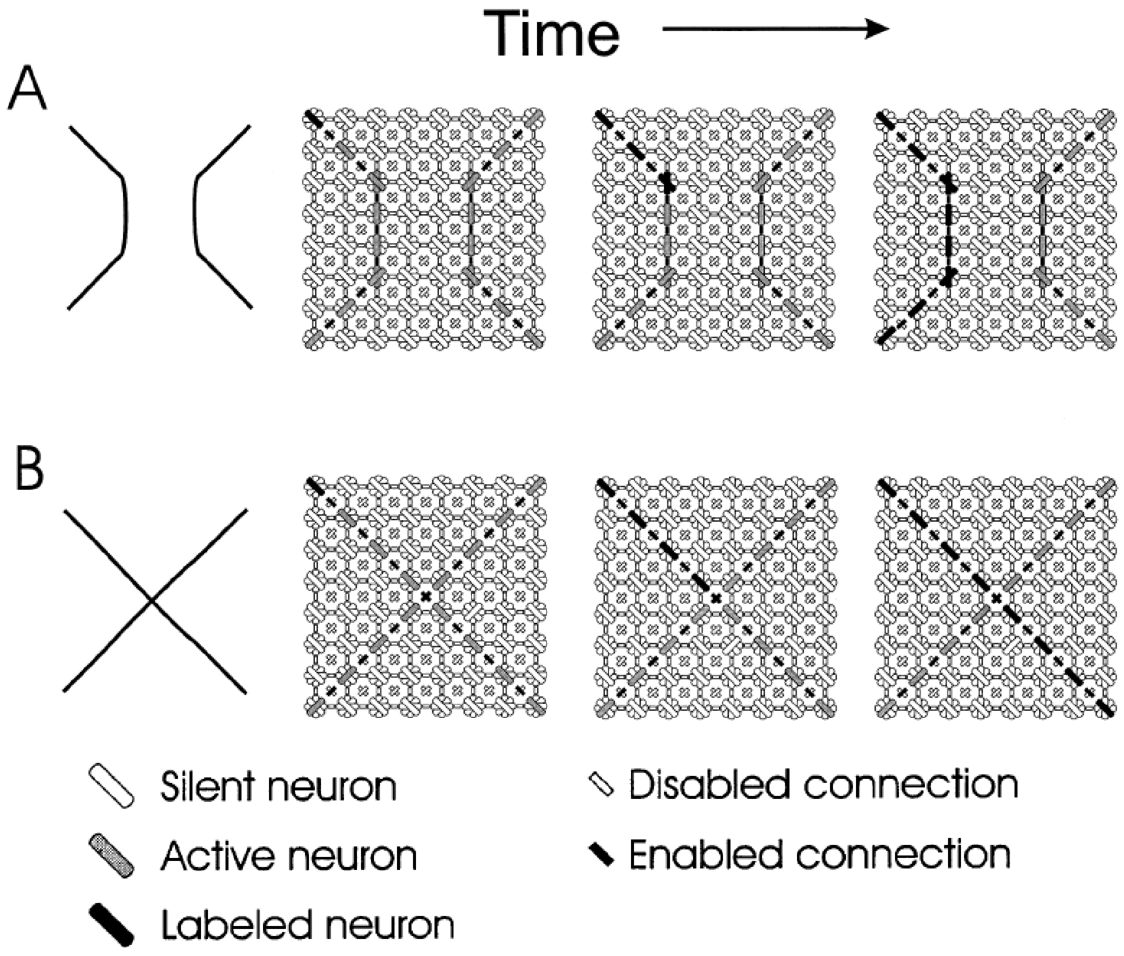


Figure 5-4: *The contour perception visual routine proposed by Roelfsema et al. [68]. First, parallel bottom-up processing activates neurons corresponding to a perceptual edge (shown in grey). Next, a sequential tracing operator labels the neurons as part of the target curve, starting from the upper-left corner (shown in black) (Modified from Roelfsema et al. [68]).*

The tracing operator is implemented in the model using feedback activation from the label cortex to V1. The neural activity corresponding to the labeled state originates with activity in the label cortex and produces signals that feed back to V1. The feedback signals then propagate both laterally throughout V1, as well as feedforward back to the label cortex. The signaling along the lateral path propagates the labeling

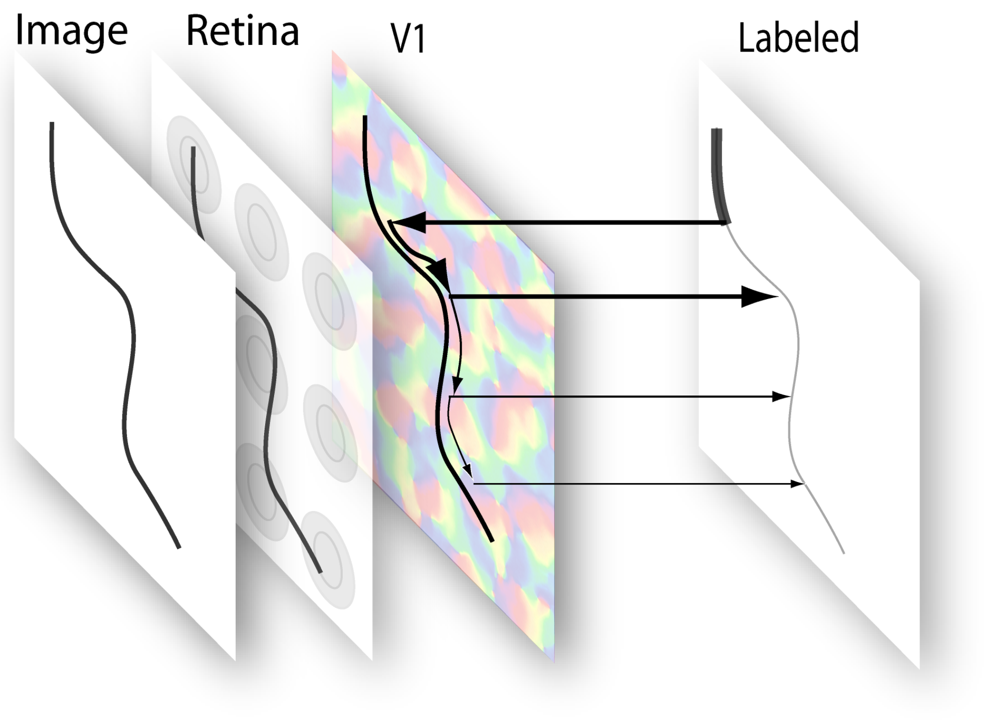


Figure 5-5: *Bottom up edge detection and enhancement produces initial activity in V1. Then, feedback signals from higher-level cortical areas travels laterally across the contours detected in V1, before looping back to the attention cortex.*

state along the activated contour, the feedforward path serves to update the labeled state of neurons in the label cortex (Figure 5-5). This behavior is expressed by the following dynamics:

$$S^{(0)}_{i,j,\theta} = V1'_{i,j,\theta} L(t)_{i,j} \tag{5.1}$$

$$S^{(n+1)}_{i,j,\theta} = V1'_{i,j,\theta} \sum_{x,y} S^{(n)}_{i+x,j+y,\theta} E_{x,y,\theta} \tag{5.2}$$

$$L(t+1)_{i,j} = \rho \sum_{n} \sum_{\theta} S^{(n)}_{i,j,\theta} \tag{5.3}$$

These equations describe the spread of neural activity in the label cortex associated with the labeled state produced by the feedback activity between the labeling cortex and V1. Activity in the label cortex is denoted with an L, and activity in V1 that is associated with the labeled state is denoted with an S. The parenthesized superscript indicates the number of times the labeling signal has been relayed laterally. Equation 5.1 describes the base case in which the feedback signal has reached V1, but no spreading within V1 has occurred. The excitation in V1 produced by the feedforward edge detection and enhancement, denoted by V1' (Equation 4.7), acts as a gating operator in this feedback loop. When V1' is close to zero, it serves to shut off the feedback signal path. As a result, label propagation occurs only among neurons corresponding to contours that were detected by the feedforward portion of the model. The signal is also relayed laterally across V1 neurons with collinear receptive fields as described by Equation 5.2. The relayed signal depends on both the gating produced by the V1 neuron, V1', and by the synaptic strength of the lateral connections, denoted by E and defined in Equation 4.7. The overall feedback signal from V1 to the label cortex is found by summing over all the feedback paths as expressed by Equation 5.3. The signal rapidly decays as it is relayed over several lateral connections due to the repeated multiplications by V1' and E, thus only the first four components are computed in the CortexSim 3.0 model. Noise is introduced into the model with ρ, which is a uniform random variable between .95 and 1.05. This enables the model to sometimes trace a different contour in cases where the contour path is ambiguous such as with acute crossings.

