Visualization Design and Analysis: Abstractions, Principles, and Methods

Tamara Munzner

Department of Computer Science
University of British Columbia

(DRAFT – Friday 3rd August, 2012 – 01:51 – DRAFT – DO NOT CITE)

Contents

0

Preface

0.1 Why A New Book

This book was written to scratch my own itch: the book I wanted to teach out of for my graduate infovis course did not exist. The itch grew through the years of teaching my own course at UBC seven times, co-teaching a course at Stanford in 2001, and helping with the design of an early visualization course at Stanford in 1996 as a teaching assistant.

I was dissatisfied with teaching primarily from original research papers. While it is very useful for graduate students to learn to read papers, what was missing was a synthesis view and a framework to guide thinking. The principles and methods that I intended a particular paper to illustrate were often only indirectly alluded to in the paper itself. Even after assigning many papers or book chapters as preparatory reading before each lecture, I was frustrated by the many major gaps in the ideas discussed. Moreover, the reading load was so heavy that it was impossible to fit in any design exercises along the way, so the students only gained direct experience as designers in a single monolithic final project.

I was also dissatisfied with the lecture structure of my own course because of a problem shared by nearly every other course in the field: an incoherent approach to crosscutting the subject matter. Courses that lurch from one set of crosscuts to another are intellectually unsatisfying in that they make visualization seem like a grab-bag of assorted topics rather than a field with a unifying theoretical framework. There are at least four major ways to crosscut visualization material. One is by the field from which we draw techniques: cognitive science for perception and color, human-computer interaction for user studies and user-centered design, computer graphics for rendering, and so on. Another is by the problem domain addressed: biology vs. software engineering vs. computer networking vs. medicine vs. social/casual use. Yet another is by the families of techniques: focus+context vs. overview/detail vs. volume rendering vs. statistical graphics. Finally, evaluation is an important and central topic that should be interwoven throughout, but it did not fit into the standard pipelines and models. It was typically relegated to a single lecture, usually near the end,

so that it felt like an afterthought.

This book is structured around a new model of visualization that unifies design and evaluation issues in a single framework. It splits the design issues into four levels of concerns: problem characterization, data and task abstraction, visual encoding and interaction techniques suitable for the chosen abstractions, and algorithms to instantiate those techniques. The book begins with abstraction of tasks and data in Part II, followed by a set of principles of visual encoding and interaction in Part III. These principles do draw on other fields including perception and human-computer interaction, but the focus is on the underpinnings needed to understand visualization methods rather than these fields for their own sake. At the book's core is a taxonomy of methods that are the building blocks of visualization, in Part IV. These methods are a general framework through which any specific technique can be analyzed, starting from the highest-level categories of the construction of views and the reduction of data. In Part V a broad set of techniques are analyzed through this lens, for all of the fundamental data types addressed by visualization. This book focuses on analysis and design at the abstraction and encoding/interaction levels of the model. I have left out algorithms for reasons of space and time, not of interest – to discuss them at a useful level of detail would lead to a book that is an order of magnitude longer. Also, many good resources exist to learn about algorithms, including original papers and some of the previous books discussed below. In contrast, previous sources didn't provide enough of a framework for analysis, particularly at the abstraction level.

0.2 Existing Books

Visualization is a young field and there are not many books that provide a synthesis view of the field. The few books that exist all fell short in different ways. Tufte is a curator of glorious examples [Tufte 83, Tufte 91, Tufte 97], but he focuses on what can be done on the static printed page for purposes of exposition. The hallmark of the last twenty years of computer-based visualization is interactivity rather than simply static presentation, and the use of visualization for exploration of the unknown in addition to exposition of the known. Tufte does not address these topics, so while I use them as supplementary material, I find they cannot serve as the backbone for my own visualization course. However, any or all of them would well as supplementary reading for a course structured around this book; my own favorite for this role is Envisioning Information [Tufte 91].

Some instructors use the reader by Card, Mackinlay and Shneiderman [Card et al. 99]. Most of the book is a collection of seminal papers, and thus shares same problem as directly reading original papers: a lack

of synthesis. The first chapter provides a useful synthesis view of the field, but it is only one chapter. This book is an attempt to write an entire book along those lines, and one that is informed by the wealth of progress in our field in the past dozen years.

Ware has written a thorough book on visualization design as seen through the lens of perception [Ware 04], and I have used it as the backbone for my own course for many years. While it discusses many issues on how one could design a visualization, it does not cover what has been done in this field for the past fifteen years from a synthesis point of view. I wanted a book that allows a beginning student to learn from this collective experience rather than starting from scratch. This book does not attempt to teach the wonderful topic perception per se, covering only the aspects directly needed to get started with visualization and leaving the rest as further reading. Ware's shorter book on Visual Thinking For Design [Ware 08] would be excellent supplemental reading for a course structured around this book.

The recent book of Ward, Grinstein, and Keim [Ward et al. 10] works from the bottom up with algorithms as the base, whereas I wanted to work from the top down from the levels above the algorithmic. This book offers a considerably more extensive model and framework than Spence [Spence 07]. Wilkinson's Grammar of Graphics [Wilkinson 05] is a deep and thoughtful work, but is dense enough that it is more suitable for visualization insiders than for beginners. Conversely, Few's book [Few 04] is extremely approachable and has been used at the undergraduate level, but the scope is much more limited than the coverage of this book.

0.3 Audience

The primary audience of this book is students in a first visualization course, particularly at the graduate level. While admittedly written from a computer scientist's point of view, the book aims to accessible not only to those with a computer science background but also those without, such as students in a geography department or an i-school. Other audiences are people from other fields with an interest in visualization who would like to understand the principles and methods of this field, and practitioners in the field who might use it as a reference for a more formal analysis and improvements of production visualization applications.

This book is intended for people with an interest in the design and analysis of visualization techniques and systems. That is, this book is aimed at visualization designers, both nascent and experienced. This book is not directly aimed at end users of visualization techniques or systems, although they may well find some of this material informative.

The book is aimed at both both those who take a problem-driven approach and those who take a technique-driven approach. The focus is on broad synthesis of the general underpinnings of visualization in terms of principles and methods to provide a framework for the design and analysis of techniques, rather than the algorithms to instantiate those techniques.

The book considers visualization in a unified way, but as seen through the eyes of somebody who grew out of the tradition of information visualization. Somebody coming from a scientific visualization background would likely have a different conceptual view.

Part I

Introduction

1

Visualization Definition and Implications

The definition of visualization used in this book is:

> Computer-based **visualization** systems provide visual representations of datasets intended to help people carry out some task better. These visualization systems are often but not always interactive. Resource limitations include the capacity of computers, of humans, and of displays. The space of possible visualization system designs is huge and full of tradeoffs; many of the possibilities are ineffective.

The many implications of this definition frame the scope of this book; the rest of this introductory chapter expands on them:

- there is a human in the decision-making loop;
- the representation is generated by a computer, not by hand;
- the human visual perception system is the channel of communication;
- an external representation is used;
- the detailed structure of the dataset is important;
- there is an intended task, whether implicit or explicit;
- there is an operational definition of *better*;
- interactivity is on the table;
- resource limits for humans, computers, and displays all matter;
- the visualization design space is huge and full of tradeoffs;
- most visualization designs are ineffective.

Pronouns indicate roles in this book. **You** are the designer of a visualization system, the reader of this book. **They** are the intended users, the target audience for whom a visualization system is designed. **We** refers to all humans, especially in terms of our shared perceptual and cognitive responses. **I** am the author of this book, a role that I particularly emphasize when advocating a position or opinion that is not necessarily shared by all visualization researchers and practitioners.

1.1 A Human In The Loop

Some jobs that were once done by humans can now be completely automated so that the human is replaced by a computer-based solution. In contrast, visualization systems are appropriate for use when the goal is to augment human capabilities, rather than completely replace the human in the loop. If a fully automatic solution is acceptable and there is no need for human judgement, then there is no need for visualization. For example, a low-level algorithm for selling stocks when the market hits a specific price point can be run with no need at all for a time-consuming check from a human in the loop, so there would be no point in creating a visualization tool to help somebody make that check faster.

However, there are still many situations where total automation is inappropriate. In some cases, the problem is too complex and not sufficiently well defined for a computer to handle algorithmically. For example, finding a cure for cancer is a long-term goal for many human biologists. Many of them make heavy use of computational tools towards that end, but most people would agree that their work is unlikely to be automated away any time soon. Building visualization tools to help this target audience carry out some aspect of their daily workflow faster or better seems to be a good investment of resources.

In other cases, a proposed algorithmic solution may exist, but before it can be deployed it needs to be refined and extended, or have its results checked and validated by humans. In this case the goal of a visualization tool would be as an interim tool to help people debug that algorithm, and thus "work itself out of a job": after the algorithm refinement and verification phase is over, the automatic algorithm would be deployed. The need for intensive use of that particular visualization system would end, except perhaps for occasional monitoring. Returning to the stock market example, a higher-level system that determines which of multiple trading algorithms to use in varying circumstances might require careful tuning. A visualization system to help the algorithm developers analyze its performance might be an interim tool, not something intended to be used on a daily basis for an indefinite period of time.

1.2 A Computer In The Loop

People can and have created visual representations of datasets manually, either completely by hand with pencil and paper, or with computerized drawing tools where they individually arrange and color each item. However, there are major limits on the amount of time and attention that people are willing and able to devote to this endeavor. While people might arrange dozens of items, even small datasets have hundreds of items. Most real-world datasets are much larger, ranging from thousands to millions to even more. Moreover, many datasets are not static, but rather change dynamically over time. Having a computer-based tool generate the visual representation automatically obviously saves human effort compared to manual creation. It allows us to see larger datasets that would be completely infeasible to draw by hand, and opens up the possibility of seeing how data changes over time.

One important research frontier is to characterize the important aspects of hand-drawn diagrams in order to automatically create drawings similar in spirit to them [Agrawala and Stolte 02, Barsky et al. 08].

1.3 Visual Perception

Visualization, as the name implies, is based on exploiting the human visual system as a means of communication because it is a very high-bandwidth channel to the brain. A significant amount of visual information processing occurs at the pre-conscious level. One example is visual popout, such as when one red item is immediately noticed from a sea of grey ones. The popout occurs whether the field of other objects is large or small, because of processing done in parallel across the entire field of vision. Of course, the visual system also feeds into higher-level processes that involve the conscious control of attention. Another example of visual information processing is our ability to think and act as if we see a huge amount of information at once, even though we actually see only a tiny part of our visual field in high resolution at any given instant. In contrast, we experience the perceptual channel of sound sequentially, rather than as a simultaneous overview. The perceptual channels of taste and smell do not yet have viable recording and reproduction technology. Haptic input and feedback devices exist to exploit the touch and kinesthetic perceptual channels, but exploration of their effectiveness for communicating abstract information is still at an early stage. In this book, the focus is on the visual system, which is thus far the best characterized and understood.

Chapter 5 contains a further discussion of many important aspects of visual perception.

1.4 An External Representation

Visualization allows people to offload internal cognition and memory usage to the perceptual system, using carefully designed images as a form of **external representations**, sometimes also called *external memory*. External representations can take many forms, including touchable physical objects like an abacus or a knotted string. This book focuses on display surfaces that are perceived by the human visual system, ranging from paper to whiteboards to napkins to computer screens.

The advantages of diagrams as external memory is that information is organized by spatial location, offering the possibility of accelerating both search and recognition. Large amounts of the search that would be required to make a problem-solving inference can be avoided by grouping all the items that are used together at the same location. Recognition can be facilitated by grouping all the relevant information about one item in the same location, avoiding the need for matching remembered symbolic labels. Diagrams support perceptual inferences, which are very easy for humans to make. The important caveat is that any particular diagram may or may not be constructed take advantage of these features. A non-optimal diagram may group irrelevant information together, or support perceptual inferences that are not useful for the problem-solving process.

1.5 Show The Data

Our definition of visualization implies that there are situations where seeing the dataset structure in detail is better than seeing only a brief summary of it. Statistical characterization of datasets is a very powerful approach, but has the intrinsic limitation of losing information through summarization. Figure 1.1 shows Anscombe's Quartet, a suite of four datasets designed by a statistician to illustrate how datasets that have identical simple statistics can have very different structures that are immediately obvious when their full structure is shown graphically [Anscombe 73]. The identical principle holds for much larger and more complex datasets: summaries often oversimplify, hiding the true structure of the dataset.

Anscombe's Quartet: Raw Data

	I		II		III		IV	
	x	**y**	**x**	**y**	**x**	**y**	**x**	**y**
	10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
	8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
	13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
	9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
	11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
	14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
	6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
	4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
	12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
	7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
	5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89
mean	9.0	7.5	9.0	7.5	9.0	7.5	9.0	7.5
var.	10.0	3.75	10.0	3.75	10.0	3.75	10.0	3.75
corr.	0.816		0.816		0.816		0.816	

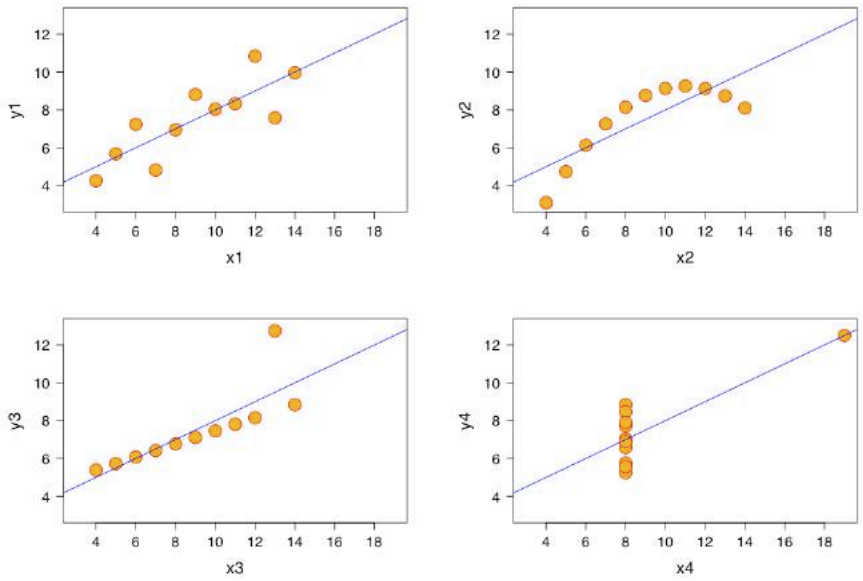


Figure 1.1: Anscombe's Quartet is four datasets with identical simple statistical properties: mean, variance, correlation, and linear regression line. However, visual inspection immediately shows how their structures are quite different [Anscombe 73].

1.6 Driving Task

Our definition is also centered around an intended task. In some cases this task is very explicitly characterized and crisply stated. In others, the task is broader or more general, for instance to find a trend in a dataset. At a very high level, we can divide the intended tasks into three categories: hypothesis generation, hypothesis confirmation, or presentation. Visualization can be used to generate new hypotheses, for example when exploring a completely unfamiliar dataset. It can be used to confirm existing hypotheses about some dataset that is partially understood. A third use is to present information about a dataset that is understood by one person to some other target audience. As we will discuss at length, we can and should tune the visual representation for the task. We further discuss tasks in Chapter 3.

1.7 The Meaning of Better

Our definition includes the idea that one representation can be better or more effective than another when judged by how well the information contained in the dataset is communicated to the human. There are a vast number of possible visual representations, also known as **visual encodings**, for even a single dataset. One of the central problems in visualization is choosing appropriate representations from this huge space. A determination that one is better than another must take into account many factors, including the characteristics of the human perceptual system, the dataset in question, and the task at hand. This determination is a very difficult problem. Chapter 2 presents a model of visualization design in terms of multiple levels at which the design of a visualization system can and should be evaluated. The book has further discussion of dataset characteristics in Chapter 4, tasks in Chapter 3, and perceptual issues in Chapter 5.

The idea that one representation can be better than another for showing dataset structure leads to a great concern with correctness, accuracy, and truth. The data-driven aspect of visualization is in contrast to the many other uses of human visual perception: to convey emotion or provoke thought or set a mood, as in art; to tell a narrative story, as in movies or comics; to sell, as in advertising. The goals of visualization are a contrast with those of these other fields, where the evocative term *artistic license* holds sway. For the goals of emotional engagement, storytelling, or allurement, the deliberate distortion of facts and reality is often entirely appropriate, and of course fiction is as respectable as nonfiction.

However, *any* depiction of data is an abstraction where choices are made about which aspects to emphasize. No picture can communicate the truth,

the whole truth, and nothing but the truth. Even the goal of a photorealistic depiction of a real-world scene involves choices of abstraction and emphasis, for example the choice of what to include in the frame of the photograph. Cartographers have thousands of years of experience with articulating the difference between the abstraction of a map, and the terrain that it represents. Abstraction is discussed in much more detail in Chapter 2 and Part II.

1.8 Interactivity

Before the widespread deployment of fast computer graphics, visualization was limited to the use of static images on paper. With computer-based visualization, interactivity becomes possible, vastly increasing the scope and capabilities of visualization tools. Although static visualization techniques are indeed within the scope of this book, interaction is an intrinsic part of many of the methods discussed in Part IV. The discussion begins with principles of interaction, covered in Chapter 6.

1.9 Resource Limitations

Visualization systems are inevitably used for situations past their design targets, especially in terms of dataset size. The continuing increase in dataset size is driven by many factors: improvements in data acquisition and sensor technology, bringing real-world data into a computational context; improvements in computer capacity, leading to ever-more generation of data from within computational environments including simulation and logging; and the increasing reach of computational infrastructure into every aspect of life.

Thus, **scalability** is a central concern: designing systems to handle large amounts of data gracefully. Users might also want to explore a heterogenous collection of data, rather than whatever limited set the system was designed to handle.

When analyzing a visualization system, the designer must consider at least three different kinds of limitations: computational capacity, human perceptual and cognitive capacity, and display capacity. As with any application of computer science, computer time and memory are limited resources and there are often a mix of soft and hard constraints on how much of each are available. For instance, if the visualization system needs to deliver interactive response in response to user input, then it must use algorithms when drawing each frame that can run in a fraction of a second rather than minutes or hours. In some usage scenarios, users are unwilling

or unable to wait a long time for the system to preprocess the data before starting to interact with it. A soft constraint is that the visualization system should be parsimonious in its use of computer memory because the user needs to run other programs simultaneously. A hard constraint is that even if the visualization system is free to use nearly all available memory in the computer, dataset size can easily outstrip that finite capacity. Designing systems that gracefully handle larger datasets that do not fit into core memory requires significantly more complex algorithms. Thus, the computational complexity of algorithms for dataset preprocessing, transformation, layout, and rendering is a major concern. However, that is by no means the only concern!

On the human side, memory and attention must be considered as finite resources. Chapter 5 will discuss some of the power and limitations of the low-level visual preattentive mechanisms that carry out massively parallel processing of our current visual field. However, human memory for things that are not directly visible is notoriously limited. These limits come into play not only for long-term recall but also for shorter-term working memory, both visual and non-visual. We store surprisingly little information internally in visual working memory, leaving us vulnerable to **change blindness**: the phenomenon where even very large changes are not noticed if we are attending to something else in our view [Simons 00], as discussed in Chapter **??**. Human attention also has definite limits. Conscious search for items is an operation that grows more difficult with the number of items there are to check. Moreover, vigilance is also a highly limited resource: our ability to perform visual search tasks degrades quickly, with far worse results after several hours than in the first few minutes [Ware 04].

Display capacity is a third kind of limitation to consider. Visualization designers often "run out of pixels", where the resolution of the screen is not enough to show all desired information simultaneously. The **information density** of a single image is a measure of the amount of information encoded versus the amount of unused space.[1] Figure 1.2 shows an example tree visually encoded three different ways. The encoding in Figure 1.2a encodes the depth from root to leaves in the tree with vertical spatial position. However, the information density is low. Quantitatively, we could compute the ratio between the size of each node, and the area required to display the entire tree. In contrast, the encoding in Figure 1.2b uses nodes of the same size but is drawn more compactly, so it has higher information density; however, the depth cannot be easily read off from spatial position. Figure 1.2c shows a very good alternative that combines the benefits of both previous approaches, with both high information density from a

[1] Vocabulary note: Synonyms for information density include graphic density and data-ink ratio.

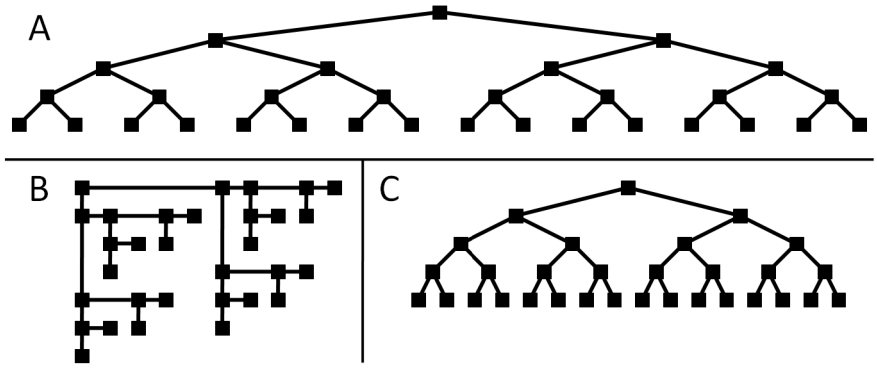


Figure 1.2: Low and high information density visual encodings of the same small tree dataset. a) Low information density. b) Higher information density, but depth in tree cannot be read off from spatial position. c) High information density, while maintaining property that depth is encoded with position. From [McGuffin and Robert 10], Figure 3.

compact view and position coding for depth.

There is a tradeoff between the benefits of showing as much as possible at once, to minimize the need for navigation and exploration, and the costs of showing too much at once, where the user is overwhelmed by visual clutter. Most of the methods discussed in Part IV can be considered as a framework in which to find an appropriate balance between these two ends of the information density continuum.

1.10 Visualization Design Space

The **design space** of possibile ways to design of visualization systems and techniques is enormous. Chapter 5 will provide an initial explanation of part of this design space in terms of how to analyze basic visual encodings with the theory of visual channels. The chapters in Part IV will provide a further framework for considering the design possibilities available to you as a visualization designer with an even larger set of methods.

As with all design problems, visualization design cannot easily handled as a simple process of optimization because tradeoffs abound. A design that does well by one measure will rate poorly on another. The characterization of tradeoffs in the visualization design space is a very open problem at the frontier of visualization research. There are several guidelines and suggested processes in this book, based on my synthesis of what is currently known, but few absolute truths.

Chapter 2 introduces a model for thinking about the design process at four different levels, intended to guide your thinking through these tradeoffs in a systematic way.

1.11 Most Designs Ineffective

The most fundamental reason that visualization design is a difficult enterprise is that the vast majority of the possibilities in the design space will be ineffective. In some cases, a possible design is a poor match with the properties human perceptual and cognitive systems. In others, the design would be comprehensible by a human, but it's a bad match with the intended task. Only a very small number of possibilities are in the set of reasonable choices, and of those only an even smaller fraction are excellent choices. Thus, a random walk through the space of possibilities is a bad idea because the odds of finding a very good solution in this way are very low.

Figure 1.3 shows a diagram to help you think about design in terms of traversing a search space. In design problems, it's not a very useful goal to **optimize**; that is, to find the very best choice. A more appropriate goal when you design is to **satisfy**; that is, to find one of the many possible good solutions rather than one of the even larger number of bad ones. The diagram shows five spaces, each of which is progressively smaller than the previous. First, there is the space of all possible solutions, including potential solutions that nobody has ever thought of before. Next, there is the set of possibilities that are *known* to you, the visualization designer. Of course, this set might be small if you are a novice designer who is not aware of the full array of methods that have been proposed in the past. If you're in that situation, one of the goals of this book is to enlarge the

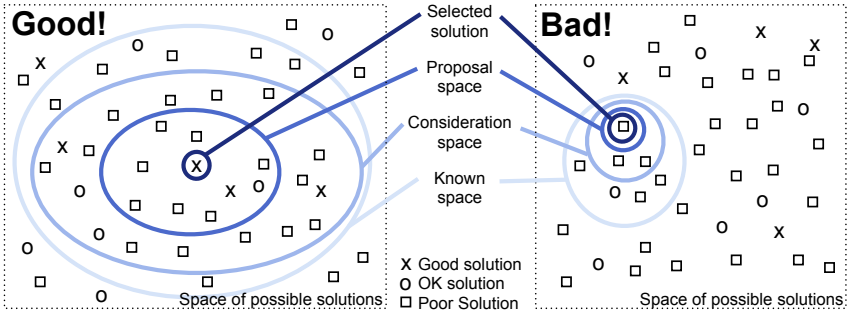


Figure 1.3: A search space metaphor for visualization design.

set of methods that you know about. The next set is the *consideration* space of the solutions that you actively consider. This set is necessarily smaller than the known space, because you can't consider what you don't know. An even smaller set is the *proposal* space of possibilities that you investigate in detail. Finally, one of these becomes the *selected* solution.

The diagram in Figure 1.3 contrasts a good strategy on the left, where the known and consideration spaces are large, with a bad strategy on the right, where these spaces are small. The problem of a small consideration space is the higher probability of only considering ok or poor solutions and missing a good one. A fundamental principle of design is to consider multiple alternatives and then choose the best, rather than to immediately fixate on one solution without considering any alternatives. One way to ensure that more than one possibility is considered is to explicitly generate multiple ideas in parallel. This book is intended to help you as a designer entertain a broad consideration space by systematically considering many alternatives, and also to help you rule out some parts of the space by noting when there are mismatches of possibilities with human capabilities or the intended task.

1.12 Further Reading

This section provides pointers for further reading about the ideas presented in this chapter and acknowledges key sources that influenced my discussion; there is a section like this at the end of each chapter in the book.

Zhang has analyzed the role and use external representations extensively, including the nature of external representations in problem solving [Zhang 97] and a representational analysis of number systems [Zhang and Norman 95]. Larkin and Simon's influential work on "Why A Diagram is (Sometimes) Worth Ten Thousand Words"is the basis for my discussion of diagrams in this chapter [Larkin and Simon 87].

Anscombe proposed his quartet of illustrative examples in a lovely, concise paper [Anscombe 73].

Ware's textbook provides a very thorough discussion of human limitations in terms of perception, memory, and cognition [Ware 04]. Simons presents a good overview of the change blindness literature [Simons 00].

The idea of information density dates back to Bertin's discussion of *graphic density* [Bertin 67], and Tufte has discussed the *data-ink ratio* at length [Tufte 83, Tufte 91, Tufte 97].

My discussion of the visualization design space is based on a paper on the methodology of design studies, written in collaboration with Sedlmair and Meyer, that presents a framework for problem-driven visualization research [Sedlmair et al. 12].

2

A Visualization Design Framework

Many computer-based visualization techniques and systems have been created in the past thirty years. If you simply look at them one by one as a big collection of different possibilities, it is hard to decide what to do when you are confronted with a visualization problem as a designer. You might be able to check whether a particular idea that you have has been tried before, and if you dig into the literature on empirical evaluation you might even be able to check whether it worked well or poorly for the tasks that it was tested against. However, it is hard to learn from past work without an underlying framework for thinking about your design choices systematically.

This book is built around such a framework, which is summarized in this chapter. The first section discusses the structure of the book itself. The next section presents a four-level model for design and validation, followed by the threats to design validity at each level. The chapter concludes with six case studies of visualization projects that focus on design and validation at different levels; they also serve as a preview of the wide scope of possibilities for interactive visualization systems.

2.1 Book Structure

Part II of this book introduces the idea of abstractions, motivated by the four-level design model introduced in Section 2.2. Chapter 3 covers task abstractions, and Chapter 4 presents abstractions for data.

The design of many different techniques is motivated by the same underlying principles, which are presented in Part **??**. The principles of visually encoding data are covered in Chapter 5, and of interacting with data in Chapter 6. Figure 2.1 provides a preview of the core visual encoding principle that images can analyzed according to how geometric marks convey information with different visual channels.

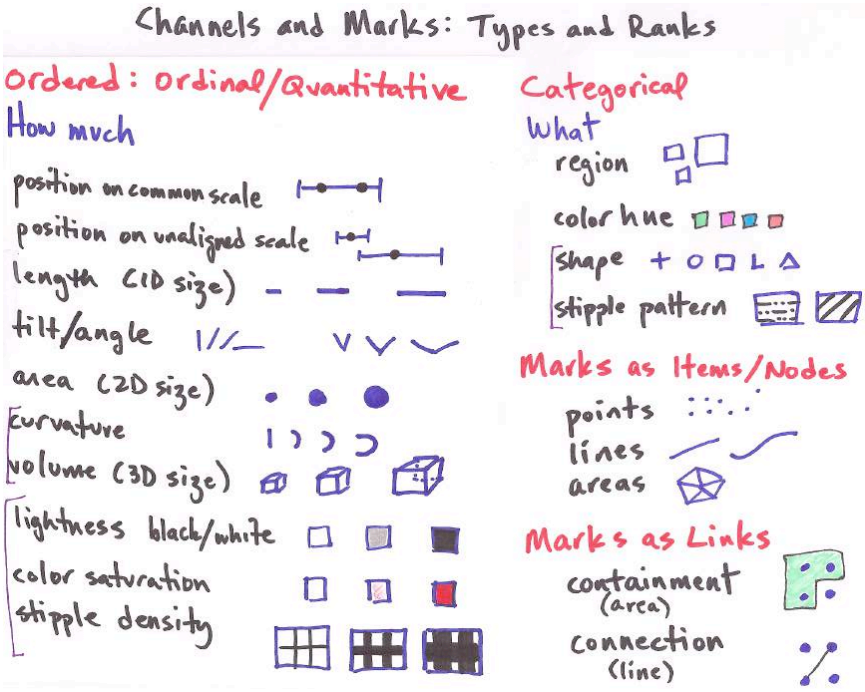


Figure 2.1: Framework of principles for visual encoding with geometric marks and visual channels, discussed in detail in Chapter 5.

Part IV of this book is built around an analysis framework that breaks down the visualization design space into a taxonomy of methods. I introduce it with the goal of helping you analyze existing techniques and systems and to design new ones. This taxonomy of methods is only one possible framework for analysis and design that may be particularly useful as a scaffold for newcomers to visualization; it is not intended as a straitjacket!

Figures 2.2 and 2.3 provide a preview of this methods framework. The methods are divided into two main concerns: Chapter 8 covers the fundamental set of ways to make a view, and Chapter 9 covers the ways to reduce the number of things to show within views. A central concern in making views is how to use space, the most important visual channel. When multiple views are combined, there are methods for coordinating them to work together. Many datasets are far too large and complex to understand completely with a single static diagram, so most interactive visualization techniques employ methods to reduce the number of things to show at once and to change what is visible in response to user actions. These methods

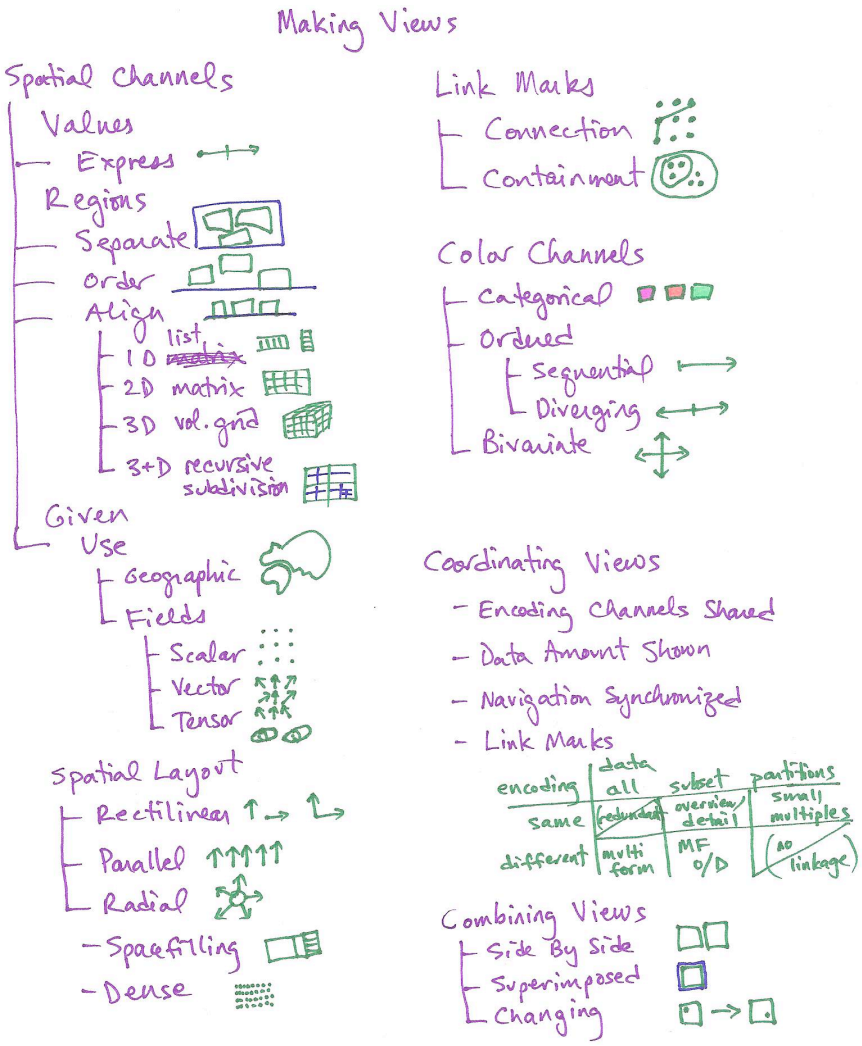


Figure 2.2: Framework of methods for making views, discussed in detail in Chapter 8.

act as building blocks for the visualization designer; existing techniques can be analyzed in terms of them.