5.3.2 Grossberg's Shunting Neural Dynamics

The CortexSim 3.0 model contains feedback between V1 and the label cortex that, unless normalized, becomes unstable, amplifying neural activity without bound over many iterations. This problem is addressed by incorporating Grossberg shunting neural dynamics [58] [69] to normalize local neural activity in the label cortex after each iteration. Grossberg's shunting neural dynamics describe how populations of neurons behaving according to the Hodgkin and Huxley equations [28] can produce competitive and normalizing behavior. In the shunting network, neuron activity has an inhibitory effect on neighboring neurons, producing competition among the neurons within the region that causes the total local signal to be bounded to a constant. We use the steady-state form of these dynamics to find L'_i, the locally normalized activity in the label cortex:

$$L'_i = \frac{Bf(L_i)}{A + \sum_{k=0}^{n} G_k f(L_k)} \tag{5.4}$$

These dynamics describe the competition between a neuron, x_i, and neighboring neurons, $x_{k \neq i}$. The local neighborhood is defined by the Gaussian envelope, G_k, centered around the neuron x_i. The overall activity within the Gaussian envelope is bounded by B, which is set to one. The variable A, for which we use the value 0.1, describes the leakage current of the neuron and serves to limit the amplification of noise in cases where surrounding activity, $\sum_k f(x_k)$, is near zero. The activation function $f(x)$ describes the mean output signaling of the neuron as a function of the input signal to the neuron. In shunting networks, activation functions with a positive second derivative serves to suppress low activity neurons and enhance high activity neurons. Physiologically, the activation function of neurons has been observed to closely resemble a sigmoid function [70], defined by:

$$f(x) = \frac{1}{1 + e^{s(m-x)}} \tag{5.5}$$

Thus, we employ the portion of the sigmoid function before the inflection point, to achieve both noise suppression and biological plausibility, by choosing the inflection point m to be twice the average activity, $\frac{2B}{n}$. The value of s was chosen to be $\frac{4}{m}$, yielding a tangent line at the inflection point that passes through the origin.

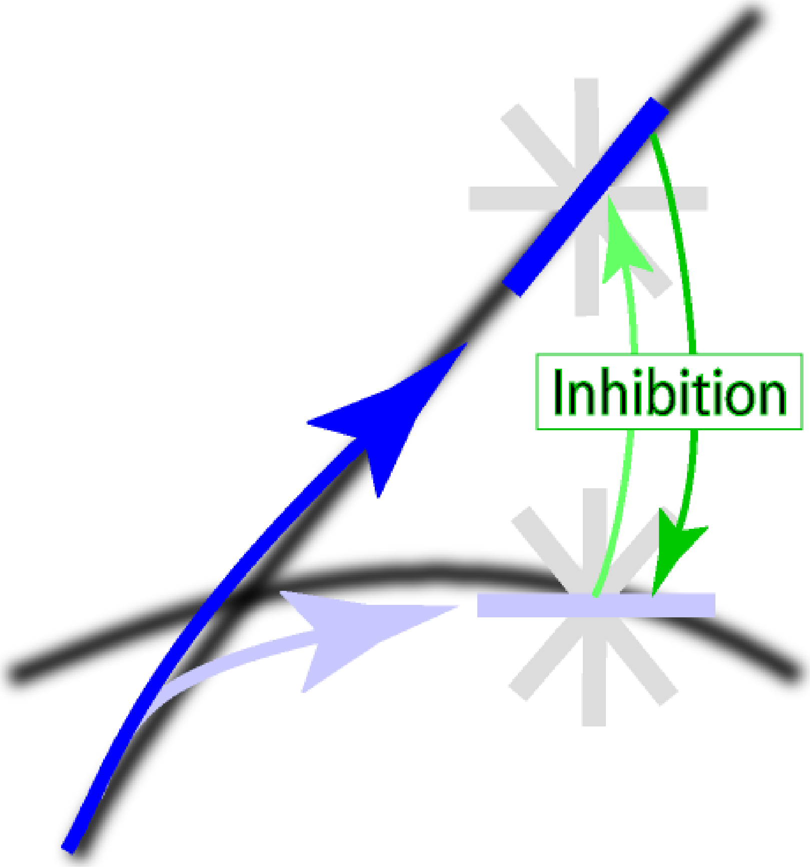


Figure 5-6: *The effect of competitive dynamics [58] on the spread of labeling activity across a crossing. The straighter path (dark blue) suppresses the alternative path (light blue) via inhibition (green) and become dominant.*

In addition to maintaining stability, competitive neural dynamics have an important effect on the spread of labeling activity across edge crossings. Labeling activity reaching the edge crossing proceeds to spread along multiple paths (Figure 5-6). Competitive dynamics causes the activity corresponding to these paths to compete, producing a winner-take-all effect [58]. The path corresponding to the straightest path across the crossing becomes dominant, due to the activity in V1 from aligned edges, and alternative paths become suppressed. At orthogonal crossings, where activity along the alternative paths is minimal, the straightest path quickly suppresses alternative paths and become dominant. At acute crossings, on the other hand, the level of activity along alternative paths is closer to that of the straightest path, requiring more time for neurons corresponding to the straighter path to become dominant via winner-take-all behavior.

5.4 Model Evaluation

The CortexSim 3.0 model was used to evaluate two-layer node-link graph visualizations and its performance was compared with that of human test subjects. For both the model and human test subjects, we test the accuracy and time required to trace out links in node-link diagram visualizations.

5.4.1 Node-link Graph Diagram Generation

The node-link graph visualizations used in this study are two-layer graph diagrams (Figure 5-7). Two-layer graph diagrams provide a building block for more complicated multi-level diagrams, yet restricting the visualizations to two-layer graph diagrams serves to vastly limit the space of visualizations. The graph diagrams are constructed from two parallel rows of five evenly-spaced nodes. The links are defined by cubic Hermite splines (Figure 5-8) drawn to visually represent the edges between corresponding nodes in the top and bottom rows. The cubic Hermite splines are defined by two endpoints and two tangent vectors, one at at each endpoint. The vectors that define the Hermite spline are constrained to orientations within 45° of vertical.

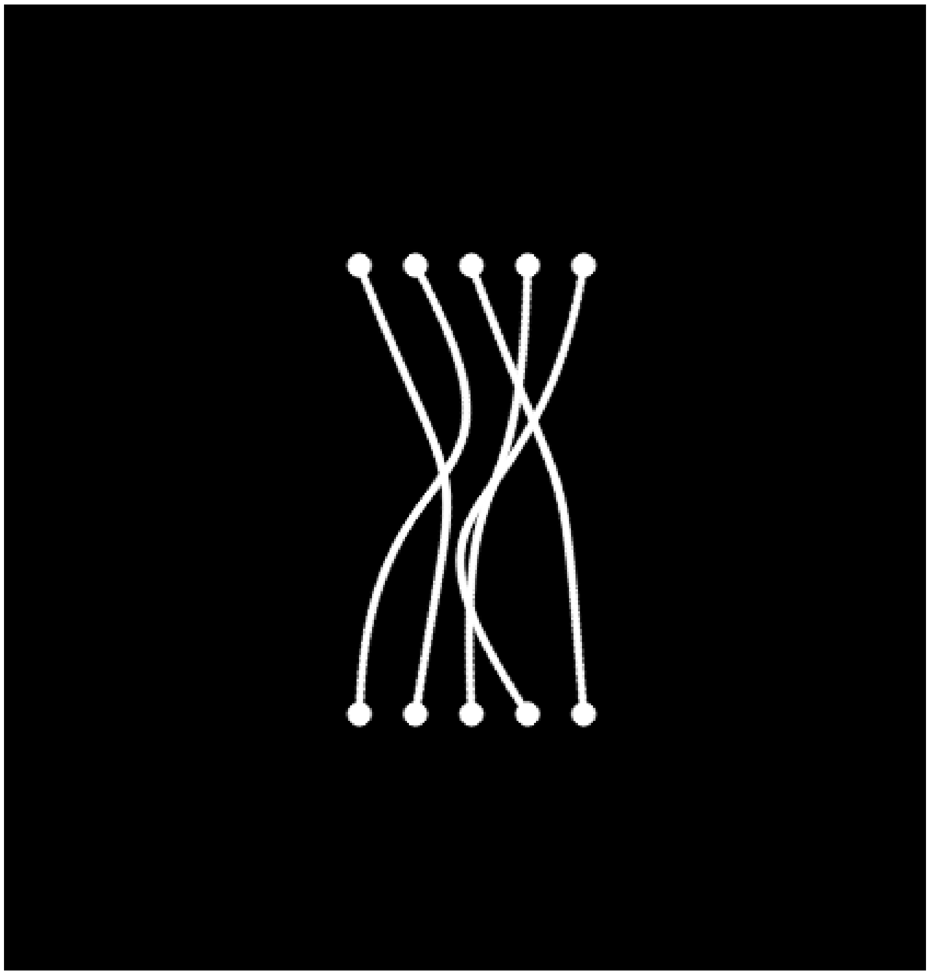


Figure 5-7: *An example of a two-layer node-link graph diagram visualization used for this experiment.*

5.4.2 Graph Data Generation

The graph data describes the connectivity between the upper and lower rows of nodes. The data is defined by a vector of five integers, with each value indicating the index of the top node to which the corresponding bottom node is connected. Random graph data is generated by randomly swapping two adjacent elements in the vector, this is repeated five times. This method of partial randomization was chosen in favor of full randomization, due to an excessive amount of crossings produced by full

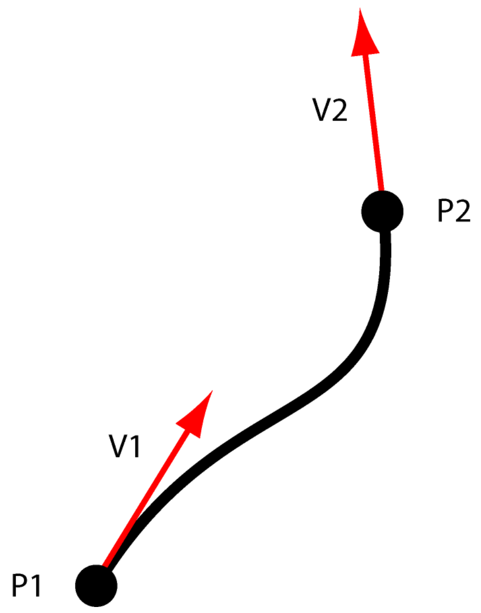


Figure 5-8: *A cubic Hermite spline is defined by two endpoints and two tangent vectors. The endpoint positions correspond to the nodes of the node-link diagram, the tangent vectors define the path of the link between the endpoints.*

randomization.