All of the methods are illustrated with previous techniques and systems as concrete examples. Each example is analyzed with as much of the analysis framework as has been covered where it appears. I define a **technique**

as a specific approach to visual encoding or interaction, as opposed to a **method** that is a more general design choice that could apply to many different techniques. A full visualization **system** often combines multiple techniques.

Part V concludes the book with several case studies that show how to analyze complex visualization systems using the full principles and methods frameworks presented in this book.

The last chapters in Parts **??** and IV present guidelines; many of them are summarized with mnemonic slogans. The principles slogans in Chapter **??** are *No Unjustified 3D*, *Eyes Over Memory*, *Resolution Over Immersion*, *Function First, Form Next*, and *Get It Right in Black and White*. Guidelines that pertain to the strengths and weaknesses of single methods are interleaved with their presentation in Chapters 8 and 9, with extensive discussions of tradeoffs that touch on many methods at the end in Chapter **??**. These guidelines include *Overview First, Detail on Demand, Zoom and Filter*, *Search, Show Context, Expand on Demand*, *Distortion Costs and Benefits*, *Network Tradeoffs: Connection, Containment, and Matrix Veiws*, and *Displaying Dimensionally Reduced Data*. These guidelines are my own interpretation given the present knowledge of the field; while some reflect the mainstream consensus, others are more contentious. Visualization is a relatively young field and many questions remain open.

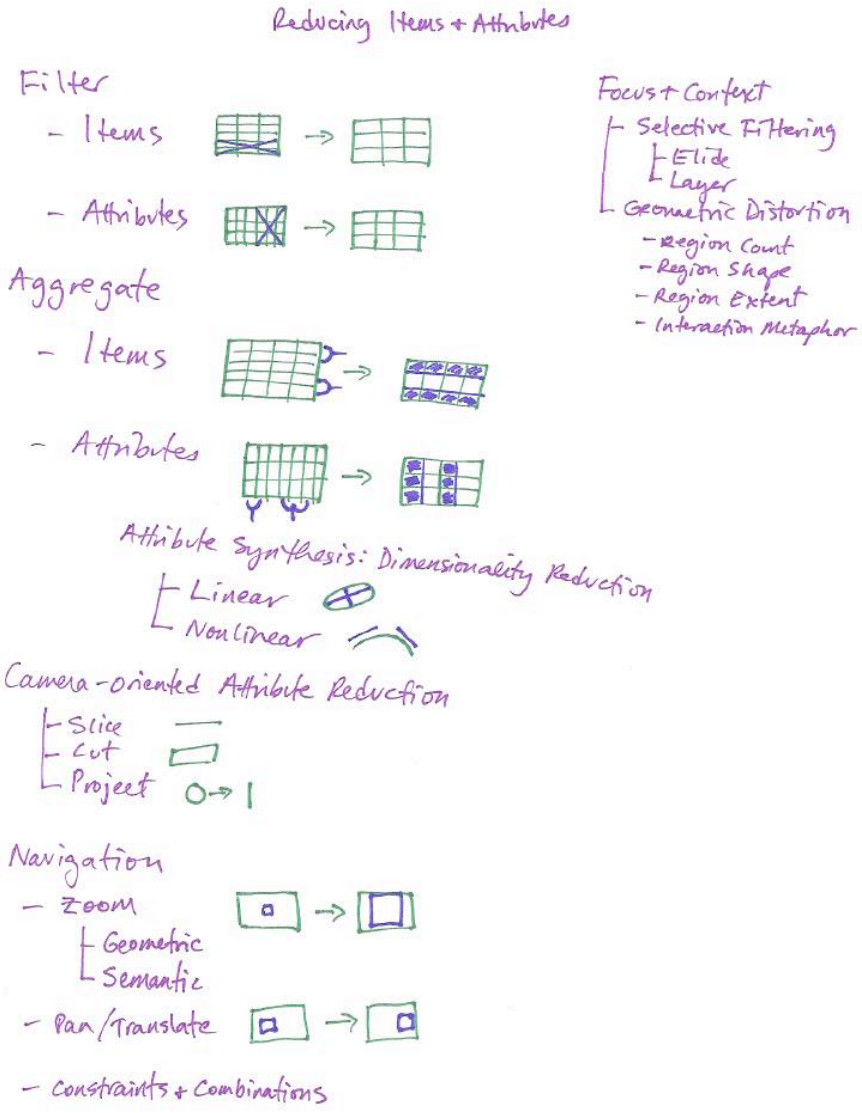


Figure 2.3: Framework of methods for reducing what is shown at once, discussed in detail in Chapter 9.

2.2 Four Levels of Design

When we design a visualization, how do we figure out if we have succeeded? There are many criteria we might use. We could ask whether somebody using the system can do something better. But what does *better* mean? Do they get something done faster? Do they have more fun doing it? Can they work more effectively? But what does *effectively* mean? How do we measure *insight* or *engagement*? And better than *what*? Another visualization system? Doing the same things manually, without visual support? Doing the same things completely automatically? And to do *something* better - what sort of thing? That is, how do we decide what sort of task they should do when testing the system? And who is this *somebody*? An expert who has done this task for decades, or a novice who needs the task explained before they begin? Are they familiar with how the system works from using it for a long time, or are they seeing it the first time? Even a concept like *faster* that might seem straightforward gets tricky. Are they limited by the speed of their own thought process, or their ability to move the mouse, or simply the speed of the computer in drawing each picture?

Considering all these questions at the same time is difficult and confusing. The complex problem of visualization design is made easier by splitting concerns into four cascading levels, as shown in Figure 2.4. The top level of **problem characterization** is to characterize the problems and data of a particular target domain. The next level of **abstraction** is to map those domain-specific problems and data into abstract and generic tasks and data types. The third level, **encoding and interaction**, is to design the visual encoding and interaction techniques that use the data to support those tasks. The innermost fourth level of **algorithm** is to create an algorithm to carry out that design efficiently.

This particular split into levels is motivated by shared **threats to validity** at each one. In brief, these threats are:

- wrong problem: they don't do that;

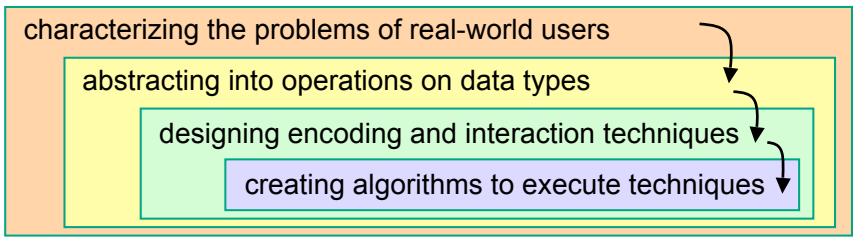


Figure 2.4: The four nested levels of visualization design.

- wrong abstraction: you're showing them the wrong thing;
- wrong encoding/interaction: the way you're showing them the thing doesn't work;
- wrong algorithm: your code is too slow.

The top level, understanding the requirements of some target audience, is a tricky problem. In a human-centered design approach, the visualization designer works with a group of target users over time. In most cases users know they need to somehow view their data, but usually cannot directly articulate their needs as clear-cut tasks that operate on data types. The iterative design process includes gathering information from the target users about their problems through interviews and observation of them at work, creating prototypes, and observing how users interact with those prototypes to see how well the proposed solution actually work.

The next level requires abstracting the specific domain problem identified in the first level into a more generic representation. Problems from very different domains can map to the same generic visualization tasks such as sorting, filtering, characterizing trends and distributions, finding anomalies and outliers, and finding correlation. This abstraction step usually also involves transformations from the original raw data into suitable derived dimensions, as discussed in Chapter 4.

At the third level, there are many possible visual encodings and ways to interact with the transformed dataset. The choices at this level can be guided by the abstract tasks identified at the previous level, since they correspond to the specific needs of some intended user. The fourth level involves all of the choices in designing algorithms to carry out the choices made at the three higher levels.

These levels are nested; the output from an *upstream* level above is input to the *downstream* level below, as indicated by the arrows in Figure 2.4. The challenge of this nesting is that an upstream error inevitably cascades to all downstream levels. If a poor choice was made in the abstraction stage, then even perfect visual encoding and algorithm design will not create a visualization system that solves the intended problem. The three inner levels are all instances of design problems, although it is a different problem at each level.

Although this model is cast as four nested levels in order to discuss the issues at each level separately, in practice these four stages are rarely carried out in strict temporal sequence. There is usually an iterative refinement process, where a better understanding of one level will feed back and forward into refining the others. The intellectual value of separating these four stages is that the question of whether each level has been addressed

correctly can be separately analyzed, no matter in what order they were undertaken.

2.2.1 Domain Problem and Data Characterization

At this first level, a visualization designer must learn about the tasks and data of target users in some particular target **application domain**, such as microbiology or high-energy physics or e-commerce. Each domain usually has its own vocabulary for describing its data and problems, and there is usually some existing workflow of how the data is used to solve their problems.

A central tenet of human-centered design is that the problems of the target audience need to be clearly understood by the designer of a tool for that audience. Although this concept might seem obvious, sometimes designers cut corners by making assumptions rather than actually engaging with any target users. Moreover, eliciting system requirements is not easy, even when a designer has access to target users fluent in the vocabulary of the domain and immersed in its workflow. Asking users to simply introspect about their actions and needs is notoriously insufficient: interviews are only one of many methods in the arsenal of ethnographically-inspired methodology that has been used in human-computer interaction (HCI). The methodology of requirements analysis from the field of software engineering can also be useful.

The output of domain workflow characterization is often a detailed set of questions asked about or actions carried out by the target users for some heterogeneous collection of data. The details are necessary: in the list above, the high-level domain problem of "cure disease" is not sufficiently detailed to be input to the next abstraction level of the model, whereas the lower-level domain problem of "investigate microarray data showing gene expression levels and the network of gene interactions" is more appropriate. In fact, even that statement is a drastic paraphrase of the domain problem and data description in the full design study [Barsky et al. 08].

2.2.2 Task and Data Type Abstraction

The abstraction stage is to map problems and data from the vocabulary of the specific domain into a more abstract and generic description that is in the vocabulary of information visualization. The output of this level is a description of generic tasks and data types, which are the input required for making visual encoding and interaction decisions at the next level.

Although different domains typically use completely different specialized vocabulary to discuss their problems, the challenge for the visualization designer doing task abstraction is to map these needs to generic tasks

that do not rely on domain-specific concepts. For example, one list of high-level generic tasks is expose uncertainty, concretize relationships, formulate cause and effect, determine domain parameters, multivariate explanation, and confirm hypotheses [Amar and Stasko 04].

One list of low-level generic tasks is retrieve value, filter, compute derived value, find extremum, sort, determine range, characterize distribution, find anomalies, cluster, and correlate [Amar et al. 05]. Another categorization of low-level generic tasks is identify, determine, visualize, compare, infer, configure, and locate [Valiati et al. 06]. Task taxonomies can also include tasks that only pertain to specific data types, such as following a path through a network. The key idea that runs through all of these task taxonomies is that they describe generic tasks without ties to the problems and vocabulary of a specific application domain. Chapter 3 presents a taxonomy of abstract tasks and discusses the problem of moving from the problem characterization level to the abstraction level.

Similarly, data abstraction requires the visualization designer to consider whether and how the same dataset provided by a user should be transformed into another form. Many visualization methods are specific to a particular **data type**, such as a table of numbers where the columns contain quantitative, ordered, or categorical data; a node-link graph or tree; or a field of values at every point in space. The goal of the designer is to determine which data type would support a visual representation of it that addresses the problem of the user. Although sometimes the original form of the dataset is a good match for a visual encoding that solves the problem, often a transformation to another data type provides a better solution. Chapter 4 discusses these abstract data types and their transformations in detail.

Explicitly considering the choices made in abstracting from domain-specific to generic tasks and data can be very useful in the visualization design process. The unfortunate alternative is to do this abstraction implicitly and without justification. For example, many early web visualization papers implicitly posited that solving the "lost in hyperspace" problem should be done by showing the searcher a visual representation of the topological structure of its hyperlink connectivity graph [Munzner and Burchard 95]. In fact, people do not need an internal mental representation of this extremely complex structure to find a web page of interest. Thus, no matter how cleverly the information was visually encoded, these visualizations all incurred additional cognitive load for the user rather than reducing it.

Some designers skip over the domain problem characterization level completely, short-circuit the abstraction level by assuming that the first abstraction that comes to mind is the correct one, and jump immediately into the third level of visual encoding and interaction design. I argue

against this approach; the abstraction stage is often the hardest to get right. A designer struggling to find the right abstraction may end up realizing that the problem domain has not yet been adequately characterized, and jump back up to work at that level before returning to this one. As is discussed in Section 2.2, the design process is rarely strictly linear.

2.2.3 Visual Encoding and Interaction Design

The third level is designing the **visual encoding**, the visual representation that the user sees, and the **interaction**, the techniques that allow the user to change what is shown dynamically. These two pieces are often mutually interdependent, so they are considered at the same level. The principles of visual encoding are covered in Chapter 5, and of interaction in Chapter 6. The methods discussed in Part IV are informed by these principles, and many methods intertwine visual encoding and interaction considerations. In this introductory chapter, the goal is simply to emphasize the difference between these very visible aspects of visualization design, versus the abstraction design at the level above, and the algorithm design at the level below.

2.2.4 Algorithm Design

The innermost level is to create an **algorithm**, a detailed procedure that allows a computer to automatically carry out the visual encoding and interaction designs from the level above. This four-level model collapses algorithmic and implementation considerations together; a more detailed model could pull them apart into two separate levels. The distinction made here in terms of design is that the third level of encoding and interaction addresses the question of *what* should happen, while the algorithm level addresses the question of *how* to do it. Of course, there is an interplay between these levels. A design that requires something to change dynamically when the user moves the mouse may not be feasible if computing that would take minutes or hours instead of a fraction of a second – but perhaps some combination of precomputation, caching, and clever algorithm design could save the day.

While algorithm design for visualization is a rich and wonderful topic, it is outside the scope of this book for several reasons. This book focuses on synthesis across the upper levels because the algorithm level is much easier than the other three to understand from reading existing resources. In most cases, the best place to start for algorithms is recent survey papers on a particular topic, or specific research papers. The Further Reading sections at the end of each chapter provide some pointers for where to get started. There are also a several existing books with a heavy focus on algorithms,

including textbooks from Ward et al. [Ward et al. 10] and Telea [Telea 07] and the handbook from Johnson et al. [Hansen and Johnson 05].

Moreover, the state of the art in algorithms changes quickly, whereas this book aims to provide a framework for thinking about design that will last longer. The book uses specific past systems and techniques as examples that illustrate the framework principles and methods, not as the final answer for the very latest and greatest way to solve a particular design problem. More pragmatically, this book would be much too long if it covered algorithms at any reasonable depth; the other three levels comprise more than enough material for a single volume of readable size. Finally, this book is intended to be accessible to people without a computer science background. The larger issues of algorithm design are certainly not unique to visualization; an excellent general reference for algorithms is the textbook of Cormen et al. [Cormen et al. 90].

2.3 Threats and Validation

Each level in this model has a different set of threats to validity, and thus requires a different approach to validation. Figure 2.5 shows a summary of the threats and validation approaches possible at each level, which are discussed in detail in the rest of this section.

The analysis below distinguishes between **immediate** and **downstream** validation approaches. An important corollary of the model having nested levels is that most kinds of validation for the outer levels are not immediate because they require results from the downstream levels nested within them. The length of the red lines in Figure 2.5 shows the magnitude of the dependencies between the threat and the downstream validation, in terms of the number of levels that must be addressed. These downstream dependencies add to the difficulty of validation: a poor showing of a test may misdirect attention upstream, when in fact the problem results from a poor choice at the current level. For example, a poor visual encoding choice may cast doubt when testing a legitimate abstraction choice, or poor algorithm design may cast doubt when testing an interaction technique. Despite their difficulties, the downstream validations are necessary. The immediate validations only offer partial evidence of success; none of them are sufficient to demonstrate that the threat to validity at that level has been addressed.

This model uses the language of upstream and downstream in order to make the discussion of the issues at each level easier to understand – but it is not always necessary to carry out the full process of design and implementation at each level before doing any downstream validation. There are many rapid prototyping methodologies for accelerating this process by creating low-fidelity stand-ins exactly so that downstream validation can

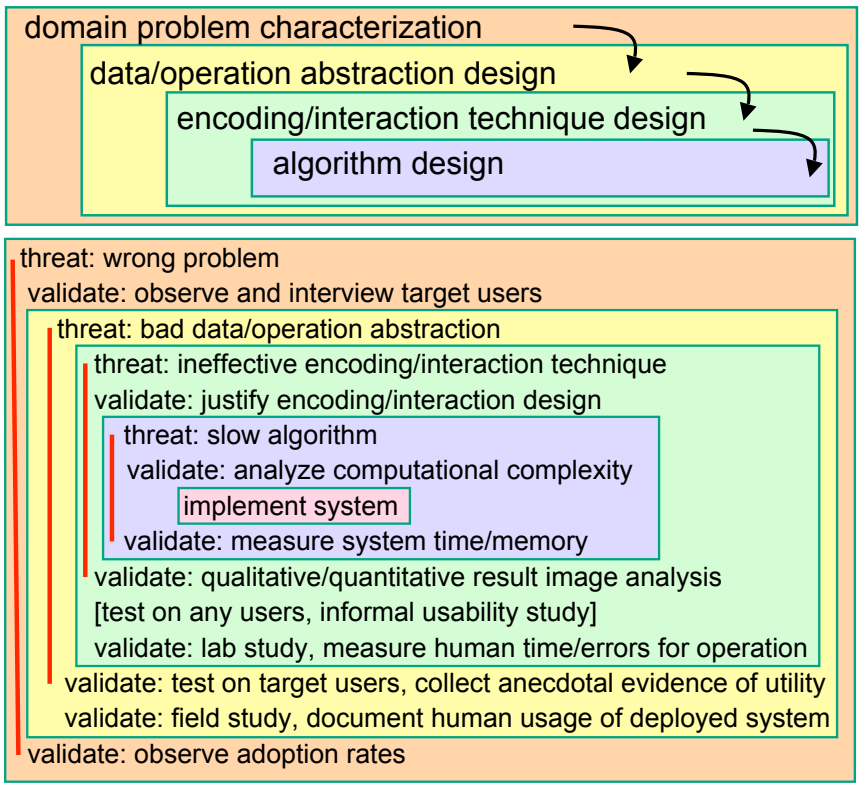


Figure 2.5: Levels and threats. (a) The four levels of visualization design. (b) Threats and validation at each level. Many threats at the outer levels require downstream validation, which cannot be carried out until the inner levels within them are addressed, as shown by the red lines. Any single project would only address a subset of these levels, not all of them at once.

occur sooner. For example, paper prototypes and wizard-of-oz testing [Dow et al. 05] can be used to get feedback from target users about abstraction and encoding designs before diving into designing or implementing any algorithms.

2.3.1 Domain Threats

At the top level of characterizing the domain problems and data, a visualization designer is asserting that particular problems of the target audience would benefit from visualization tool support. The primary threat is that the problem is mischaracterized: the target users do not in fact have these

problems. An immediate form of validation is to interview and observe the target audience to verify the characterization, as opposed to relying on assumptions or conjectures. These validation approaches are mostly qualitative rather than quantitative, and appropriate methodologies include ethnographic field studies and semi-structured interviews.

A downstream form of validation is to report the rate at which the tool has been adopted by the target audience. Adoption rates can be considered to be a weak signal with a large rate of false negatives and some false positives: many well-designed tools fail to be adopted, and some poorly-designed tools win in the marketplace. Nevertheless, the important aspect of this signal is that it reports what the target users do of their own accord, as opposed to the approaches below where target users are implicitly or explicitly asked to use a tool.

2.3.2 Abstraction Threats

At the abstraction design level, the threat is that the chosen tasks and data types do not solve the characterized problems of the target audience. The key aspect of validation against this threat is that the system must be tested by target users doing their own work, rather than an abstract task specified by the designers of the study.

A common downstream form of validation is to have a member of the target user community try the tool, in hopes of collecting anecdotal evidence that the tool is in fact useful. These anecdotes may have the form of insights found or hypotheses confirmed. Of course, this observation cannot be made until after all three of the other levels have been fully addressed, after the algorithm designed at the innermost level is implemented. Although this form of validation is usually qualitative, some influential work towards quantifying insight has been done [Saraiya et al. 05].

A more rigorous validation approach for this level is to observe and document how the target audience uses the deployed system as part of their real-world workflow, typically in the form of a longer-term field study. These field studies of deployed systems, which are appropriate for this level, can be distinguished from the exploratory pre-design field studies that investigate how users carry out their tasks before system deployment that are appropriate for the characterization level above.

2.3.3 Encoding and Interaction Threats

At the visual encoding and interaction design level, the threat is that the chosen design is not effective at communicating the desired abstraction to the person using the system. One immediate validation approach is to carefully justify the design with respect to known perceptual and cognitive

principles, as will be covered in Chapters 5 and 6. Methodologies such as heuristic evaluation [Zuk et al. 08] and expert review [Tory and Möller 05] are a way to systematically ensure that no known guidelines are being violated by the design.

A downstream approach to validate against this threat is to carry out a formal user study in the form of a laboratory experiment. A group of people use the implemented system to carry out assigned tasks, usually with both quantitative measurements of the time spent and errors made and qualitative measurements such as preference. The size of the group is chosen based on the expected experimental effect size in hopes of achieving statistically significant results.

Another downstream validation approach is the presentation of and qualitative discussion of results in the form of still images or video. This approach is downstream because it requires an implemented system to carry out the visual encoding and interaction specifications designed at this level. This validation approach is strongest when there is an explicit discussion pointing out the desirable properties in the results, rather than assuming that every reader will make the desired inferences by unassisted inspection of the images or video footage. These qualitative discussions of images sometimes occur in a case study format, supporting an argument that the tool is useful for a particular task-dataset combination.

A third appropriate form of downstream validation is the quantitative measurement of result images created by the implemented system. For example, many measurable aesthetic criteria such as number of edge crossings and edge bends have been proposed in the subfield of graph drawing, some of which have been empirically tested.

Informal usability studies do appear in Figure 2.5, but are specifically not called a validation method. As Andrews eloquently states: "Formative methods [including usability studies] lead to better and more usable systems, but neither offer validation of an approach nor provide evidence of the superiority of an approach for a particular context" [Andrews 08]. They are listed at this level because it is a very good idea to do them upstream of attempting a validating laboratory or field study. If the system is unusable, no useful conclusions can be drawn from these methods. This model distinguishes usability studies from informal testing with users in the target domain, as described for the level above. Although the informal testing with target users described at the level above may uncover usability problems, the goal is to collect anecdotal evidence that the system meets its design goals. In an informal usability study, the person using the system does not need to be in the target audience, the only constraint is that the user is not the system designer. Such anecdotes are much less convincing when they come from a random person rather than a member of the target audience.

2.3.4 Algorithm Threats

At the algorithm design level, the primary threat is that the algorithm is suboptimal in terms of time or memory performance, either to a theoretical minimum or in comparison with previously proposed algorithms. Obviously, poor time performance is a problem if the user expects the system to respond in milliseconds but instead the operation takes hours or days. The issue of matching system latency to user expectations is discussed in more detail in Section 6.2.2.

An immediate form of validation is to analyze the computational complexity of the algorithm. The downstream form of validation is to measure the wall-clock time and memory performance of the implemented algorithm. Again, the methodology for algorithm analysis and benchmark measurements is so heavily addressed in the computer science literature that it is not discussed further in this book.

Another threat is incorrectness at the algorithm level, where the implementation does not meet the specification for the visual encoding and interaction design that comes from the level above. The problem could come from poor algorithm design, or the implementation of the algorithm could have bugs like any computer program. Establishing the correctness of a computer program is a notoriously difficult problem, whether through careful testing or formal methods.

The threat of algorithm incorrectness is often addressed implicitly rather than explicitly within the visualization literature. Presenting still images or videos created by the implemented algorithm is a one form of implicit validation against this threat, where the reader of a paper can directly see that the algorithm correctness objectives have been met. Explicit qualitative discussion of why these images show that the algorithm is in fact correct is not as common.

2.3.5 Mismatches

A common problem in weak visualization projects is a mismatch between the level at which the benefit is claimed and the validation methodologies chosen. For example, the benefit of a new visual encoding cannot not be validated by wall-clock timings of the algorithm, which addresses a level downstream of the claim. Similarly, the threat of a mischaracterized task cannot be addressed through a formal laboratory study where the task carried out by the participants is dictated by the study designers, so again the validation method is at a different level than the threat against the claim. The nested model explicitly separates the visualization design problem into levels in order to guide validation according to the unique threats at each level.

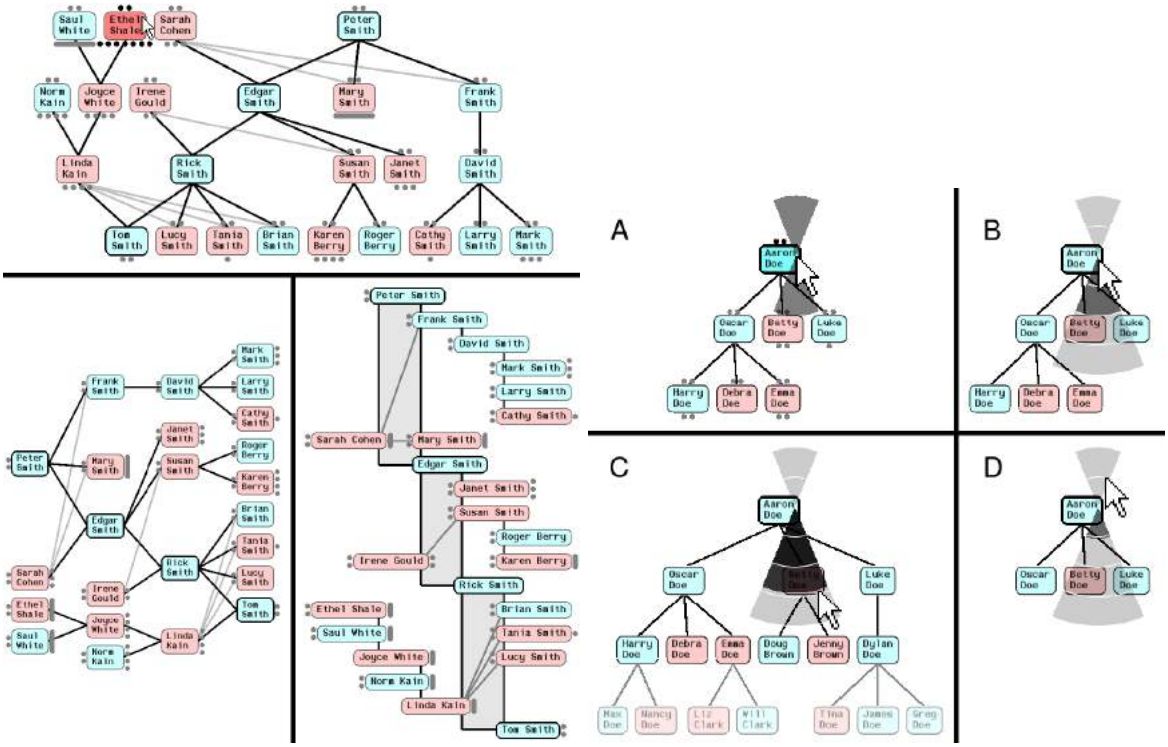


Figure 2.6: Genealogical graphs. (a) Three layouts for the dual-tree: classical node-link top-to-bottom at the top, classical left-to-right on the left, and the new indented outline algorithm on the right. (b) Widget for subtree collapsing and expanding with ballistic drags. From [McGuffin and Balakrishnan 05], Figures 13 and 14.

2.4 Example Gallery

This gallery contains examples of several visualization projects, analyzed according to the levels of visualization design that they target and the methods used to validate their benefits. These projects also provide a preview of the possibilities of visualization, as discussed in this book.

2.4.1 Genealogical Graphs

McGuffin and Balakrishnan present a system for the visualization of genealogical graphs [McGuffin and Balakrishnan 05], shown in Figure 2.6. They propose multiple new representations, including one based on the *dual-tree*, a subgraph formed by the union of two trees. Their prototype

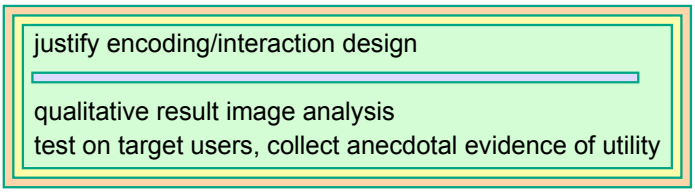


Figure 2.7: Genealogical graphs [McGuffin and Balakrishnan 05] validation levels.

features sophisticated interaction, including automatic camera framing, animated transitions, and a new widget for ballistically dragging out subtrees to arbitrary depths.

This exemplary paper explicitly covers all four levels. The first domain characterization level is handled concisely but clearly: their domain is genealogy, and they briefly discuss the needs of and current tools available for genealogical hobbyists. The paper particularly shines in the analysis at the second abstraction level. They point out that the very term *family tree* is highly misleading, because the data type in fact is a more general graph with specialized constraints on its structure. They discuss conditions for which the data type is a true tree, a multitree, or a directed acyclic graph. They map the domain problem of recognizing nuclear family structure into generic tasks about subgraph structure, and discuss the *crowding* problem at this abstract level. At the third level of our model, they discuss the strengths and weaknesses of several visual encoding alternatives, including using connection, containment, adjacency and alignment, and indentation. They present in passing two more specialized encodings, fractal node-link and containment for free trees, before presenting in detail their main proposal for visual encoding. They also carefully address interaction design, which also falls into the third level of our model. At the fourth level of algorithm design, they concisely cover the algorithmic details of dual-tree layout.

Three validation methods are used in this paper, shown in Figure 2.7. There is the immediate justification of encoding and interaction design decisions in terms of established principles, and the downstream method of a qualitative discussion of result images and videos. At the abstraction level, there is the downstream informal testing of a system prototype with a target user to collect anecdotal evidence.

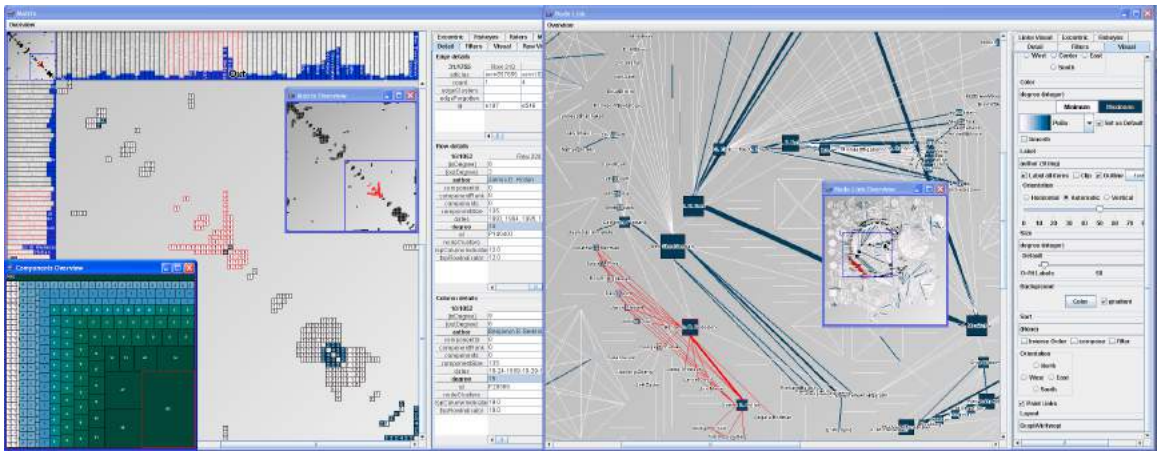


Figure 2.8: MatrixExplorer features both node-link and matrix representations in an interface designed for sociologists and historians to explore social networks. From [Henry and Fekete 06], Figure 1.

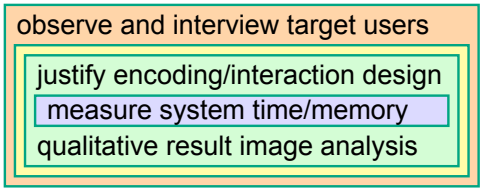


Figure 2.9: MatrixExplorer [Henry and Fekete 06] validation methods.

2.4.2 MatrixExplorer

Henry and Fekete present the MatrixExplorer system for social network analysis [Henry and Fekete 06], shown in Figure 2.8. Its design comes from requirements formalized by interviews and participatory design sessions with social science researchers. They use both matrix representations to minimize clutter for large and dense graphs, and the more intuitive node-link representations of graphs for smaller networks.

All four levels of the model are addressed, with validation at three of the levels, shown in Figure 2.9. At the domain characterization level, there is explicit characterization of the social network analysis domain, which is validated with the qualitative techniques of interviews and an exploratory study using participatory design methods with social scientists and other researchers who use social network data. At the abstraction level, the paper

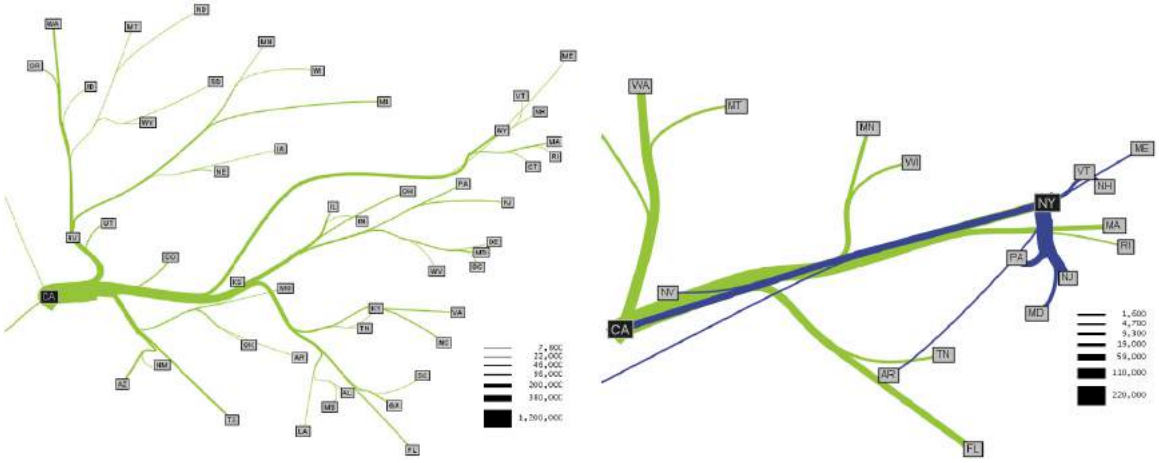


Figure 2.10: Flow maps showing migration patterns from 1995-2000 US Census data. (a) Migration from California. (b) The top ten states that migrate to California shown in green, and to New York in blue. From [Phan et al. 05], Figures 1c and 10.