5.4.3 Instructions

Test subjects were instructed to begin each trial by attending to the node indicated by a red circle. The subject was instructed to indicate whether or not this node was connected to a second node indicated by a blue circle. The subject was told to indicate that the two nodes were connected by pressing the "j" key, and unconnected by pressing the "f" key.

5.4.4 Procedure

The human performance of the node-link graph visualization was measured using a two-alternative, forced choice test. The visualization was presented on a 15.1, 133dpi LCD screen, viewed at a distance of approximately 57 cm. The test begins

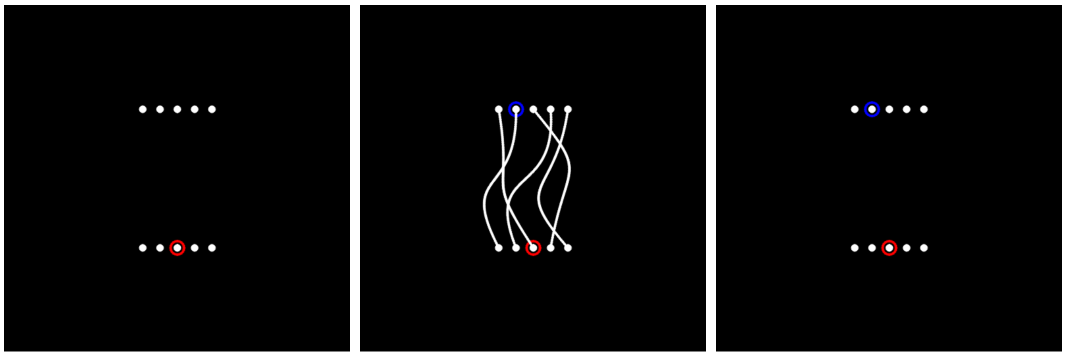


Figure 5-9: *The flow of the graph link tracing test.* ***Left:*** *The nodes are displayed with the bottom starting location circled in red, this is displayed for one second.* ***Center:*** *The links are displayed and an endpoint circled in blue, this is displayed for 300 milliseconds.* ***Right:*** *The links are removed and only the nodes and circles remain, this is displayed until a response is given.*

by presenting the test subject with only the nodes of the visualization, and a random bottom node circled in red. This is presented for one second to allow time for the test subject to focus attention to the circled node. Next, the links are drawn, as well as a blue circle around one of the top nodes. The circled top node is always chosen such that 50% of the time it corresponds to the circled bottom node, 25% of the time it is one to the left, and 25% one to the right. After 300 milliseconds, the links are removed, leaving only the nodes and the top and bottom circles. The test subject was then given as long as needed to indicate whether or not they perceived the circled nodes as connected.

The experiment was performed using 40 node-link graph diagram visualizations. In each block, the 40 visualizations were presented in a random order. Each subject participated in four experiment blocks, for a total of 160 trials for each subject, and a total of 960 for the experiment. Prior to the experiment, each test subject was allowed to practice the task to minimize learning effects.

5.4.5 Participants

Six volunteer test subjects participated in the experiment, including five males and one female.

5.4.6 Computer Trials

The task was also performed using the CortexSim 3.0 computational model. The model was used in four blocks, each containing the 40 visualizations used in the human trials. The computer trial was initiated by setting the neural activity in the label cortex, L(t=0) in Equation 5.1, to one at the neurons with receptive fields corresponding to the starting node, and setting the activity to zero elsewhere. After ten iterations of the model, the feedforward activity of the label cortex is assumed to have reached steady state, and the activity level associated with the "active" state is recorded at the five endpoint nodes. With each iteration, the activity is examined for an increase in activity associated with the labeled state. Neurons with activity that have increased to over 1.5 times the activity of the active state are assumed to have transitioned to the labeled state. When a neuron corresponding to one of the endpoint nodes transitions to the labeled state, the model has completed tracing the contour. The number of iterations and the correctness of the transitioning endpoint are recorded.

5.4.7 Node-link Graph Modeling Results

The mean human time and accuracy performance for each graph was averaged, as was the mean performance of the model, and fit with a least-squares fit linear regression. Despite the highly inconsistent human performance, there was still a significant correlation in the data. The correlation between the human response time and model iteration time of 0.50, which an analysis of variance revealed to be significant

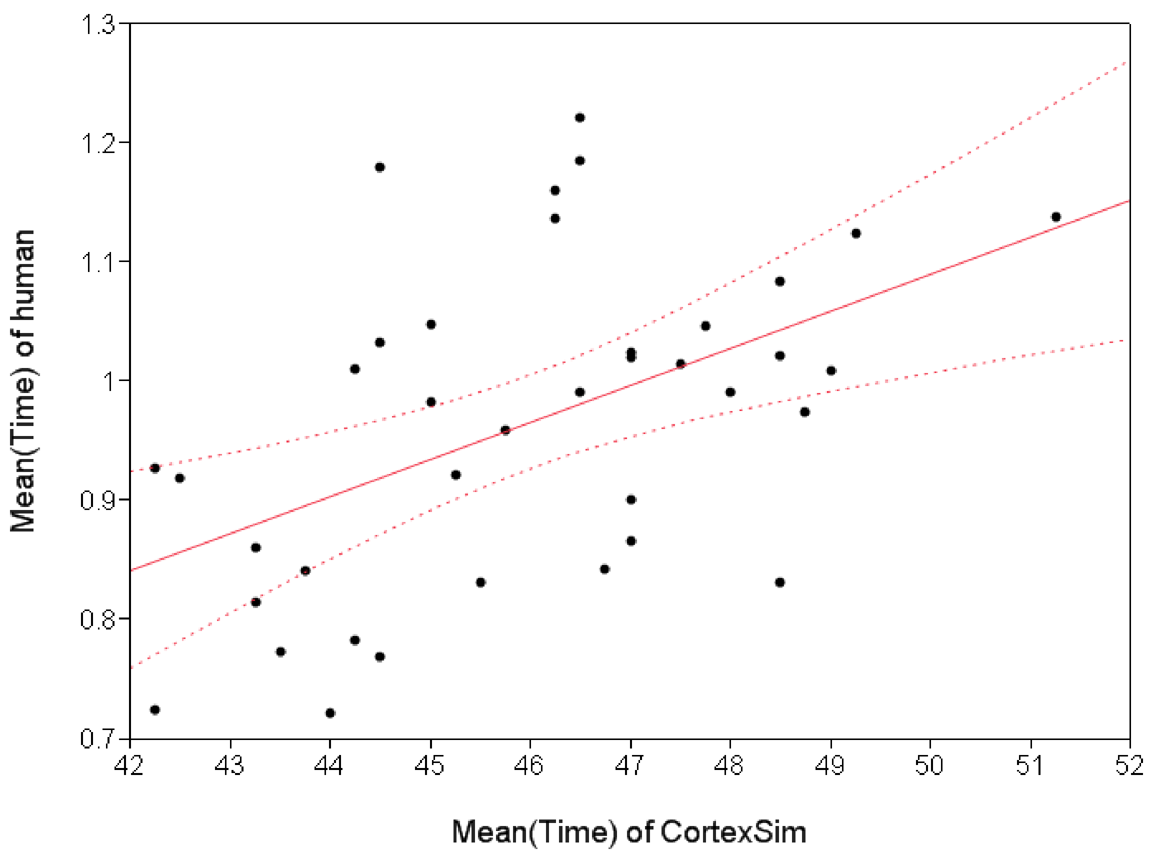


Figure 5-10: *Scatter plot of the number of iterations required by the model to trace the contour, and human response time. The red line indicates the least-squares linear fit, and the dotted lines represent the 95% confidence interval.*

$[F(1, 36) = 11.91, p < .01]$ (Figure 5-10). The correlation between the human and model error rates was found to be 0.37, this positive correlation was also found to be significant $[F(1, 36) = 5.65, p = 0.023]$ by analysis of variance (Figure 5-11). These results indicate that human time and accuracy performance does positively correlate with model performance, and therefore the model is suitable for application as a utility function in an optimization algorithm.