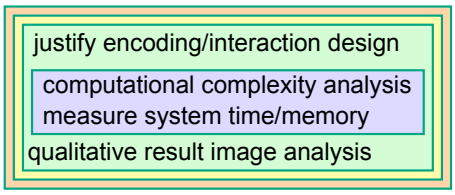


Figure 2.11: Flow Map [Phan et al. 05] validation methods.

includes a detailed list of requirements of the target user needs discussed in terms of generic tasks and data types. There is a thorough discussion of the primary encoding design decision to use both node-link and matrix views to show the data, and also of many secondary encoding issues. There is also a discussion of both basic interactions and interactive reordering and clustering support. In both cases the authors use the immediate validation method of justifying these design decisions. There is also an extensive downstream validation of this level using qualitative discussion of result images. At the algorithm level, the focus is on the reordering algorithm. Downstream benchmark timings are mentioned very briefly.

2.4.3 Flow Maps

Phan et al. propose a system for creating flow maps that show the movement of objects from one location to another, and demonstrate it on network traffic, census data, and trade data [Phan et al. 05]. Flow maps reduce visual clutter by merging edges, but most previous instances were hand drawn. They present automatic techniques inspired by graph layout algorithms to minimize edge crossings and distort node positions while maintaining relative positions, as shown in Figure 2.10.

This paper has a heavy focus on the innermost algorithm design level, but also covers the encoding and abstraction levels. Their analysis of the useful characteristics of hand-drawn flow maps falls into the abstraction level of our model. At the visual encoding level, they have a brief but explicit description of the goals of their layout algorithm, namely intelligent distortion of positions to ensure that the separation distance between nodes is greater than the maximum thickness of the flow lines while maintaining left-right and up-down ordering relationships. The domain characterization level is addressed more implicitly than explicitly: there is no actual discussion of who uses flow maps and why. However, the analysis of hand-drawn flow maps could be construed as an implicit claim of longstanding usage needs.

Three validation methods were used in this paper, shown in Figure 2.11. At the algorithm level, there is an immediate complexity analysis. There is also a brief downstream report of system timing, saying that all images were computed in a few seconds. There was also a more involved downstream validation through the qualitative discussion of result images generated by their system. In this case, the intent was mainly to discuss algorithm correctness issues at the innermost algorithm level, as opposed to addressing the visual encoding level.

2.4.4 LiveRAC

McLachlan et al. present the LiveRAC system for exploring system management time-series data [McLachlan et al. 08], as shown in Figure!2.12. It uses a reorderable matrix of charts with stretch and squish navigation combined with semantic zooming, so that the chart's visual representation adapts to the available space. They carry out an informal longitudinal field study of its deployment to operators of a large corporate web hosting service. Four validation methods were used in this paper, shown in Figure 2.13.

At the domain characterization level, the paper explains the roles and activities of system management professionals and their existing workflow and tools. The immediate validation approach was interviews with the

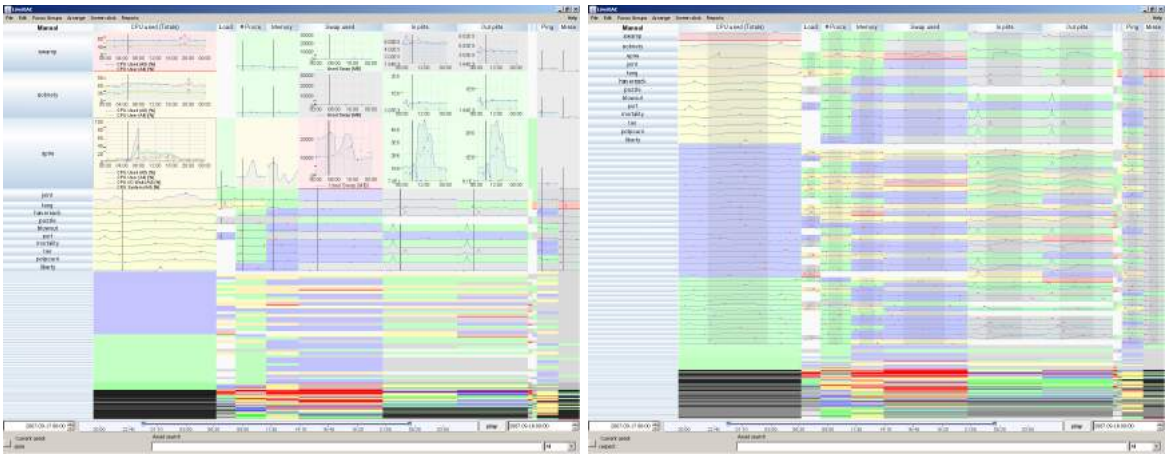


Figure 2.12: LiveRAC supports exploration of system management time-series data with a reorderable matrix and semantic zooming. (a) The first several dozen rows have been stretched out to show sparklines for the devices. (b) The top three rows have been enlarged more, so the charts appear in full detail. From [McLachlan et al. 08], Figure 3.

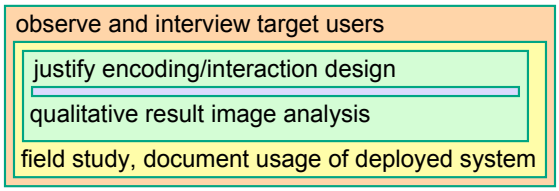


Figure 2.13: LiveRAC [McLachlan et al. 08] validation methods.

target audience. Their phased design methodology, where management approval was necessary for access to the true target users, makes our use of the word *immediate* for this validation a bit counterintuitive: many of these interviews occurred after a working prototype was developed. This project is a good example of the iterative process we allude to in Section 2.2.

At the abstraction level, the choice of a collection of time-series data for data type is discussed early in the paper. The rationale is presented in the opposite way from our discussion above: rather than justifying that time-series data is the correct choice for the system management domain, they justify that this domain is an appropriate one for studying this data type. The paper also contains a set of explicit design requirements, which includes abstract tasks like search, sort, and filter. The downstream validation for

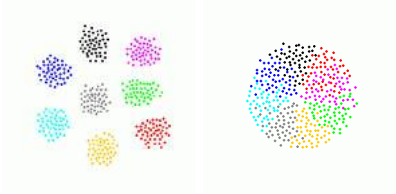


Figure 2.14: The LinLog energy model reveals clusters in node-link graphs. (a) LinLog clearly shows clusters with spatial separation. (b) The popular Fructerman-Reingold model for force-directed placement does not separate the clusters. From [Noack 03], Figure 1.

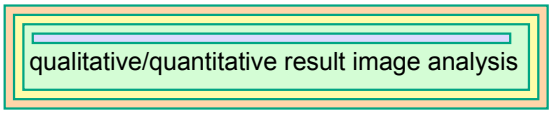


Figure 2.15: LinLog [Noack 03] validation methods.

the abstraction level is a longitudinal field study of the system deployed to the target users, life cycle engineers for managed hosting services inside a large corporation.

At the visual encoding and interaction level, there is an extensive discussion of design choices, with immediate validation by justification in terms of design principles. Algorithms are not discussed.

2.4.5 LinLog

Noack's LinLog paper introduces an energy model for graph drawing designed to reveal clusters in the data, where clusters are defined as a set of nodes with many internal edges and few edges to nodes outside the set [Noack 03]. Energy-based and force-directed methods are related approaches to graph layout, and have been heavily used in information visualization. Previous models strove to enforce uniform edge lengths as an aesthetic criterion, but Noack points out that to create visually distinguishable clusters requires long edges between them.

Although a quick glance might lead to an assumption that this graph drawing paper has a focus on algorithms, the primary contribution is in fact at the visual encoding level. The two validation methods used in the paper are qualitative and quantitative result image analysis, shown in Figure 2.15.

Noack clearly distinguishes between the two aspects of energy-based methods for force-directed graph layout: the energy model itself, versus

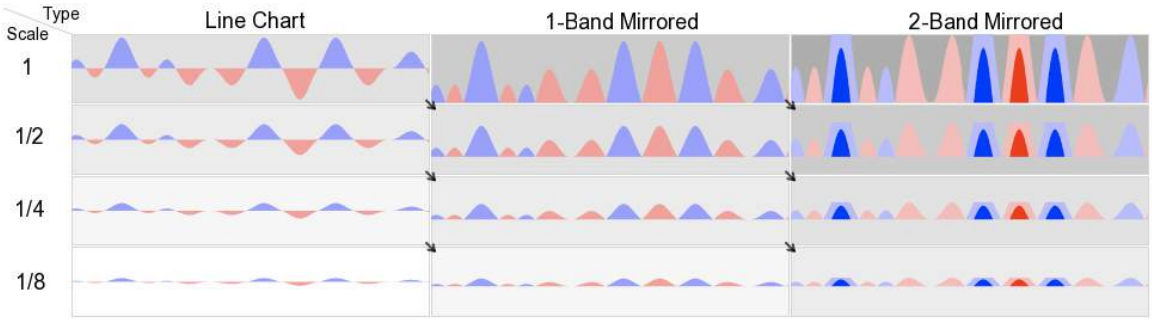


Figure 2.16: Experiment 2 of Sizing the Horizon compared filled line charts, 1-band horizon graphs, and 2-band horizon graphs of different sizes to find transition points where reducing chart height results in major drops in estimation accuracy across chart types. From [Heer et al. 09], Figure 7.

the algorithm that searches for a state with minimum total energy. In the vocabulary of our model, his LinLog energy model is a visual encoding design choice. Requiring that the edges between clusters are longer than those within clusters is a visual encoding using the visual channel of spatial position. One downstream validation approach in this paper is a qualitative discussion of result images, which is appropriate for a contribution at the encoding level. This paper also contains a validation method not listed in our model, because it is relatively rare in visualization: mathematical proof. These proofs are about the optimality of the model results when measured by quantitative metrics involving edge lengths and node distances. Thus, this model classifies it in the quantitative image analysis category, another appropriate method to validate at the encoding level.

This paper does not in fact address the innermost algorithm level. Noack explicitly leaves the problem of designing better energy-minimization algorithms as future work, using previously proposed algorithms to showcase the results of his model. The top domain characterization level is handled concisely but adequately by referencing previous work about application domains with graph data where there is a need to see clusters. For the second abstraction level, although the paper does not use the vocabulary of *task* and *data type*, it clearly states the abstraction that the intended task is finding clusters for the data type of a node-link graph.

2.4.6 Sizing the Horizon

Many laboratory studies are designed to validate and invalidate specific design choices at the visual encoding and interaction level by measuring

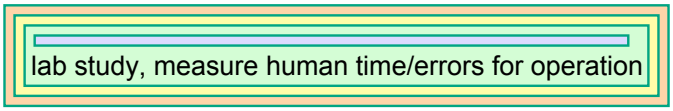


Figure 2.17: Lab studies as a validation method.

time and error rates of people carrying out abstracted tasks, as Figure 2.17 shows. Heer et al. compare line charts to the more space-efficient horizon graphs [Heer et al. 09], as Figure 2.16 shows. They identify transition points at which reducing the chart height results in significantly differing drops in estimation accuracy across the compared chart types, and find optimal positions in the speed-accuracy tradeoff curve at which viewers performed quickly without attendant drops in accuracy.

2.5 Problem-Driven vs. Technique-Driven Work

In **problem-driven** visualization, you begin by grappling with the problems of some real-world user and attempt to design a solution that helps them work more effectively. This style of work constitutes starting from the top level of the four-level model and working downwards. The type of visualization research paper typically used to report on this style of work is the **design study paper**; the LiveRAC and MatrixExplorer examples above are from this kind of paper. Often the problem can be solved using existing visual encoding and interaction techniques, and the challenge lies at the abstraction level. Sometimes the problem motivates the design of new techniques, when you decide that no existing ones will adequately solve the abstracted design problem.

In **technique-driven** work, your starting point is an idea for a new visual encoding or interaction technique, or a better algorithm for an existing one. In this style of work, you start within one of the two lower levels, and immediately focus on technique or algorithm design. Your interaction with the model is upward: your job is to articulate the assumptions and requirements of your technique in terms of the expected abstractions that it can handle. This kind of work is typically documented in a **technique paper**; the Genealogical Graphs, Flow Maps, and LinLog examples above are from this paper type.

The four-level model is useful for both, but in opposite directions. Similarly, the methods framework of Part IV can help you do both styles of work. For problem-driven work, it allows you to work downwards by searching for existing techniques by analyzing what methods are appropriate for

the abstraction that you have identified. For technique-driven work, it allows you to work upwards by classifying your proposed technique within the framework of methods, giving you a clear framework in which to discuss its relationship with previously proposed technique.

2.6 Further Reading

The four-level nested model of visualization design and evaluation was first presented as a paper [Munzner 09], which contains many more citations to the previous and related work. The partition of validation techniques according to levels was inspired in part by McGrath's research strategy circumplex [McGrath 94].

The literature on human-centered design is large; a very accessible starting point for practitioners is Kuniavsky [Kuniavsky 03], and a more academic one is [Sharp et al. 07]. Some of the challenges inherent in bridging the gaps between visualization designers and users are discussed by van Wijk [van Wijk 06]. Pretorius and van Wijk present good arguments that both data and task abstractions are important points of departure for information visualization designers [Pretorius and van Wijk 09].

Carpendale provides an excellent survey and overview of evaluation methods for visualization [Carpendale 08], including an extensive discussion of qualitative and ethnographically-inspired methods. Shneiderman and Plaisant also advocate working closely with domain users for multi-dimensional long-term in-depth case studies [Shneiderman 96].

Sedlmair et al. discuss the methodology of design studies and present a nine-stage model for conducting them [Sedlmair et al. 12].

Part II

Abstractions

This part of the book covers task and data abstractions, which are the second level of the nested model covered in Chapter 2.

Chapter 3 presents a taxonomy of tasks. TBD.

Chapter 4 presents a taxonomy of dataset and attribute types and semantics to give you a framework for thinking about data abstractions, and to establish a set of terms that will be used throughout the book. While there are many possible ways to think about data, this taxonomy matches up with the taxonomy of methods that are the building blocks for a visualization designer covered in Part IV. It also introduces the idea of transforming data by creating derived attributes. Data abstraction goes beyond simply analyzing the dataset that you are given; you can also change it into another form. There is a strong relationship between the form of the data and what you can do with it. The good news is that your hands are not tied as a designer because you can transform the data into a form useful for the task at hand. Don't just draw what you're given; decide what the right thing to show is, and draw that!

3

Task Abstractions

TODO

4

Data Abstractions

Many aspects of a visualization design are driven by the kind of data that we have at our disposal: what kind of data do we need to look at? What information can we figure out from the data itself, versus the meanings that we must be told explicitly? What high-level concepts will allow us to split datasets apart into general and useful pieces? What kind of attributes does the data have to begin with, and what kinds of derived data might we compute in order to draw a more effective picture?

This chapter approaches these questions with a taxonomy of types and semantics for data, which is split up into attributes and datasets. Of course, other approaches to data abstraction are possible; my taxonomy was chosen to mesh well with the principles in Part III and methods in Part IV.

The chapter begins with the distinction between types and semantics, and then presents attribute types and dataset types. It continues with attribute semantics and dataset semanatics. The chapter then covers data transformations and the idea of derived attributes, concluding with three example case studies.

4.1 Semantics vs. Types

Suppose that I give you the following data:

```
Basil, 7, S, Pear
```

What does this sequence mean? You can't know yet. These numbers and words could have many possible meanings. Maybe a food shipment of produce has arrived in satisfactory condition on the 7th of the month, containing basil and pears. Maybe the Basil Point neighborhood of the city has had 7 inches of snow cleared by the Pear Creek Limited snow removal service. Maybe the lab rat code-named Basil has made 7 attempts to find his way through the south section of the maze, lured by scent of the reward food for this trial, a pear.

To move beyond guesses, we need to know two crosscutting pieces of information about these terms: their semantics and their types. The **se-**

mantics of the data is its real-world meaning. For instance, does a word represent a human first name, or is it the shortened version of company name where the full name can be looked up in an external list, or is it a city, or is it a fruit? Does a number represent a day of the month, or an age, or a measurement of height, or a unique code for a specific person, or a code for a neighborhood?

The **type** of the data is its mathematical interpretation. If a number represents a quantity like a count of boxes of detergent, then adding two of these numbers together makes sense. If it is simply a numeric code representing a neighborhood, then addition is not a meaningful concept.

In short, I define **types** as information that could be directly inferred from the dataset structure itself, whereas **semantics** is meaning that must be provided along with the dataset in order for it to be interpreted correctly.

Name	Age	Shirt Size	Favorite Fruit
Amy	8	S	Apple
Basil	7	S	Pear
Clara	9	M	Durian
Desmond	13	L	Elderberry
Ernest	12	L	Peach
Fanny	10	S	Lychee
George	9	M	Orange
Hector	8	L	Loquat
Ida	10	M	Pear

Table 4.1: The approximate semantics of the dataset would easier to guess with information about many rows of the table; the exact semantics is given by the column titles.

Table 4.1 shows several more lines of the same dataset. This simple example table is tiny, with only 9 rows and 4 columns. The approximate semantics are easier to guess with many example values. Each row represents a person. The first column gives the name; the second column is a number; the third column looks like size in terms of small, medium, or large; and the last column is a kind of fruit. The exact semantics needs to be provided by the creator of the dataset; I give it with the column titles. In this case, for each person we have a name, their age, their shirt size, and their favorite fruit. (The people are the unfortunate Gashlycrumb Tinies whose fates were documented by Edward Gorey [Gorey 63].)

The question that remains is how to categorize attribute types in a way that is be general enough to cover all possible semantics, yet specific enough to be helpful for later visual encoding stage.

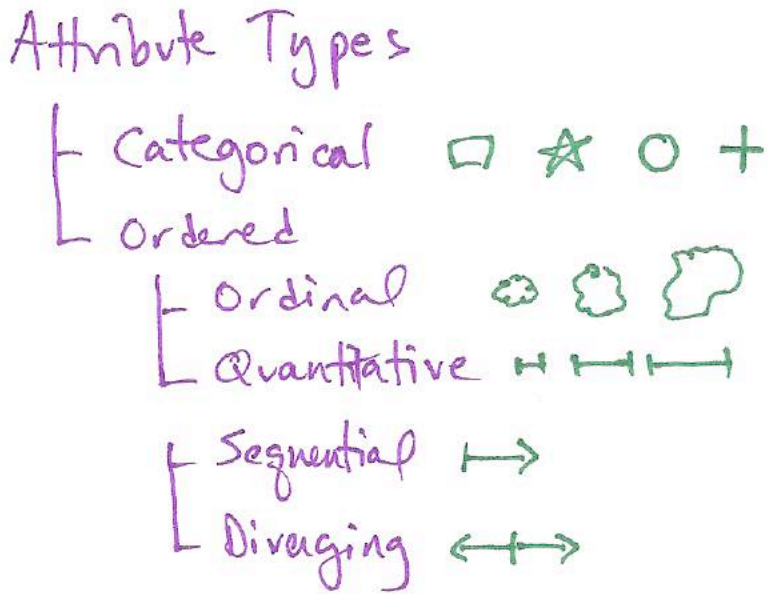


Figure 4.1: The taxonomy of attribute types.

4.2 Attribute Types

Figure 4.1 shows the taxonomy of attribute types, where the major distinction is between categorical versus ordered, and within the ordered type between ordinal versus quanitative. This section also covers the number of attributes and their hierarchical structure.

4.2.1 Categorical

The first distinction is between categorical and ordered data. **Categorical** data, such as favorite fruit or names, does not have an implicit ordering, but it often has hierarchical structure. Categories can only distinguish whether two things are the same (apples) or different (apples vs. oranges). Of course, any arbitrary external ordering can be imposed upon categorical data. Fruit could be ordered alphabetically according to its name, or by its price – but only if that auxiliary information were available. However, these orderings are not implicit in the attribute itself, the way they are with quantitative or ordered data. Other examples of categorical attributes are movie genres, file types, and city names.

4.2.2 Ordered: Ordinal and Quantitative

All **ordered** data does have an implicit ordering. This category can be further subdivided. With **ordinal** data, such as shirt size, we cannot do full-fledged arithmetic, but there is an well-defined ordering. For example, Large minus Medium is not a meaningful concept, but we know that Medium falls between Small and Large. Rankings are another kind of ordered data, for example when people create top-ten lists of movies or initial lineups for sports tournaments depending on past performance.

Quantitative data, such as age, is a meaningful numerical magnitude and we can do arithmetic on it. For example, the quantity of 68 inches minus 42 inches is 26 inches. Other examples of quantitative data are height, weight, temperature, stock price, number of calling functions in a program, and number of drinks sold at a coffee shop in a day. Both integers and real numbers are considered quantitative data.[1]

In this book, the ordered type includes both the ordinal and quantitative subtypes and is used often, in contrast to the unordered categorical type. The ordinal type is only occasionally mentioned, when the distinction between it and the quantitative type matters.

Ordered data can be either **sequential**, where there is a homogeneous range from a minimum to a maximum value, or **diverging**, which can be deconstructed into two sequences pointing in opposite directions that meet at a common zero point. For instance, a mountain height dataset is sequential, when measured from a minimum point of sea level to a maximum point of Mount Everest. A bathymetric dataset is also sequential, with sea level on one end and the lowest point on the ocean floor at the other. A full elevation dataset would be diverging, where the values go up for mountains on land and down for undersea valleys, with the zero value of sea level being the common point joining the two sequential datasets.

4.2.3 Attribute Count

The number of attributes that need to be visually encoded is a critical question when designing a visualization. Techniques that work for a dataset with a few attributes will typically fail for a datasets with dozens or hundreds of attributes. Section **??** covers methods for reducing the number of attributes to show.

[1] In some domains the quantitative category is further split into interval vs. ratio data [Stevens 46]; this distinction is typically not useful when designing a visual encoding, so in this taxonomy these types remain collapsed together into this single category.

4.2.4 Hierarchical Attributes

There may be hierarchical structure within an attribute or between multiple attributes. The daily stock prices of companies collected over the course of a decade is an example of a time-series dataset, where one of the attributes is time. In this case, time can be aggregated hierarchically, from individual days up to weeks up to months up to years. There may be interesting patterns at multiple temporal scales, such as very strong weekly variations for weekday vs. weekend, or more subtle yearly patterns showing seasonal variations in summer vs. winter. Many kinds of attributes might have this sort of hierarchical structure: for example, the geographic attribute of a postal code can be aggregated up to the level of cities or states or entire countries. Section 9.2 covers hierarchical aggregation in more detail, and Section **??** covers the visual encoding of attribute hierarchies.

4.3 Dataset Types

Figure 4.2 shows the three dataset types in this taxonomy: tables, networks, and text. For any of these three types, the full dataset could be available immediately in the form of a file, or it might be dynamic data processed gradually in the form of a stream.

4.3.1 Tables

Many datasets come in the form of **tables** that are made up of rows and columns, a familiar form to anybody who has used a spreadsheet. For simple flat tables, the vocabulary of this book is that each row represents an **item** of data, and each column is an **attribute** of the dataset.[2]

For example, a table representing the transactions at a grocery store might contain 109 rows, where each row represents a purchase, and 4 columns: the date, the location code for the store where it was bought, the price, and the method of payment. A table representing a medical scan might contain 4 columns: three numbers that indicate a position in 3D space, and a fourth number indicating the density of tissue at that point. A low-resolution scan might have 262,144 rows, providing information about a cubical volume of space with 64 bins in each direction.

The discussion in this chapter focuses on the table as a data type, not the visual encoding in chart form of a grid of information.

[2]Vocabulary note: One common synonym for attribute is *data dimension*, or just *dimension* for short; since this term is highly overloaded, in this book it is reserved for the visual channels of spatial position as discussed in Section 7.1.

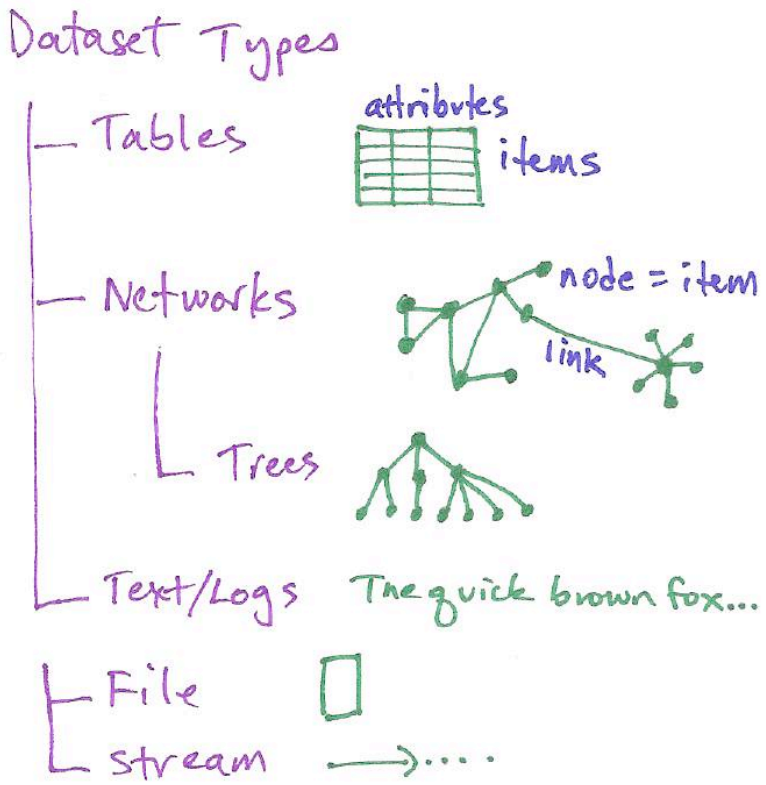


Figure 4.2: The taxonomy of data types.

4.3.2 Networks and Trees

Another dataset type is the **network**, which is well suited for specifying that there is some kind of relationship between two or more items.[3] An item in a network is also called a **node**.[4] A **link**, sometimes called an **edge**, is a relation between two items. For example, in an articulated social network the nodes are people, and links mean friendship. In a gene interaction network, the nodes are genes, and links between them mean that these genes have been observed to interact with each other. In a computer

[3]Vocabulary note: A synonym for network is **graph**. The word *graph* is also deeply overloaded in visualization. Sometimes it is used to mean *network* as we discuss here, for instance in the visualization subfield called *graph drawing* or the mathematical subfield called *graph theory*. Sometimes it is used in the field of statistical graphics to mean *chart*, as in bar graphs and line graphs.

[4]Vocabulary note: A synonym for node is **vertex**.

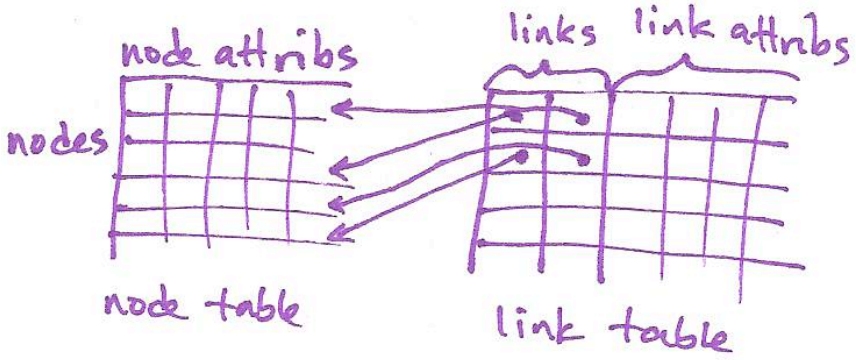


Figure 4.3: Networks can be represented with two tables, with separate attributes for nodes and links.

network, the nodes are computers, and the links represent the ability to send messages directly between two computers using physical cables or a wireless connection.

Network nodes can have associated attributes, just like items in a table. In addition, the links themselves could also be considered to have attributes associated with them, that might be partly or wholly disjoint from the node attributes. Thus, one way to specify a network is through two tables. One is a table of nodes where each row represents the nodes as items complete with their associated attributes. The other is a table of links, where each link is an item along with its associated attributes, and the link item specifies the connection by referring to two of the rows in the node table. Figure 4.3 shows this structure.

It is important to distinguish between the abstract concept of a network, and any particular visual layout of that network where the nodes and edges have particular spatial positions. In this chapter we concentrate on the abstraction, and return to questions of layout in Chapter 11.2.

A **tree** is a specific kind of network with a hierarchical structure, where there are no cycles because there is only one parent node pointing to any child node. One example of a tree is the organization charts a company, showing who reports to whom.

4.3.3 Text and Logs

A text **document** can also be considered to be a dataset. A document can be considered an ordered list of words at the lowest level, and at one level higher it is an ordered list of sentences. Documents often have hierarchical structure in terms of sections and subsections. Although there is an implied

linear ordering of the words within a document, it can also be treated simply as a bag of words where that ordering is ignored. This approach is common when dealing with large **document collections**, where the goal might be to analyze the differential distribution of words between the documents, or finding clusters of related documents based on common word use between them.

Log files are another form of document where there is a mix of free-form and structured text, but they are primarily designed for machine readability rather than the human reading experience. In a log file the basic unit of analysis is usually a line, rather than a sentence as in a prose file.

This chapter does not go into further detail about this data type, which will be covered more thoroughly in Section 11.4. It is included in this taxonomy so that it is clear that it can be transformed into one of the other dataset types, tables or networks, as we will discuss in Section 4.6.

4.3.4 Static Files vs. Dynamic Streams

The default approach to visualization assumes that the entire dataset is available all at once, as a **static file**. However, some datasets are instead **dynamic streams**, where the dataset information trickles in over the course of the visualization session. [5] One kind of dynamic change is to add new items or delete previous items. Another is to change the values of existing items.

This distinction crosscuts the other three dataset types of tables, networks, and text; any of the three can be static or dynamic. Designing for streaming data adds complexity to many aspects of the visualization process that are straightforward when there is complete dataset availability up front.

4.4 Attribute Semantics

Knowing the type of an attribute does not tell us about its semantics, because these two questions are crosscutting. There are many different approaches to considering the semantics of attributes that have been proposed across the many fields where these semantics are studied. The data abstraction framework proposed in this book is heavily focused on keys vs. values and spatial vs. nonspatial semantics, because of the structure of the methods framework in Part IV. Two additional questions are whether an attribute is temporal, and whether it is continuous vs. discrete. Figure 4.4 summarizes this taxonomy.

[5] Vocabulary note: A synonym for dynamic is **online**.

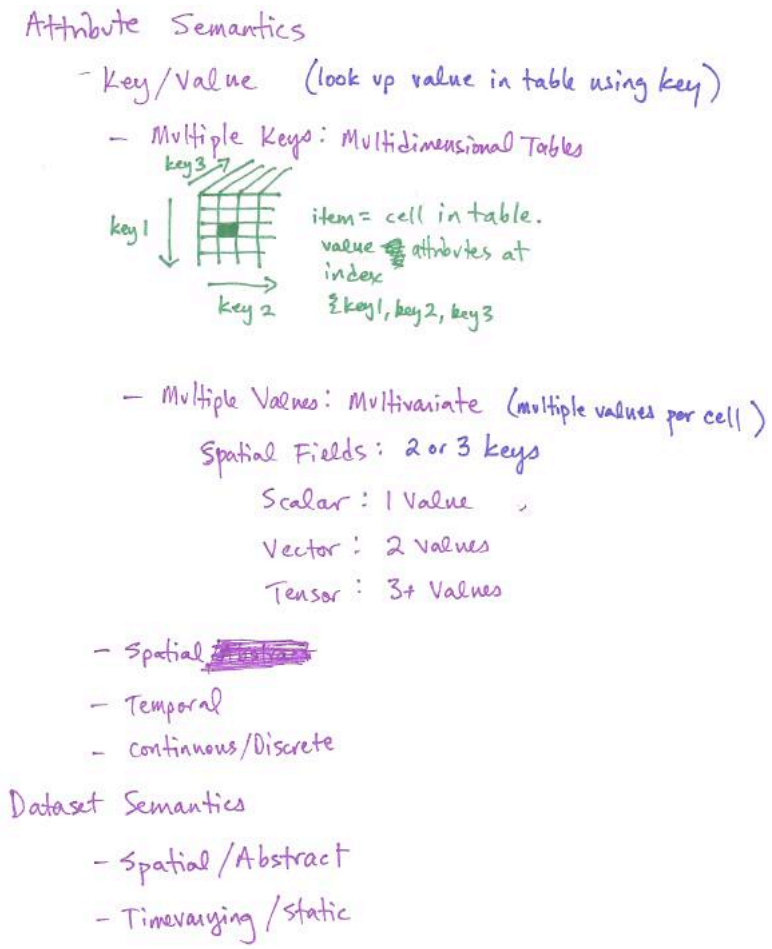


Figure 4.4: The taxonomy of attribute and dataset semantics.