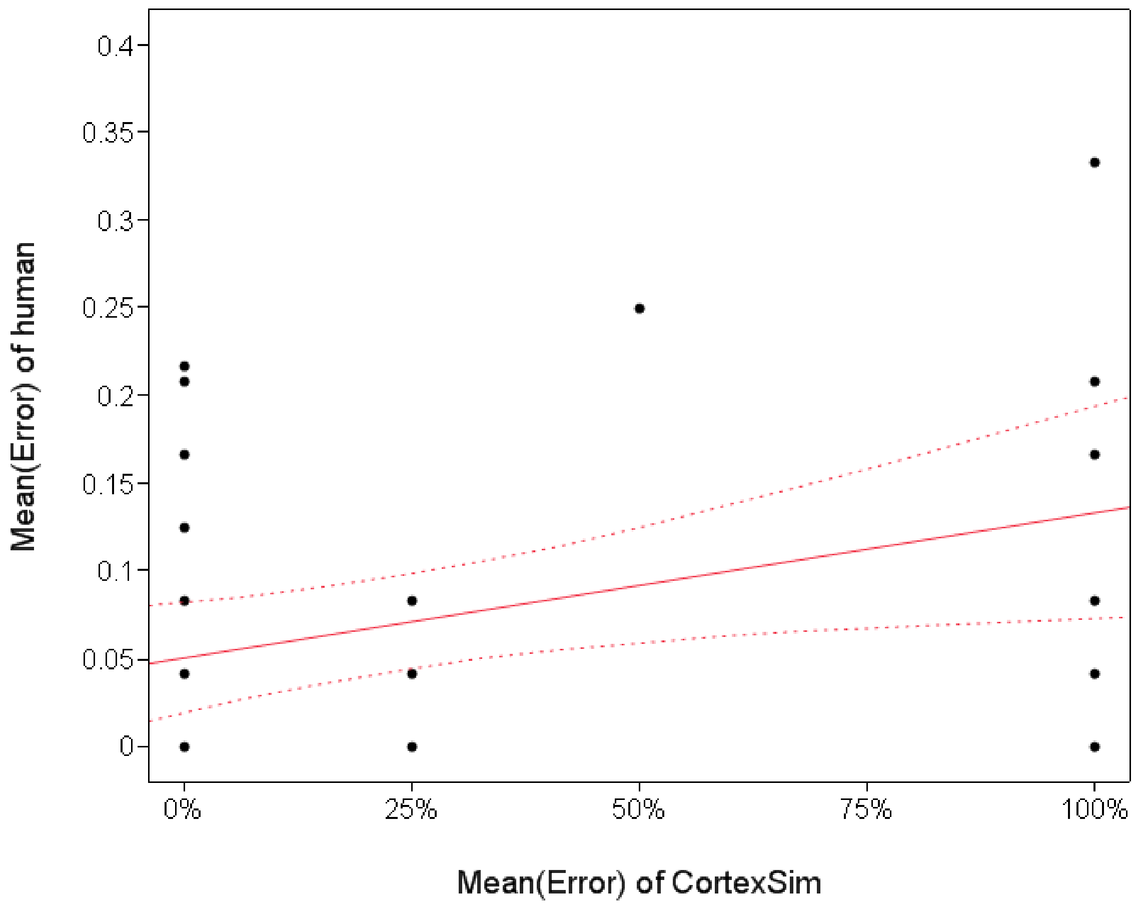


Figure 5-11: *Scatter plot of model error percentage and human error percentage*

5.5 Node-link Graph Diagram Optimization

The node-link graph diagram visualizations were optimized using the stochastic hill-climbing technique described in section 4.3.2. The evaluation function for the optimization algorithm was constructed from the accuracy and number of iterations that CortexSim 3.0 model produces when tracing edges in node-link diagrams. The evaluation contains both a primary and secondary component. The primary component is the number of errors that occur when tracing each of the five links from its starting to ending node. The secondary component is the total number of iterations required to trace all five links. Comparing two evaluations is done by first comparing the primary

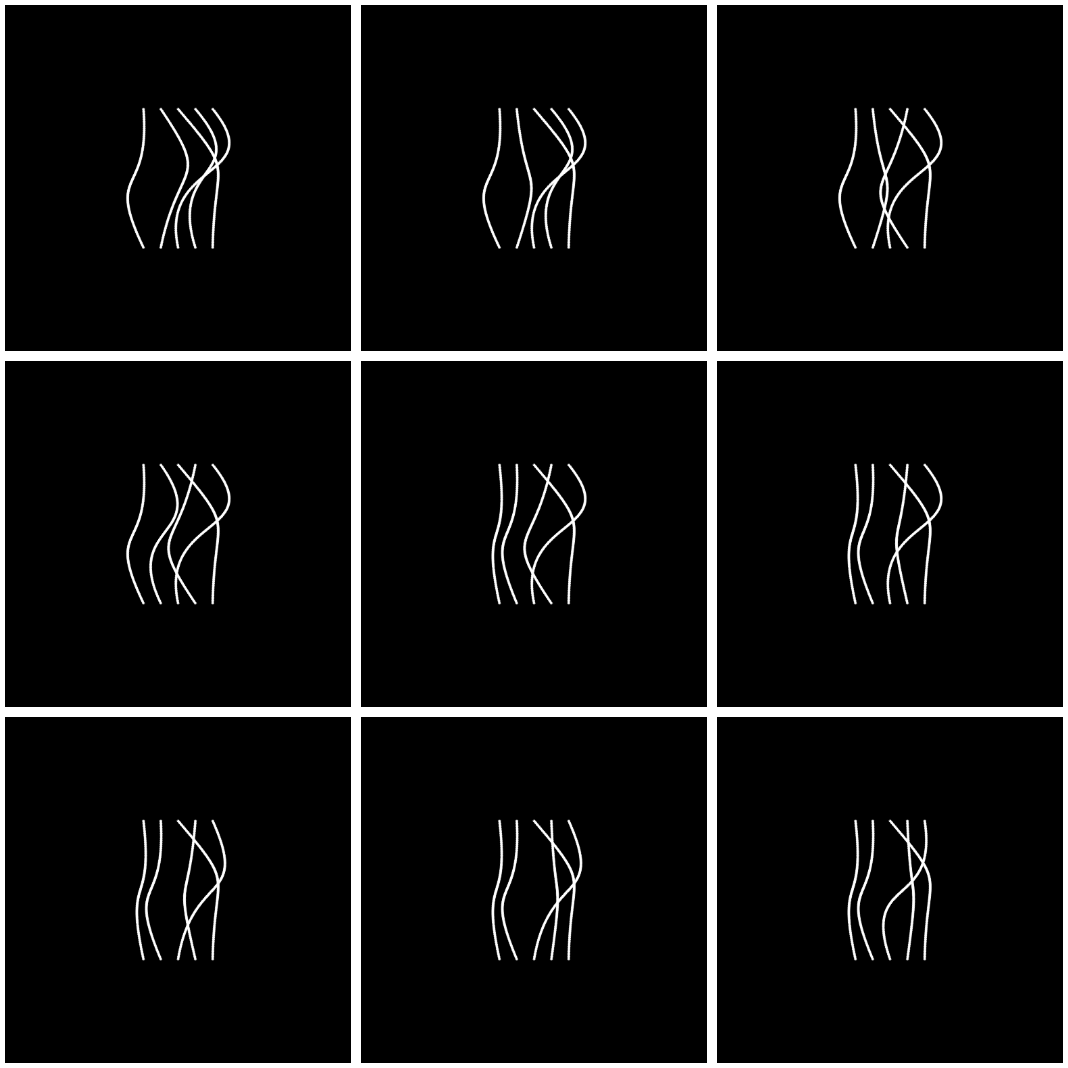


Figure 5-12: *An example of the sequence of graph optimizations performed by the algorithm, progressing from left to right and from top to bottom. Highly acute and ambiguous crossings are eliminated, and edges are straightened.*

components, the visualization that produce the fewer errors has a higher evaluation. If both evaluations produce an equal number of errors, then the visualization that produces that requires the fewest number of total iterations is evaluated higher.

At each iteration of the optimization, one of the five edges was randomly chosen and modified by generating new random vectors defining the Hermite spline for that link, adhering to the constraint that they be within ±45° of vertical. The model was used to reevaluate the visualization, and the change was maintained the modification if it resulted in a higher evaluation, and reverted otherwise. The optimization algorithm was run on each visualization for 100 iterations.

5.5.1 Optimization Evaluation

The psychophysical tracing experiment was repeated using the optimized, as well as the original, node-link graph diagrams. Original and optimized graphs were mixed and presented in a random order within each block, increasing the total number visualizations per block to 80. The results indicate that the optimized node-link graphs yielded an improvement in both the time and accuracy performance. The error rates on the original and optimized graphs were 105 and 61, respectively, representing a decrease in error rate of 44%. A Fisher's 1-tailed test indicates this decrease is significant [$Prob < 0.01$]. The response time performance was also improved by the optimization. The mean response time decreased from 0.92 seconds for the original graphs, to 0.86 seconds for the optimized graphs. A One-way analysis of variance showed that this improvement is significant [$F(1, 1918) = 11.58, p < 0.01$].

5.6 Discussion

The successful application of the CortexSim 3.0 model to node-link graph diagram visualization is evidence that the computational modeling approach presented in this thesis work is generally applicable to the problem of data visualization. We found a correlation between the model and human performance in regards to both response time and accuracy. The correlation was positive and significant, but arguably not very strong. However, given the variance in the between humans results, and even between trials of an individual human, a very strong correlation would be expecting too much.

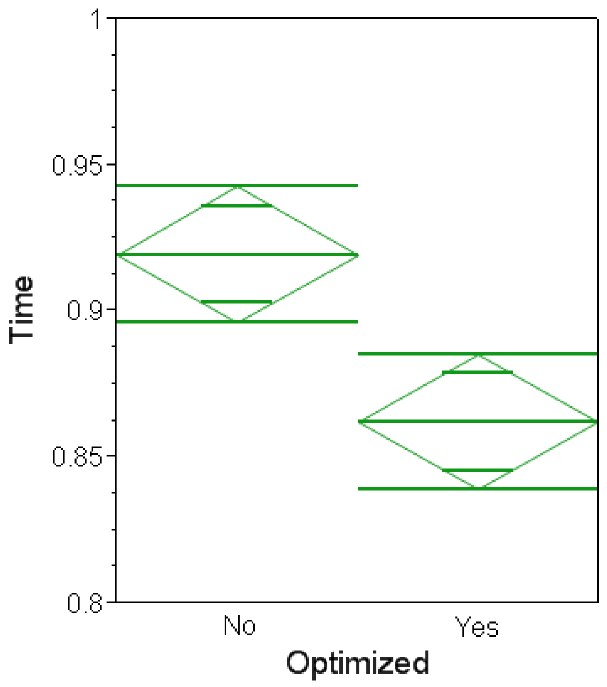


Figure 5-13: *Means diamonds for the response times of the original and optimized node-link graphs. The center of the diamonds indicate the means, and the end bars indicate the 95% confidence interval.*

An unexpected result that emerged from the node-link graph diagram optimization is that it generally did not produce orthogonal crossings, as had been anticipated. Instead, it often created angles of approximately 60°. The reason for this is that that all crossing angles greater than 60° have nearly perfect probability of being traced correctly by the model, and produce a nearly identical delay in tracing. Thus, the secondary evaluation criteria of minimizing the tracing time dominates, which encourages the links to be as short and direct as possible. Crossing angles of 60° are often better able to achieve this secondary criteria than crossings of 90°.