4.4.1 Key vs. Value

Many tables have both **key** attributes and **value** ones, where a key acts as a unique index for an item used to look up its value in the table. [6] A simple **flat table** has only one key, where each item corresponds to a row in the table. More complex tables require multiple keys to look up an item, as shown in Figure 4.4. These tables are called **multidimensional**

[6] Vocabulary note: A synonym for key attribute is independent attribute. A synonym for value attribute is dependent attribute. The language of key/value is common in the database field, while the language of independent/dependent is common in statistics.

tables in the language of computer science. A multidimensional table can be reshaped into flat form by decomposing it into a set of single-key tables and combining all of the resulting columns together, but this transformation loses information about the dataset semantics that is often crucial for the user's task. Of course, this information may not be available; in many instances determining which attributes are independent keys versus dependent values is the goal of the visualization process, rather than its starting point.

The combination of all keys must be unique for each item. Thus, quantititive attributes are not usually keys, because they often have the same values for multiple items. Keys can be categorical or ordered, but not all such attributes are keys because duplicate values are possible. For example, in Table 4.1, `Name` is a categorical attribute that appears to be a key, but `Favorite Fruit` is clearly not a key despite being categorical because `Pear` appears in two different rows. The quantitative attribute of `Age` has many duplicate values, as does the ordered attribute of `Shirt Size`.

Two common keys are space and time. In spatial fields, spatial position is always a key. For example, a measurement would be taken at regular spatial intervals, so the spatial position of the measurement is used to look up the value. In both spatial fields and abstract tables, time can also be an independent key. Time is an example of a key attribute that is ordered rather than categorical. A dataset is said to have **timevarying** semantics when each item has values at many points in time. For example, a timevarying medical scan would have the independent keys of x, y, z, t to cover spatial position and time, with the dependent value attribute of density for each combination of four indices to look up position and time. This dataset can be considered as a multidimensional table with four indices and one value at each table cell.

Spatial fields in particular are typically characterized in terms of their **multivariate** structure; that is, the number of dependent value attributes. A **scalar** field is univariate: has a single value attribute at each point in space, for example the preceding example or the temperature in a room at each point in 3D space. A **vector** field is bivariate with two attributes values at each point, for example the velocity of air in the room: there is a direction and speed for each item. A **tensor** field has three or more value attributes for each item. A physical example is stress, which is force per unit area. The force is one vector quantity that has a direction, and the area is described by another vector, a normal. The geometric intuition is that each point in a scalar field has a single value. Each point in a vector field has length and direction, like an arrow. Each point in a tensor field has a more complex geometry, like an ellipsoid. This categorization of spatial fields requires knowledge of the attribute semantics, not simply attribute types. If we are given a field with multiple measured values at each point

and no further information, there is no sure way to know its structure. For example, 9 values could represent many things: 9 separate scalar fields, or 4 vector fields plus a scalar field, or a single tensor field. [7]

4.4.2 Spatial vs. Nonspatial

One key question is whether a dataset has attributes specifying the spatial position of each item. Some datasets are inherently spatial, such as 3D medical scans of a human body as in the table discussed above. When creating a visual encoding using this data, it usually makes sense to show something that spatially looks like an X-ray image of the human body, and use color coding to highlight suspected tumors. Another example is measurements made in a real or simulated wind tunnel of the temperature and pressure of air flowing over airplane wings at many points in 3D space. One possible visual encoding would use the geometry of the wing as the spatial substrate, showing the temperature and pressure using size-coded arrows. Many datasets contain spatial position attributes in the form of geographic information, for example the locations of stores where items were purchased. A geographic view of the dataset would use a map containing all of those locations as the spatial backdrop for the display, and then could show additional information with marks at the spots of interest. Spatial data often has a multiscale hierarchical structure already, or a hierarchy can be imposed upon it through aggregation.

The specific name for datasets where spatial position is an attribute of primary importance is **spatial fields**. More generally, any dataset with attributes that have spatial position semantics are called **spatial** datasets. In contrast, many datasets are completely abstract. These datasets, where no attributes have spatial position semantics, are called **nonspatial** or **abstract** datasets.

4.4.3 Temporal vs. Nontemporal

A specific attribute might record some temporal data value for each item, such as a duration of elapsed time or specific timepoint of occurrence. Time is complicated to handle because of both the rich hierarchical structure that we use to reason about it, and the potential for periodic structure. The time hierarchy is deeply multiscale: the scale of interest could range anywhere from nanoseconds to hours to decades to millennia. Even the common words *time* and *date* are a way to partially specify the scale of

[7] Vocabulary note: Spatial fields with multiple dependent value attributes are called **multivariate**; when they have multiple independent key attributes they are called **multidimensional**. Thus, a multidimensional multivariate field has multiple keys and multiple values.

temporal interest. Temporal analysis tasks often involve finding or verifying periodicity either at specific scales or at some scale not known in advance. Moreover, the temporal scales of interest do not all fit into a strict hierarchy; for instance, weeks do not fit cleanly into months.

Thus, the generic visualization problems of transformation and aggregation are often particularly complex when dealing with temporal data. Section 4.6 below introduces the problem of data transformation, and Section 9.2 discusses the question of aggregation in detail.

4.4.4 Continuous vs. Discrete

Some attributes represent a continuous quantity: for example, the quantitative density values in the 3D medical scan dataset are samples taken at many points in space. If the points in space were closer together, then more information about the phenomenon under study would be available. Continuous data requires careful treatment to avoid presenting the user with misleading artifacts, such as might occur by naively interpolating between values at neighboring positions. The field of signal processing covers the mathematics of sampling, interpolation, and reconstruction in detail. Spatial fields typically have continuous attributes. The visualization subfield of volume rendering grapples extensively with these issues.

Other attributes represent a discrete quantity: for example, the categorical attribute of favorite fruit in Table 4.1. It makes no sense to interpolate between values between durian and elderberry. Abstract datasets often have discrete attributes. Many subfields of mathematics focus on working with discrete data, including graph theory and combinatorics.

4.5 Dataset Semantics

Given the semantics of attributes, thee discussion can now continue with the semantics at the level of the entire dataset. The two main splits are into spatial versus abstract datasets, and static versus timevarying ones, as shown in Figure 4.4.

4.5.1 Spatial vs. Abstract

As we defined above, a dataset has **spatial semantics** if any of its attributes are spatial, and has **abstract semantics** when none of its attributes are spatial. This definition is straightforward on the surface, but has deep implications.

It's not immediately obvious how to make a picture of these abstract datasets. If your job is to inspect a database of sales records and find the

inventory items that are selling poorly, what kind of picture should you make? That is, which aspect of the data should be the spatial substrate, when the spatial structure isn't dictated by the dataset itself?

As will be discussed in Chapters 5 and 8, at the visual encoding stage a visual representation must be chosen that maps some nonspatial attributes of the dataset to spatial position in the drawing. For abstract datasets, the choice of which attributes to map to spatial position used in the visual encoding is chosen by the designer rather than given implicitly in the semantics of the dataset itself. This choice of spatialization is the one of the most central and difficult problems of visualization design. One of the most fundamental dataset semantics distinctions is thus whether the dataset is a spatial field or is abstract.

4.5.2 Timevarying vs. Static

Another central division in terms of dataset semantics is whether it is **timevarying**, representing something that is changing over time. Although any dataset with spatial attributes is considered to have spatial semantics, the situation is not so clear with time. Usually the dataset is considered timevarying when the temporal attribute is independent and thus more central, but not if the temporal attribute is dependent. In these cases, the temporal aspects could be considered peripheral. As with decisions about semantics, this decision requires external knowledge about the nature of the dataset and cannot be made purely from data type information. One example of a dataset with timevarying semantics would be one created with a sensor network that tracks the location of each animal within a herd by taking new measurements every second. Each animal will have new location data at every time point, so the temporal attribute is central and independent.

It is possible for a dataset to have attributes that deal with temporal information but not have timevarying semantics. For example, a horseracing dataset covering a year's worth of races could have attributes such as race start time, or elapsed time since the current frontrunner took the lead in a particular race. These attributes do indeed deal with temporal information, but the dataset is not time-varying. Also, note that this distinction between timevarying vs. semantic dataset *semantics* is different from the distinction of dataset *types* in terms of static files vs. dynamic streams, as discussed in Section 4.3.4.

One important idea is that even though the dataset semantics involves change over time, there are many approaches to visually encoding that data – and only one of them is to show it changing over time in the form of an animation. These issues are discussed further in Section 7.2.

A common special case is called a **time series dataset**, namely an

ordered sequence of time-value pairs. This dataset fits into the taxonomy of types as a single-index table. These time-value pairs are often but not always spaced at uniform temporal intervals. Typical time-series analysis tasks involve finding trends, correlations, and variations at multiple time scales such as hourly, daily, weekly, and seasonal.

4.6 Derived and Transformed Data

The dataset as given to the visualization designer is not the final story: the original attributes can be transformed into new derived attributes, which can be used to create derived spaces.

4.6.1 Derived Attributes

A dataset often needs to be transformed beyond its original state in order to create a visual encoding that can solve the desired problem. To do so, we can create a **derived attribute** from the original set provided with the dataset.

In some cases, the derived attribute encodes the same data as the original, but with a change of type. For example, a dataset might have an original attribute that is quantitative data: for instance, floating point numbers that represent temperature. For some tasks, like finding anomalies in local weather patterns, that raw data might be used directly. For another task, like deciding whether water is an appropriate temperature for a shower, that quantitative attribute might be transformed into new derived attribute that is ordered: hot, warm, or cold. In this transformation, most of the detail is aggregated away. In a third example, when making toast, an even more lossy transformation into a binary categorical attribute might suffice: burned, or not burned. A fourth example is to transform from sequential to diverging: when Goldilocks makes toast, temperatures could range from underdone to overdone, with a point at the middle that is just right.

In other cases, creating the derived attribute requires access to additional information. In a geographic example, an original dataset with the categorical attribute of city name, could be transformed into one with two more derived quantitative attributes for the latitude and longitude of the city. This transformation could be accomplished through a lookup to a separate, external database.

A new derived attribute may be created using arithmetic, logical, or statistical operations ranging from simple to complex. A very common simple and local operation is subtracting two quantitative attributes to create a new quantitative difference attribute, which can then be directly visually

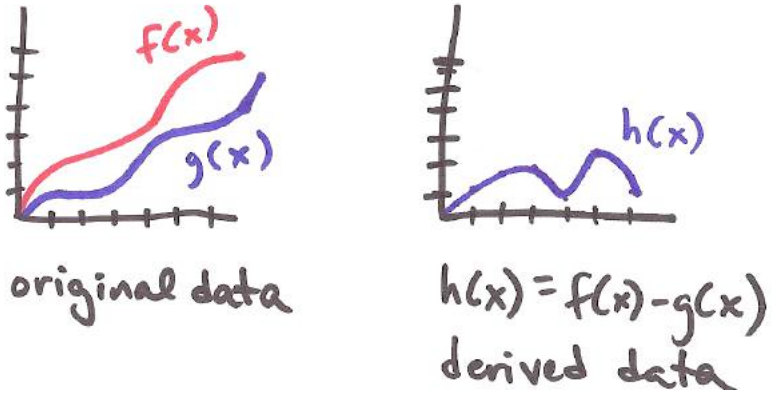


Figure 4.5: Derived attributes can be directly visually encoded. (a) Two original data attributes are plotted. (b) The quantitative derived attribute of the difference between the two originals can be plotted directly.

encoded. Figure 4.5 shows an example of encoding two attributes directly, versus encoding the derived variable of the difference between them. For tasks that require understanding this difference, directly encoding it allows the user to internalize the information directly using the perceptual system, rather than requiring the user to do internal computations of the difference in heights between the two original curves at each step along the way. Many operations require global computations across all values for an attribute, such as averaging for a single attribute or the correlation between two of them.

The data transformation may involve multiple stages. For example, a table of genomics data can be transformed into a network by first creating a quantitative derived attribute of similarity, and then creating a network with links only between the most similar items [Davidson et al. 01]. In this case, the table had 6000 rows of yeast genes, and 18 columns containing measurements of the gene activity level in a specific experimental condition. The values in the columns were used to derive a new attribute, the similarity score, defined between each pair of genes. The similarity score was computed using sophisticated statistical processing to be robust in the presence of nonlinear and noisy data, as occurs in this sort of biological application. This derived attribute was then used to create a network, where the nodes in the network were genes. A link was established between two genes when the similarity score was high; specifically, links were created only for the top 20 similarity scores.

4.6.2 Derived Spaces

A **derived space** is any data abstraction that includes derived attributes. A derived space may have only derived attributes, or a mix or original and derived ones. That is, calling a dataset abstraction a derived space simply emphasizes that the abstraction chosen by the visualization designer is a transformation of an original dataset, rather than the untransformed original data.

This section illustrates the power of derived attributes and spaces with three detailed examples. Thus far, this chapter has presented a taxonomy of dataset and attribute types and semantics. The examples below are a first look at how this taxonomy is used in the abstraction step of visualization design. The examples below include discussion not only at the abstraction level, but also at the visual encoding level, so that the derived spaces are shown in a concrete way with images.

4.6.2.1 Strahler Numbers for Trees. In a visualization showing a complex network, it is useful to be able to filter out most of the complexity by drawing a simpler picture that communicates the key aspects of the network structure. A good way to support this kind of simplification is to calculate a single number at each node in the graph that approximates its **importance**. The Straher number is an approach to calculating node importance based on network topology. It was originally developed in hydrogeology to characterize the branching structure of rivers, and has been extended for use in abstract trees and networks for visualization purposes [Auber 02]. Very central nodes have large Strahler numbers, whereas peripheral nodes have low values for this derived attribute. Strahler numbers are just one example of a **centrality metric** that measures the importance of a node within a network. They are also an example of a derived attribute for network data that is the result of a complex and global computation.

Figure 4.6 shows an example where Strahler numbers are used in an importance-based rendering approach. The 5000 nodes with the highest Strahler numbers are drawn, along with the connections between them. The result is recognizeable as a good skeleton of the full tree shown on the right, which has over half a million nodes. In contrast, if the 5000 nodes to draw were picked randomly, the structure would not be understandable. Both versions of the network are also colored according to the Strahler number, to show how the centrality measure varies within the network.

4.6.2.2 Feature Detection in Fluid Dynamics. Data transformations can shed light into spatial data as well as nonspatial data. In an example from computational fluid dynamics, linked derived spaces are used for feature detection [Henze 98]. The visualization system allows the user to quickly create plots of any two original or derived variables from the palette of

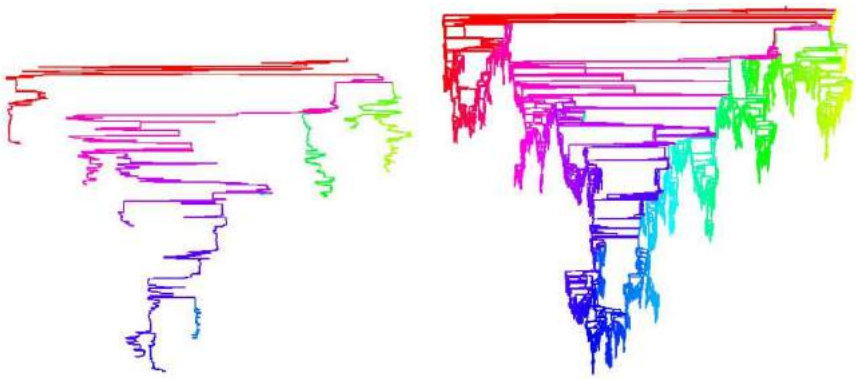


Figure 4.6: Strahler numbers are a derived attribute that can be computed to find the centrality of nodes in a graph. (a) The major structure of a large tree is visible when only 5000 of the highest-ranked nodes are drawn. (b) The full tree has over a half-million nodes. From [Auber 02], Figures 10 and 13.

variables shown in Figure 4.7(a). The views are linked together with color highlighting. The power of this technique lies in seeing where regions that are contiguous in one view fall in the other views; we discuss it further in Chapter 8.5.