The quality of the graph diagrams was likely impaired by the compute time requirements of the algorithm. Each link required approximately 50 iterations of the model, and each node-link graph diagram entailed tracing five links, for a total of approximately 250 model iterations per evaluation. This can be contrasted with the flow visualization evaluation performed in chapter 4, which required only a single model iteration, and furthermore did not require the computation of the label cortex. As a result of the evaluation cost, only 100 iterations of the optimization loop were performed on each node-link graph visualization. This was sufficient to produce clear improvements over randomly laid out graph diagrams, but there is likely room for additional improvement by allowing the optimization to continue beyond 100 iterations.

The computational cost could be addressed with a more sophisticated optimization algorithm. In this work, we used a simple stochastic hillclimbing algorithm, but improvements could likely be made by incorporating node-link graph diagram aesthetics as search heuristics. Good heuristics could be used to generate modifications to the visualization that are closer to a search optima, thus causing the algorithm to first search diagrams that are more likely to yield an improved evaluation. While the search algorithm was not the focus of this research, to use this optimization approach in an applied setting, the search algorithm and resulting runtime will be of high importance.

CHAPTER 6

CONCLUSION

This work began with the observation that computational modeling has had a profound effect on the scientific disciplines to which it has been successfully applied, and recognizing that by using a neural model of human vision, the problem of data visualization may be treated computationally. In this work, we investigated the application of computational modeling to data visualization and explored some of the opportunities that such an approach presents. Reviewing the neural architecture of the visual system, we recognized the suitability of a computational modeling approach to the problem of data visualization. Neurons present themselves as convenient computational building blocks of the visual system. These building blocks are individually simple, responding to center-surround patterns in the retina and Gabor patterns in V1, but within a large population produce the mechanisms that are vital to the perception of data visualizations. The CortexSim model of the visual system was developed to simulate these perceptual mechanisms so their application to data visualization design could be explored.

The implementation of the CortexSim model evolved over the course of this work, however, the general design can be decomposed into two major parts: a low-level neural network model of the early stages of the visual system, and a model of the high-level task-related perceptual processing of the later stages of the visual system. This suggests a general model design for the method used in this work, whereby the early visual processing is modeled with a neural network, and the resulting the

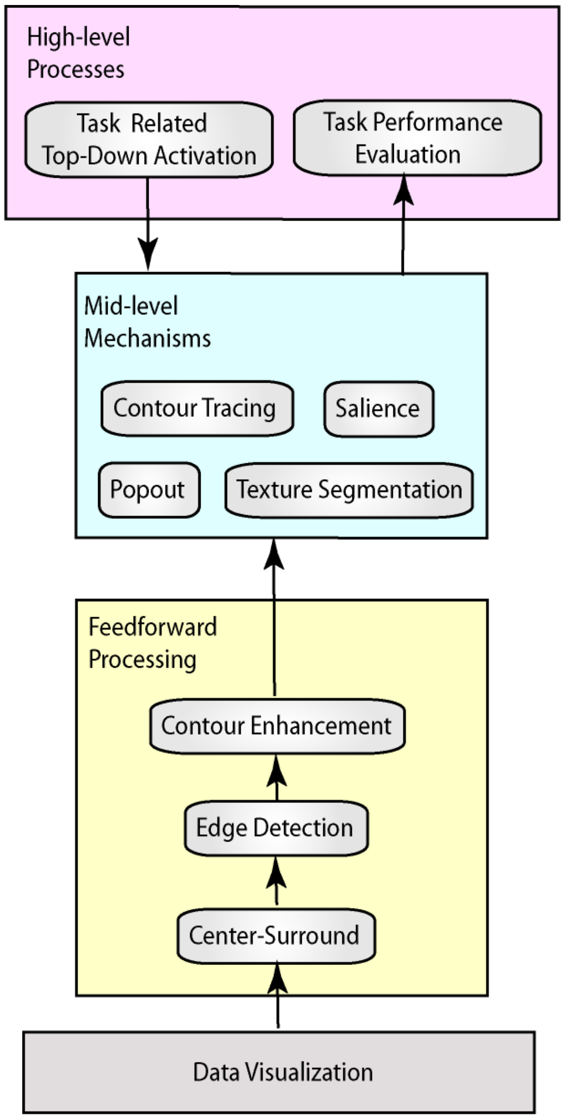


Figure 6-1: *The general design of the CortexSim model. The visual image is first processed by several stages of feedforward processing, followed by the mid-level perceptual mechanisms, which are influenced by top-down activation. The resulting neural network activity is analyzed to derive a measure of visualization effectiveness.*

activity output this neural network is analyzed by subsequent higher-level perceptual modules (See figure 6-1).

The first major component of the design, the low-level neural network, simulated the early, feedforward visual processing of the human visual system. This included the center-surround processing of the retina, the edge detection of V1, as well as the contour enhancement produced by V1 lateral connections. It has been proposed that early visual processing mechanisms such as these are responsible for the effectiveness of data visualizations [17]. Incorporating these mechanisms into a model of the visual system enabled us to test this proposition by simulating how data visualizations trigger these perceptual mechanisms.

We focused in this work on the perceptual mechanisms of edge detection and contour perception for two reasons. First, the neural architecture of the human visual system that produces these mechanisms is sufficiently understood to be implemented into a computational model. Second, edge and contour perception is vital to many important forms of data visualization, in particular the flow visualizations and node-link graph diagrams used in this work. However, there remains many other perceptual mechanisms of importance to data visualization perception that could possibly be added to the neural model. For example, the perceptual mechanisms of visual salience, popout, and symbol distinctiveness, are all of importance to data visualization and could likely also be incorporated into the neural model. These mechanisms are important for the perception of map and chart visualizations, and incorporating these mechanisms into the model would enable it to be used to simulate these visualization types. Some visualizations use highly-packed symbols, which viewers perceive as textures [16]. By incorporating the neural mechanisms of texture segmentation into the model, it could also be used to analyze such texture-based visualizations as well.

The second major component of the design is the model of the high-level task-related perceptual analysis. Because the neural processing involved in performing visual tasks is not understood at the neural level, these were built as a layer on top of the low-level neural network model. The behavior of the neural network provides the input to this high-level model of the analytical task. Because this high-level model is task specific, we have developed high-level models for a number of specific analytical tasks.

The first high-level task model, implemented in CortexSim 1.0, models the task of cognitively tracing an advection pathway. This model evaluates the activity of the edge and contour sensitive neurons in V1, and performs an operation to trace an advection pathway based on this V1 activity. This operation involved estimating the flow direction from local neural activity, and iteratively tracing along the path based on the estimated flow direction. Together with the low-level neural network model of vision, the model was used to simulate the advection pathway tracing task on a variety of flow visualizations.

Modeling the second task, estimating local orientation of the flow field at every point in the flow field, was done in CortexSim 2.0. While clearly not a task that could be reasonably asked of a human due to the large number of possible points, humans are often called upon to estimate flow direction at some arbitrary point in a flow visualization, for example, to estimate the advection pathway of a particle. Therefore, this second task can be considered simply performing all possible cases of estimating flow direction at an arbitrary point. The high-level task model was layered upon the low-level neural network model. The model of this task is based on a simple and reasonable assumption: the perceived orientation is related directly to the local activity of the orientation sensitive V1 neurons.

The third task involved tracing the links of a node-link graph diagram, and was modeled by implementing the curve tracing operator described by Rolfsema [68]. This operator is based on the idea that attention spreads across the lateral connections of V1 neurons that have become active due to a contour in the visual field. In CortexSim 3.0, top-down activation produced initial neural activity, corresponding to visual attention, in a cortical area referred to as the “label cortex”. The attention spreads by activating neurons in V1, where the activity travels along lateral connections, before returning back to the label cortex.

Unlike the previous implementations of high-level task, attention spreading was implemented using a neural network. But unlike the feedforward processing of the retina and the edge detection and contour enhancement mechanisms of V1, the attention spreading mechanism was driven by top-down, in addition to bottom-up, activation. The top-down activation provides the neural activity corresponding to the initial point of visual attention. Thus, it might be more appropriate to consider attention spreading to be a mid-level perceptual mechanism, rather than a high-level analytical task. Visual salience, popout, and texture segregation, which depend highly on top-down activation, could similarly be considered mid-level perceptual mechanism. These mechanisms, because they can be guided by top-down factors, are often targeted by data visualizations to convey information. In future development of the CortexSim approach, implementing these kinds of mid-level mechanisms could prove essential for modeling data visualization task performance.

Due to the vast complexity of the human visual system, it is unreasonable to attempt to model it precisely at this time. The CortexSim models used in this work differs from the human visual system in a number of ways. Firstly, the CortexSim model includes only the early processing areas of the visual system. Other visual areas outside V1 are known to have an effect on visual perception, but are not well understood. As these visual processing mechanisms become better understood, they can be incorporated into the model, yielding a more accurate simulation of visual perception.

Additional inaccuracies of the model stem from our incomplete understanding of how the human visual system works. For example, the spiking behavior of neurons tends to contain oscillations, commonly known as brain waves. The origin and purpose of these waves continue to be debated, and their relationship to visual perception, if any, is unknown. Without a clear understanding of these waves, they cannot not be successfully incorporated into a computational model.

Furthermore, neuron locations in V1 were idealized to lie on a hexagonal grid in CortexSim 1.0, and on regular grids in CortexSim 2.0 and CortexSim 3.0, and neuron spiking behavior was approximated with a spiking rate. These idealizations were necessary to keep the complexity and run time of the model manageable. The fidelity of the model to true human neurophysiology is limited by available computing hardware and by our state of knowledge of the visual system. As more computing power becomes available in the future, it can be applied to making a more physiologically accurate model.

Despite its limitations, the CortexSim model proved to be successful at modeling human task performance using two very distinct visualization types: flow visualizations and node-link graph diagrams. For the flow visualizations, we found that the model produced the same ordering of effectiveness that humans did for four visualization techniques. For the node-link graph visualizations, we found a positive correlation between the model iteration time and human response time. It is not clear whether this speaks to the robustness of the approach, or if even better results are possible with a more physiologically accurate model. It may be that both explanations are true. However, it is clear that with advances in our knowledge of the physiology of the visual system and improvements in computational capabilities, there is potential to improve the model, and likely the results as well.

Furthermore, the model predictions were adequate for use in optimization routines. The optimization of visualizations proves the utility of the modeling approach by demonstrating a tangible contribution to the problem of data visualization. The similarities between the optimized visualizations and visualizations developed by human designers is notable. It is an indication that perceptual theory, and by extension good data visualization design, is not based on opinion, but rather that there are physical reasons underlying the effectiveness of high-quality data visualizations.

The neural network portion of the model was largely reused with each version of CortexSim. The CortexSim 3.0 version of the neural model used for the node-link diagram evaluation and optimization was essentially the same as the CortexSim 2.0 version previously used for the flow optimization, with the exception of the added label cortex. This demonstrates the reusability of a computational neural model. This reuse of a core model is analogous to how a computational fluids dynamics model can be applied to vastly differing designs, for example an aircraft and automobile design. Tn contrast, the traditional approach of applying perceptual theory to data visualization typically requires a new theory to be developed to describe each perceptual phenomenon. As the CortexSim model is further developed and made to more accurately model the human visual system, it is likely that there will be even less need to make significant modifications to evaluate novel visualization designs.

All of the optimization techniques employed in this work optimized the location of the low-level graphical primitives within a visualization. A potential alternative to this approach is to use the model to generate and optimize the aesthetic parameters used in a traditional aesthetics-based layout algorithm. For example, the aesthetics-based algorithm could layout diagrams based on a set of parameters. Then, the model could be used to evaluate the diagrams produced using those parameters, thus evaluating the parameters used to construct the visualization. The algorithm would search for optimal parameters for constructing visualizations, rather than searching for a single optimal visualization. Such an approach could yield a layout algorithm that is optimized based on the perceptual model, but has the fast run time characteristics of an aesthetics-based algorithm.

In addition to being used as a practical tool, a computational model of perception, even one that is imperfect, can provide an invaluable tool for examining perceptual theories of data visualization in a new way. It has been conjectured that the effectiveness of popular flow visualization techniques, such as the LIC and Jobard & Lefer algorithms, are due to the contour processing mechanisms of the visual system. Now with a computational model, we can begin to examine this conjecture from a new, computational point of view. For instance, we find that the flow visualizations created by these popular techniques resemble visualizations optimized using the computational model for the flow orientation task. We find that by complementing perceptual theory and psychophysical experimentation with the "third pillar" of computational modeling, we are able to provide strong support for this conjecture.

The success of computational models in other disciplines, such as computational fluid dynamics, was the result of years of development many researchers. With further development, computational modeling could become as vital to a data visualization practitioner as computational fluid dynamics models are to aircraft designers. Perhaps the most significant contribution of this work is to justify and motivate further research in applying computational models to problems in data visualization.

BIBLIOGRAPHY

[1] D. H. Laidlaw, J. S. Davidson, T. S. Miller, M. da Silva, R. M. Kirby, W. H. Warren, and M. Tarr, “Quantitative comparative evaluation of 2d vector field visualization methods,” in *VIS ’01: Proceedings of the conference on Visualization ’01.* Washington, DC, USA: IEEE Computer Society, 2001, pp. 143–150.

[2] C. Ware, *Information Visualization: Perception for Design, 2nd Ed.* Morgan Kaufman, 2004.

[3] D. J. Field, A. Hayes, and R. F. Hess, “Contour integration by the human visual system: Evidence for a local ”association field”,” *Vision Research*, vol. 33, no. 2, pp. 173 – 193, 1993.

[4] B. Cabral and L. C. Leedom, “Imaging vector fields using line integral convolution,” in *SIGGRAPH ’93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques.* New York, NY, USA: ACM, 1993, pp. 263–270. [Online]. Available: http://dx.doi.org/10.1145/166117.166151

[5] B. Jobard and W. Lefer, “Creating evenly-spaced streamlines of arbitrary density,” in *Eurographics Workshop.* Springer Verlag, 1997, pp. 43–56.

[6] G. Turk and D. Banks, “Image-guided streamline placement,” in *SIGGRAPH ’96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques.* New York, NY, USA: ACM, 1996, pp. 453–460.

[7] J. J. van Wijk, "Spot noise texture synthesis for data visualization," in *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques.* New York, NY, USA: ACM, 1991, pp. 309–318.

[8] President's Information Technology Advisory Committee, "Computational Science: Ensuring America's Competitiveness," *Computational Science Subcommittee (PITAC), National Coordination Office for Information Technology Research and Development*, p. 117, 2005.

[9] J. F. Wendt and J. D. Anderson, *Computational Fluid Dynamics: An Introduction*, J. F. Wendt, Ed. Springer, 2009.

[10] D. J. Freeman, D. Reubush, C. R. McClinton, V. L. Rausch, and J. L. Crawford, "The nasa hyper-x program," Tech. Rep., 1997.

[11] J. Dubinski, "The great milky way -andromeda collision," *Sky and Telescope*, vol. 112, no. 10, p. 30, October 2006.

[12] C. Ware, "Color sequences for univariate maps: theory, experiments and principles," *IEEE Computer Graphics and Applications*, vol. 8, no. 5, pp. 41–49, September 1988.

[13] H. C. Purchase, "The effects of graph layout," in *Computer Human Interaction Conference, 1998. Proceedings. 1998 Australasian*, Nov-4 Dec 1998, pp. 80–86.

[14] ——, "Performance of layout algorithms: Comprehension, not computation," *Journal of Visual Languages & Computing*, vol. 9, no. 6, pp. 647 – 657, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/B6WMM-45J5BKS-5/2/f2d27b9e887b63c9f3247c933f4a9bf2

[15] C. Healey, K. Booth, and J. Enns, "Harnessing preattentive processes for multivariate data visualization," pp. 107–107, 1993.

[16] C. G. Healey and J. T. Enns, “Building perceptual textures to visualize multidimensional datasets,” in *VIS ’98: Proceedings of the conference on Visualization ’98*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1998, pp. 111–118.

[17] C. Ware, “Toward a perceptual theory of flow visualization,” *IEEE Comput. Graph. Appl.*, vol. 28, no. 2, pp. 6–11, 2008.

[18] H. C. Purchase, D. A. Carrington, and J.-A. Allder, “Experimenting with aesthetics-based graph layout,” in *Diagrams ’00: Proceedings of the First International Conference on Theory and Application of Diagrams*. London, UK: Springer-Verlag, 2000, pp. 498–501.

[19] H. C. Purchase, “Metrics for graph drawing aesthetics,” *Journal of Visual Languages & Computing*, vol. 13, no. 5, pp. 501 – 516, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/B6WMM-46YHVYF-3/2/41eddd36d6b619b866c108569cfca607

[20] A. M. Treisman and G. Gelade, “A feature-integration theory of attention,” *Cognitive Psychology*, vol. 12, no. 1, pp. 97 – 136, 1980. [Online]. Available: http://www.sciencedirect.com/science/article/B6WCR-4D6RJM2-46/2/b34aa05384b2a7702189c22840489174

[21] J. M. Wolfe, K. R. Cave, and S. L. Franzel, “Guided search: An alternative to the feature integration model for visual search,” *Journal of Experimental Psychology: Human Perception and Performance*, vol. 15, no. 3, pp. 419 – 433, 1989. [Online]. Available: http://www.sciencedirect.com/science/article/B6X08-46X8WS6-1/2/501dfc15c55b7f804b0900725ef0d027

[22] L. Itti and C. Koch, "Computational modelling of visual attention." *Nat Rev Neurosci*, vol. 2, no. 3, pp. 194–203, March 2001. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/11256080

[23] S. Fabrikant and K. Goldsberry, "Thematic relevance and perceptual salience of dynamic geovisualization displays," in *Proceedings of 22nd ICA international cartographic conference: mapping approaches into a changing world*, July 2005.

[24] S. Garlandini and S. Fabrikant, "Evaluating the effectiveness and efficiency of visual variables for geographic information visualization," *Spatial Information Theory*, pp. 195–211.

[25] S. Fabrikant, S. Hespanha, and M. Hegarty, "Cognitively inspired and perceptually salient graphic displays for efficient spatial inference making," *Annals of the Association of American Geographers*, vol. 100, no. 1, pp. 13–29, January 2010.

[26] R. Rosenholtz, Y. Li, and L. Nakano, "Measuring visual clutter," *Journal of Vision*, vol. 7, no. 2, pp. 1–22, 2007. [Online]. Available: http://journalofvision.org/7/2/17/

[27] N. A. Carlson, *Foundations of Physiological Psychology.* Needham Heights, Massachusetts: Simon & Schuster, 1992.

[28] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve." *J Physiol*, vol. 117, no. 4, pp. 500–544, August 1952. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/12991237

[29] W. Mcculloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[30] K. Naka, "Neuronal circuitry in the catfish retina," *Invest. Ophthalmol*, vol. 15, no. 11, pp. 926–935, 1976.

[31] S. Polyak, *The retina.* The University of Chicago Press, 1941.

[32] S. Schein, "Anatomy of macaque fovea and spatial densities of neurons in foveal representation," *The Journal of Comparative Neurology*, vol. 269, no. 4, pp. 479–505, 1988.

[33] K. Foster, J. Gaska, M. Nagler, and D. Pollen, "Spatial and temporal frequency selectivity of neurones in visual cortical areas v1 and v2 of the macaque monkey." *The Journal of physiology*, vol. 365, no. 1, p. 331, 1985.

[34] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J Physiol*, vol. 160, no. 1, pp. 106–154, 1962. [Online]. Available: http://jp.physoc.org

[35] ——, "Receptive fields and functional architecture of monkey striate cortex," *J Physiol*, vol. 195, no. 1, pp. 215–243, 1968. [Online]. Available: http://jp.physoc.org/cgi/content/abstract/195/1/215

[36] V. Mountcastle, "The columnar organization of the neocortex," *Brain*, vol. 120, no. 4, pp. 701–722, 1997. [Online]. Available: http://brain.oxfordjournals.org/cgi/content/abstract/120/4/701

[37] W. H. Bosking, Y. Zhang, B. Schofield, and D. Fitzpatrick, "Orientation selectivity and the arrangement of horizontal connections in tree shrew striate cortex," *J. Neurosci.*, vol. 17, no. 6, pp. 2112–2127, 1997. [Online]. Available: http://www.jneurosci.org/cgi/content/abstract/17/6/2112

[38] R. Rodieck, "Quantitative analysis of cat retinal ganglion cell response to visual stimuli," *Vision Research*, vol. 5, no. 12, pp. 583–601, 1965.

[39] J. G. Daugman, "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters," *Journal of the Optical Society of America A: Optics, Image Science, and Vision*, vol. 2, no. 7, pp. 1160–1169, 1985.

[40] Z. Li, "A neural model of contour integration in the primary visual cortex," *Neural Comput.*, vol. 10, no. 4, pp. 903–940, 1998.

[41] S. Grossberg and J. Williamson, "A neural model of how horizontal and interlaminar connections of visual cortex develop into adult circuits that carry out perceptual grouping and learning," *Cerebral Cortex*, vol. 11, no. 1, p. 37, 2001.

[42] G. Carpenter and S. Grossberg, "The art of adaptive pattern recognition by a self-organizing neural network," *Computer*, vol. 21, pp. 77–88, 1988.

[43] S. Grossberg, E. Mingolla, and W. D. Ross, "Visual brain and visual perception: How does the cortex do perceptual grouping?" vol. 20, 1997, pp. 106–111.

[44] S. Grossberg, "How does the cerebral cortex work? learning attention, and grouping by the laminar circuits of visual cortex," *Spatial Vision*, vol. 12, pp. 163–186, 1999.

[45] S. Grossberg and L. Pessoa, "Texture segregation, surface representation and figure-ground separation," *Vision Research*, vol. 38, no. 17, pp. 2657 – 2684, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/B6T0W-3VBWFR9-Y/2/6b28e12231482c4ef350d4b4afe67237

[46] S. Grossberg, L. Kuhlmann, and E. Mingolla, "A neural model of 3d shape-from-texture: Multiple-scale filtering, boundary grouping, and surface filling-in," *Vision Research*, vol. 5, pp. 634–672, 2007.

[47] M. D. Fairchild, *Color appearance models, Second Edition.* John wiley & Sons, Ltd, 2005.

[48] Z. Li, "Pre-attentive segmentation in the primary visual cortex," Cambridge, MA, USA, Tech. Rep., 1998.

[49] P. Parent and S. Zucker, "Trace inference, curvature consistency, and curve detection," vol. 11, no. 8, August 1989, pp. 823–839.

[50] N. Lund, *Attention and Pattern Recognition.* Routledge, 2001.

[51] D. Pineo and C. Ware, "Neural modeling of flow rendering effectiveness," in *APGV '08: Proceedings of the 5th symposium on Applied perception in graphics and visualization.* New York, NY, USA: ACM, 2008, pp. 171–178.

[52] H. Hauser, R. S. Laramee, H. Doleisch, F. H. Post, and B. Vrolijk, "The state of the art in flow visualization, part 1: Direct, texture-based, and geometric techniques," *Computer Graphics Forum*, vol. 23, pp. 203–221, 2004.

[53] D. Fowler and C. Ware, "Strokes for representing univariate vector field maps," in *Graphics Interface '89*, June 1989, pp. 249–253.

[54] H. Wilson, D. McFarlane, and G. Phillips, "Spatial frequency tuning of orientation selective units estimated by oblique masking," *Vision Research*, vol. 23, no. 9, pp. 873–882, 1983.

[55] . D. G. Rolls, E. T., *Computational neuroscience of vision.* Oxford University Press, 2002.

[56] C. Ware, "3d contour perception for flow visualization," in *APGV '06: Proceedings of the 3rd symposium on Applied perception in graphics and visualization.* New York, NY, USA: ACM, 2006, pp. 101–106.

[57] H. Rushmeier, H. Barrett, P. Rheingans, S. Uselton, and A. Watson, “Perceptual measures for effective visualizations,” in *Proceedings of the 8th conference on Visualization’97*. IEEE Computer Society Press, 1997, pp. 515–517.

[58] S. Grossberg, “Contour enhancement, short-term memory, and constancies in reverberating neural networks,” *Studies in Applied Mathematics*, vol. 52, pp. 213–257, 1973.

[59] D. Marr, E. Hildreth, and T. Poggio, “Evidence for a fifth, smaller channel in early human vision,” 1979.

[60] C. Wickens, J. Lee, Y. Liu, and S. Becker, *An introduction to human factors engineering*. Pearson Prentice Hall Upper Saddle River, NJ, 2004.

[61] D. Holten and J. J. van Wijk, “A user study on visualizing directed edges in graphs,” in *CHI ’09: Proceedings of the 27th international conference on Human factors in computing systems*. New York, NY, USA: ACM, 2009, pp. 2299–2308.

[62] G. Di Battista, P. Eades, R. Tamassia, and I. Tollis, *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1998.

[63] H. C. Purchase, “Which aesthetic has the greatest effect on human understanding?” in *GD ’97: Proceedings of the 5th International Symposium on Graph Drawing*. London, UK: Springer-Verlag, 1997, pp. 248–261.

[64] D. Eppstein, “Concept lattice.svg,” October 2006. [Online]. Available: http://en.wikipedia.org/wiki/File:Concept_lattice.svg

[65] K. Sugiyama, S. Tagawa, and M. Toda, “Methods for visual understanding of hierarchical system structures,” *IEEE TRANS. SYS. MAN, AND CYBER.*, vol. 11, no. 2, pp. 109–125, 1981.

[66] E. Gansner, S. North, and K. Vo, "Technique for drawing directed graphs," Aug. 28 1990, uS Patent 4,953,106.

[67] S. North and G. Woodhull, "Online hierarchical graph drawing," *Lecture notes in computer science*, pp. 232–246, 2002.

[68] P. Roelfsema, V. Lamme, and H. Spekreijse, "The implementation of visual routines," *Vision Research*, vol. 40, no. 10-12, pp. 1385–1411, 2000.

[69] S. Grossberg, "Why do cells compete? Some examples from visual perception," *The UMAP Journal, III*, pp. 103–121, 1982.

[70] D. Kernell, "The adaptation and the relation between discharge frequency and current strength of cat lumbosacral motoneurones stimulated by long-lasting injected currents," *Acta Physiologica Scandinavica*, vol. 65, no. 1-2, pp. 65–73, 1965.

APPENDICES

APPENDIX A

CORTEXSIM 2.0 GPU KERNELS

```
/*------------------- RetinaKernel ------------------------
    input: Array of CIEL*a*b* values of the visual scene
    weights: Difference-of-Gaussians kernel
    output: Center-surround receptive field response
----------------------------------------------------------*/
__global__
void RetinaKernel(uchar3 *input, float1 *weights, float1 *output) {
    int i = (((blockIdx.y)*16+threadIdx.y+8)
        *(grid_diameter/16)+blockIdx.x)*16 +threadIdx.x+8;

    __shared__ float w[16][16];
    __shared__ uchar4 in[2*16][2*16];
    in[threadIdx.y][threadIdx.x] = input[i-8*grid_diameter-8];
    in[threadIdx.y][threadIdx.x+16] = input[i-8*grid_diameter+8];
    in[threadIdx.y+16][threadIdx.x] = input[i+8*grid_diameter-8];
    in[threadIdx.y+16][threadIdx.x+16] = input[i+8*grid_diameter+8];
    w[threadIdx.y][threadIdx.x] = weights[threadIdx.y*16
        + threadIdx.x].x;
    __syncthreads();

    float3 p = make_float3(0,0,0);
    for (int dy=0; dy<16; dy++) {
        for (int dx=0; dx<16; dx++) {
            p += w[dy][dx] * in[threadIdx.y+dy][threadIdx.x+dx];
        }
    }
    output[i].x = p.x/256.0;                // white-black
    output[grid_area + i].x = p.y/256.0;    // red-green
    output[2*grid_area + i].x = p.z/256.0;  // yellow-blue
}


/*-------------------- EdgeKernel ------------------------
    input: Center-surround receptive field response
    weights: V1 Gabor kernels
    output: Edge-sensitive receptive field response
---------------------------------------------------------*/
__global__
void EdgeKernel(float1* input, float1* weights, float1* output) {
    int i = (((blockIdx.y)*16+threadIdx.y+8)
        *(grid_diameter/16)+blockIdx.x)*16 +threadIdx.x+8;

    __shared__ float w[16][16];
    __shared__ float in[2*16][2*16];
    in[threadIdx.y][threadIdx.x] = input[i-8*grid_diameter-8].x;
    in[threadIdx.y][threadIdx.x+16] = input[i-8*grid_diameter+8].x;
    in[threadIdx.y+16][threadIdx.x] = input[i+8*grid_diameter-8].x;
    in[threadIdx.y+16][threadIdx.x+16] = input[i+8*grid_diameter+8].x;
    __syncthreads();

    for ( int p=0; p<12; p++ ) {
        w[threadIdx.y][threadIdx.x] = weights[p*16*16
                + threadIdx.y*16 + threadIdx.x].x;
        __syncthreads();

        float sum=0;
        for (int dy=0; dy<16; dy++) {
            for (int dx=0; dx<16; dx++) {
                sum += in[threadIdx.y+dy][threadIdx.x+dx] * w[dy][dx];
            }
        }
        output[p*grid_area + i].x = abs(sum);
        __syncthreads();
    }
}


/*---------------- EnhancementKernel --------------------
    input: Edge-sensitive receptive field  response
    weights: V1 enhancement kernels
    output: Enhanced edge-sensitive receptive field response
---------------------------------------------------------*/
__global__
void EnhancementKernel(float1* input, float1* output, float1* weights) {
    int i = (((blockIdx.y)*16+threadIdx.y+8)
        *(grid_diameter/16)+blockIdx.x)*16 +threadIdx.x+8;

    float sum;
    for ( int p=0; p<12; p++ ) {
        __shared__ float w[16][16];
        __shared__ float in[2*16][2*16];
        in[threadIdx.y][threadIdx.x] =
                input[p*grid_area + i-8*grid_diameter-8].x;
        in[threadIdx.y][threadIdx.x+16] =
                input[p*grid_area + i-8*grid_diameter+8].x;
        in[threadIdx.y+16][threadIdx.x] =
                input[p*grid_area + i+8*grid_diameter-8].x;
        in[threadIdx.y+16][threadIdx.x+16] =
                input[p*grid_area + i+8*grid_diameter+8].x;
        w[threadIdx.y][threadIdx.x] = weights[p*16*16
                + threadIdx.y*16 + threadIdx.x].x;
        __syncthreads();


        sum=0;
        for (int dy=0; dy<16; dy++) {
            for (int dx=0; dx<16; dx++) {
                sum += w[dy][dx] * in[threadIdx.y+dy][threadIdx.x+dx];
            }
        }
        output[grid_area*p+i].x = sum;
        __syncthreads();
    }
}


/*------------------ OrientationKernel1 -------------------
    input: Enhanced edge-sensitive receptive field  response
    output: Orientation vectors
---------------------------------------------------------*/
__global__
void OrientationKernel1(float1* input, float2* output) {
    int i = (((blockIdx.y)*16+threadIdx.y+8)
        *(grid_diameter/16)+blockIdx.x)*16 +threadIdx.x+8;

    float2 v = make_float2(0,0);
    for ( int p=0; p<12; p++ ) {
        float e = abs(input[p*grid_area + i].x);
        v.x += e*cos(-2*PI*p/12.0);
        v.y += e*sin(-2*PI*p/12.0);
    }
    output[i] = v;
}
```

```
/*------------------ OrientationKernel2 -------------------
    input: Orientation vectors
    actual: Actual orientation vectors
    weights: Gaussian weighting kernel
    output: Projection of perceived onto actual orientation
----------------------------------------------------------*/
__global__
void OrientationCortex2(float1* weights, float2* input,
            float2 *actual, float1 *output) {
    int i = (((blockIdx.y)*16+threadIdx.y+8)
            *(grid_diameter/16)+blockIdx.x)*16 +threadIdx.x+8;

    __shared__ float w[16][16];
    __shared__ float2 orient[2*16][2*16];
    orient[threadIdx.y][threadIdx.x] = input[i-8*grid_diameter-8];
    orient[threadIdx.y][threadIdx.x+16] = input[i-8*grid_diameter+8];
    orient[threadIdx.y+16][threadIdx.x] = input[i+8*grid_diameter-8];
    orient[threadIdx.y+16][threadIdx.x+16] = input[i+8*grid_diameter+8];
    w[threadIdx.y][threadIdx.x] =
        weights[threadIdx.y*16 + threadIdx.x].x;
    __syncthreads();

    float2 perceived = make_float2(0,0);
    for (int dy=0; dy<16; dy++) {
        for (int dx=0; dx<16; dx++) {
            perceived += orient[threadIdx.y+dy][threadIdx.x+dx] * w[dy][dx];
        }
    }
    float2 actual = normalize(actual[i]);
    output[i].x = dotP(perceived, actual);
}
```

APPENDIX B

CORTEXSIM 3.0 GPU KERNELS

```
/*-------------------- LabelKernel0 ------------------------
    input: Labeled cortex response
    v1: Enhanced edge-sensitive receptive field response
    output: First stage of spread activity in v1
----------------------------------------------------------*/
__global__
void LabelKernel0(float1* input, float1* v1, float1* output) {
    int i = (((blockIdx.y)*16+threadIdx.y+8)
            *(grid_diameter/16)+blockIdx.x)*16 +threadIdx.x+8;

    for ( int p=0; p<12; p++ ) {
        output[p*grid_area + i].x = v1[p*grid_area + i].x*input[i].x;
    }
}

/*-------------------- LabelKernel1 ------------------------
    input: Nth stage of spread activity in v1
    v1: Enhanced edge-sensitive receptive field response
    weights: V1 Gabor kernels
    output: N+1 th stage of spread activity in v1
----------------------------------------------------------*/
__global__
void LabelKernel1(float1* input, float1* v1,
        float1* weights, float1* output) {
    int i = (((blockIdx.y)*16+threadIdx.y+8)
            *(grid_diameter/16)+blockIdx.x)*16 +threadIdx.x+8;

    __shared__ float in[2*16][2*16];
    __shared__ float w[16][16];

    for ( int p=0; p<12; p++ ) {
        in[threadIdx.y][threadIdx.x] = input[i-8*grid_diameter-8].x;
        in[threadIdx.y][threadIdx.x+16] = input[i-8*grid_diameter+8].x;
        in[threadIdx.y+16][threadIdx.x] = input[i+8*grid_diameter-8].x;
        in[threadIdx.y+16][threadIdx.x+16] = input[i+8*grid_diameter+8].x;

        w[threadIdx.y][threadIdx.x] = weights[p*16*16
                + threadIdx.y*16 + threadIdx.x].x;
        __syncthreads();

        float sum=0;
        for (int dy=0; dy<16; dy++) {
            for (int dx=0; dx<16; dx++) {
                sum += in[threadIdx.y+dy][threadIdx.x+dx]*w[dy][dx];
            }
        }
        output[p*grid_area+i].x = v1[p*grid_area+i].x*sum;
        __syncthreads();
    }
}
/*------------------- LabelKernel2 ------------------------
    input0 : First stage of spread activity in v1
    input1 : Second stage of spread activity in v1
    input2 : Third stage of spread activity in v1
    input3 : Fourth stage of spread activity in v1
    output: Total spread activity
-----------------------------------------------------------*/
```

```
__global__
void LabelKernel2(float1* input0, float1* input1,
        float1* input2, float1* input3, float1* output) {
    int i = (((blockIdx.y)*16+threadIdx.y+8)
            *(grid_diameter/16)+blockIdx.x)*16+threadIdx.x+8;

    float sum=0;
    for ( int p=0; p<12; p++ ) {
        sum += input0[p*grid_area+i] + input1[p*grid_area+i]
         + input2[p*grid_area+i] + input3[p*grid_area+i];
    }
    output[p*grid_area+i].x = sum*rand();
}
/*------------------ NormalizeKernel0 ---------------------
    input : Total spread activity in v1
    weights: Gaussian weights
    inhib: Local inhibition
---------------------------------------------------------------*/
__global__
void NormalizeKernel0(float1 *input, float1 *inhib, float1 *weights) {
    int i = (((blockIdx.y)*16+threadIdx.y+8)
        *(grid_diameter/16)+blockIdx.x)*16+threadIdx.x+8;

    __shared__ float w[16][16];
    __shared__ float in[2*16][2*16];
    w[threadIdx.y][threadIdx.x] = weights[threadIdx.y*16 + threadIdx.x].x;
    in[threadIdx.y][threadIdx.x] = input[i-8*grid_diameter-8].x;
    in[threadIdx.y][threadIdx.x+16] = input[i-8*grid_diameter+8].x;
    in[threadIdx.y+16][threadIdx.x] = input[i+8*grid_diameter-8].x;
    in[threadIdx.y+16][threadIdx.x+16] = input[i+8*grid_diameter+8].x;
    __syncthreads();
    float sum = 0;
    for (int dy=0; dy<16; dy++) {
        for (int dx=0; dx<16; dx++) {
            sum += sigm(in[threadIdx.y+dy][threadIdx.x+dx])*w[dy][dx];
        }
    }
    inhib[i].x = sum;
}

/*------------------ NormalizeKernel1 ---------------------
    input : Total spread activity
    inhib : Local inhibitory activity
    output: Normalized activity
-------------------------------------------------------------*/
__global__
void NormalizeKernel1(float1* input, float1* inhib, float1 *output) {
    int i = (((blockIdx.y)*16+threadIdx.y+8)
        *(grid_diameter/16)+blockIdx.x)*16+threadIdx.x+8;
    output[i].x = BETA*sigm(input[i].x)/(ALPHA + inhib[i].x);
}
```