The original dataset is a time-varying spatial field with measurements along a curvilinear mesh fitted to an airfoil. The plot in Figure 4.7(c) shows the data in this physical space, using the two spatial field variables. Undisturbed airflow enters the physical space from the left, and the back tip of the airfoil is on the right. Two important regions in back of the airfoil are distinguished with color: a red recirculation region and a yellow wake region. While these regions are not easy to distinguish in this physical view, they can be understood and selected by more easily by interaction with the four other derived views. For example, in the derived space of Figure 4.7(b), the recirculation zone is distinguishable as a coherent spatial structure at the top, with the yellow wake also distinguishable beneath it. As the white box shows, the recirculation zone can easily be selected in this view. Figure 4.7(e) shows another derived space made by plotting the pressure versus the temperature. In this view, the red recirculation zone and the yellow wake appear where both the pressure and temperature variables are high, in the upper right. Without getting into the exact technical meaning of the derived variables as used in fluid dynamics (vorticity, entropy, enthapy, and so on), the point of this example is that many structures of interest in fluid dynamics can be seen more easily from layouts in

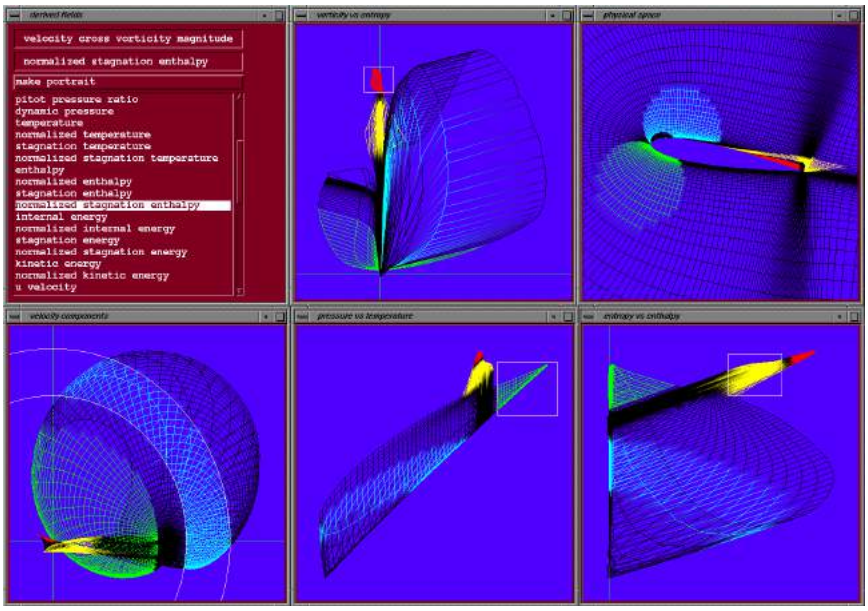


Figure 4.7: In this computational fluid dynamics visualization system, views linked by color highlighting can be created for any combination of the original spatial field variables and derived attributes. From [Henze 98], Fig 5.

the derived spaces.

4.6.2.3 Graph-Theoretic Scagnostics. Graph-theoretic scagnostics is a scalable technique for the exploration of scatterplot matrices [Wilkinson et al. 05, Wilkinson et al. 06]. A single scatterplot allows direct comparison between two dimensions. A scatterplot matrix, or **SPLOM** for short, is the systematic way to compare all possible pairs of dimensions, with the dimensions ordered along both the rows and the columns and one scatterplot at each cell of the matrix. Figure 4.8 shows an example SPLOM for a nine-dimensional dataset of abalone measurements. Interactive SPLOM exploration is supported by linked highlighting of points between views, so selecting a point in one plot shows where it falls in all other plots. SPLOMs are in the family of small multiples multiple-view techniques, as we will discuss in Chapter 8.5.

Although the matrix is symmetric, where the upper triangle shows the same information as the lower triangle, the size of the matrix still grows quadratically with the number of dimensions N: $N^2/2$ cells are needed.

Each individual plot requires enough screen space to distinguish the points within it, so this method does not scale well past a few dozen dimensions.

The idea of **scagnostics**, short for scatterplot computer-guided diagnostics, is to identify a small number of measurements that nicely categorize the shape of the point distributions within each scatterplot. The nine measures are *outlying* for outlier detection; *skewed*, *clumpy*, *sparse*, and *striated* for point distribution and density; *convex*, *skinny* and *stringy* for shape, and *monotonic* for the association. Figure 4.9 shows examples of real-world datasets rated low, medium, and high with respect to each of these measures.

These measurements are then shown in a new scagnostics SPLOM that is a scatterplot of scatterplots. That is, each point in the scagnostics SPLOM represents an entire scatterplot in the original SPLOM, which is shown when the point is selected. Figure 4.10 shows the scagnostics matrix for the abalone dataset. The idea is that the distribution of points in the scagnostics SPLOM should provide a fast overview of the most important characteristics of the original SPLOM, because the measures have been carefully chosen to reflect scatterplot shape. Looking at the outliers in this scagnostics SPLOM guides the user to the unusually shaped, and thus potentially interesting, scatterplots in the original SPLOM.

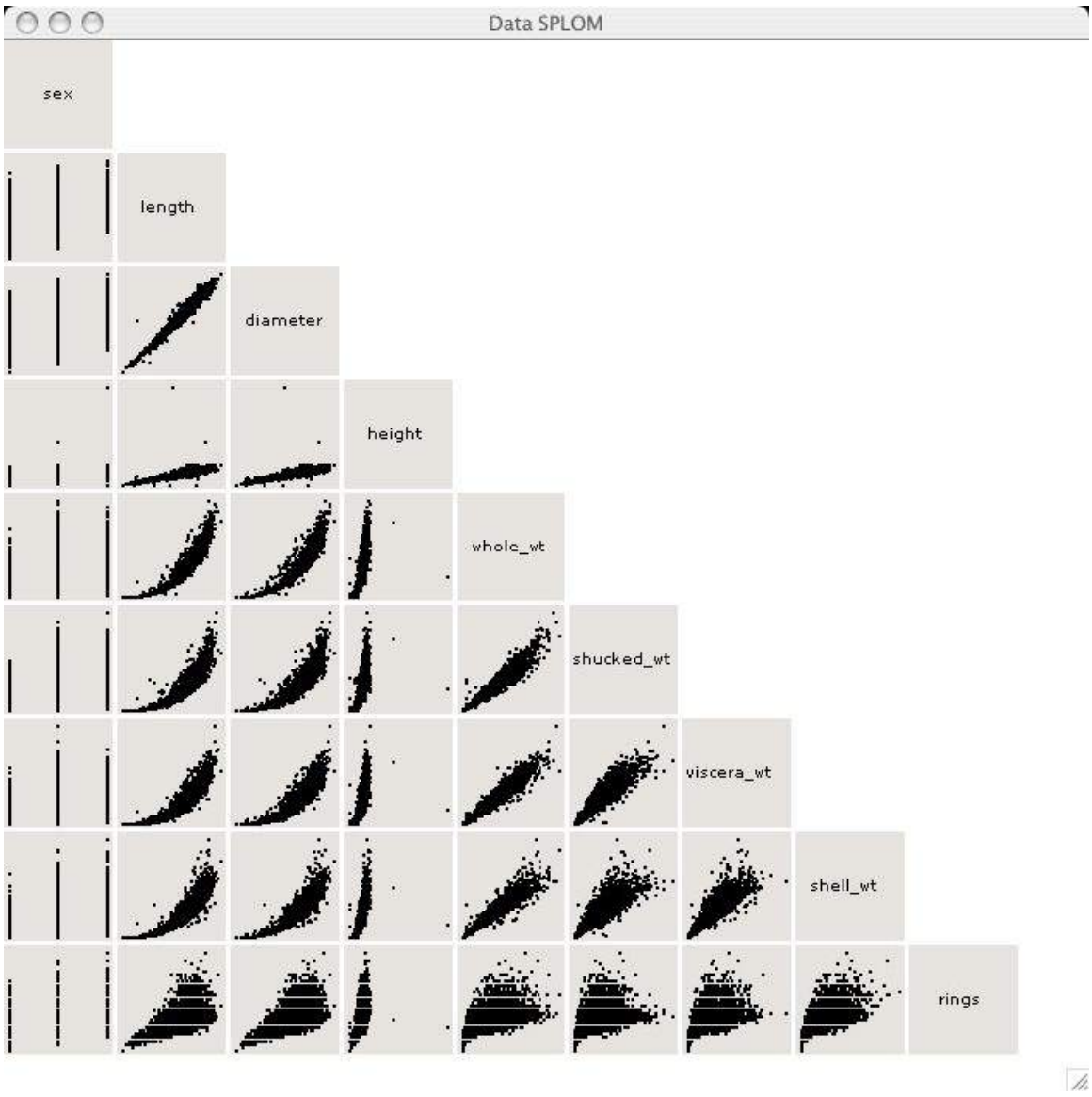


Figure 4.8: Scatterplot matrices (SPLOM) showing abalone data. From [Wilkinson et al. 05], Figure 1.

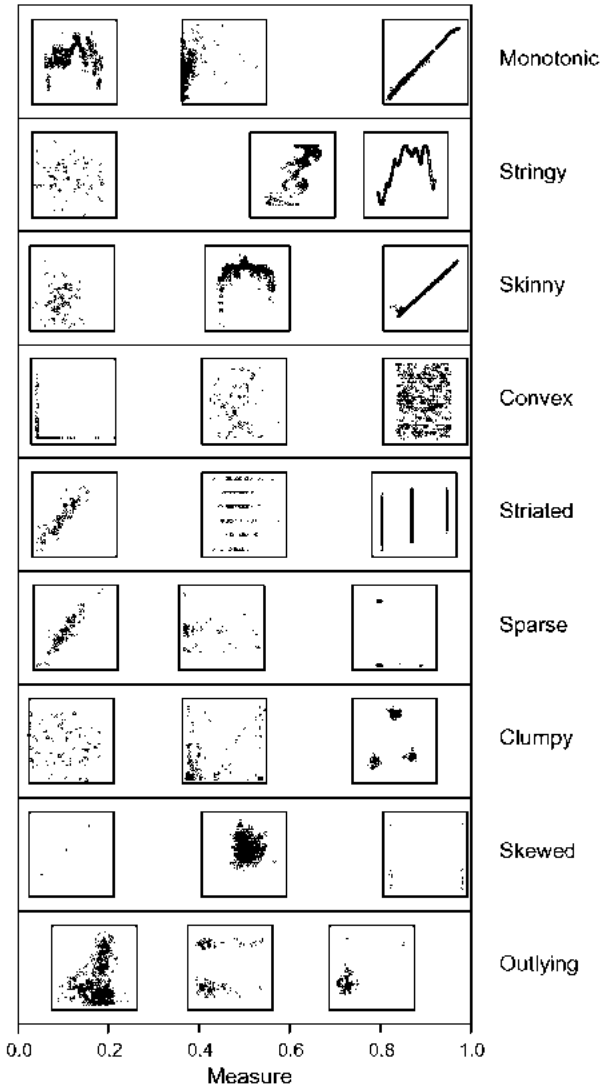


Figure 4.9: The nine scagnostics measures that describe scatterplot shape, with examples of real-world datasets rated low, medium and high for each of the nine measures. From [Wilkinson and Wills 08], Figure 6.

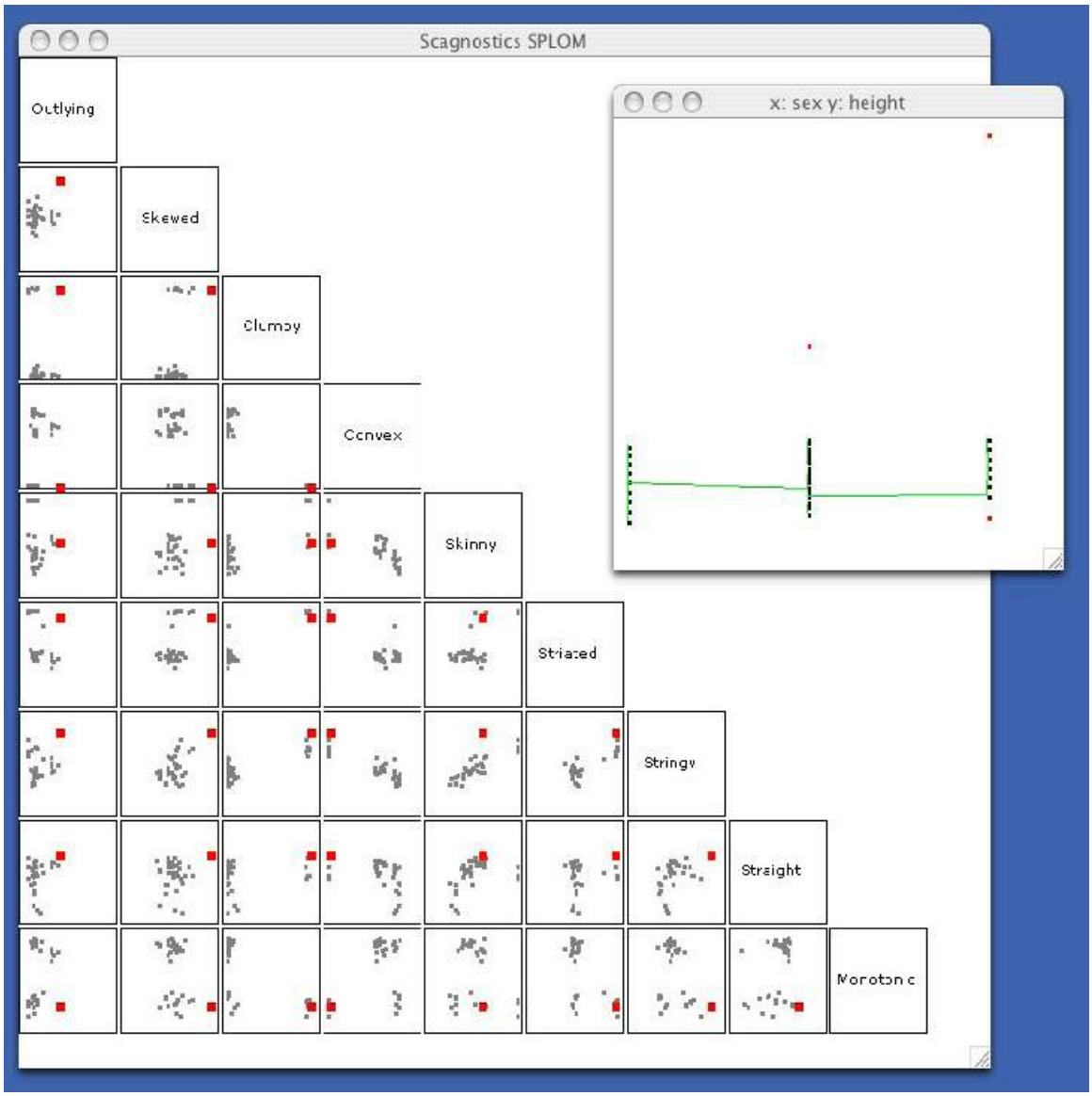


Figure 4.10: Scagnostics SPLOM for the abalone dataset, where each point represents an entire scatterplot in the original matrix. From [Wilkinson et al. 05], Figure 5.

4.7 Data Abstraction

Returning to the nested model of visualization design from Chapter 2, the abstraction stage requires choosing an appropriate abstraction for the data and tasks. This chapter has focused on the data side of the abstraction question. The taxonomy of dataset and attribute types and semantics presented in this chapter gives you, the visualization designer, a framework for thinking about data abstractions, and to establish a set of terms that will be used throughout the book. While there are many possible ways to think about data, this taxonomy matches up with the taxonomy of methods that are the building blocks for a visualization designer covered in Part IV.

The abstraction stage in visualization design reifies the idea that *the dataset* is not an immutable object that is handed to the visualization designer, who must then simply visually encode it. Rather, a key part of visualization design is the active choice of the most appropriate abstraction for both data and task. The previous section on derived attributes and spaces emphasized that choosing an appropriate abstraction often requires a transformation of the original dataset into a new form, where new attributes are derived from the original dataset attributes.

There is a strong relationship between the form of the data and what you can do with it. The good news is that your hands are not tied as a designer because you can transform the data into a form useful for the task at hand. Don't just draw what you're given; decide what the right thing to show is, and draw that!

4.8 History and Further Reading

The taxonomy presented here was inspired in part by the many taxonomies of data that have been previously proposed, including Card et al. [Card et al. 99], Tory and Möller [Tory and Möller 04a], and Shneiderman [Shneiderman 96].

Stevens originally proposed the categorical, ordered, and quantitative taxonomy in his theory of scales of measurement [Stevens 46]. Wilkinson also discusses measurement scales extensively in his Grammar of Graphics [Wilkinson 05].

The taxonomy of attribute semantics in this chapter is not exhaustive. For example, in the field of databases, it is common to call categorical attributes **dimensions** and quantitative attributes **measures**. The Polaris/Tableau visualization system is built around this distinction [Stolte et al. 08].

The cartographic literature discusses diverging versus sequential data types .

Aigner et al. discuss the visualization of time-oriented data extensively in their book [Aigner et al. 11].

Many visualization pipeline models discuss the idea of data transformation as a critical early step, including the influential one from Card et al. [Card et al. 99] and the data state reference model of Chi [Chi and Riedl 98]. The taxonomy of Tory and Moeller [Tory and Möller 04b] explicitly discusses the idea that data types can change as the result of the transformation.

Part III

Principles

This part of the book covers the principles that underlie all of the methods discussed in Part IV.

Chapter 5 covers visual encoding principles. It establishes a framework for analysis in term of marks on a page and visual channels of information including position, color, size, orientation, and shape. These channels all have different characteristics in terms of the accuracy of our perceptual system, how many levels of information they can convey, and whether they can be interpreted separately or are automatically merged with other channels. The strengths and weaknesses of each channel are deeply relevant to visualization design and analysis. They are discussed in detail here so that later chapters can briefly refer back to these ideas instead of repeating the arguments in full each time they apply.

Chapter 6 covers the principles of interaction in order to create displays that are dynamic rather than static. The chapter starts with an overview of the kinds of changes that interaction designers can use. It then covers how people respond very differently depending on how long something takes to happen, so the kind of feedback they need depends on which latency class applies. Changing the display has both benefits and costs. The chapter covers the fundamental tradeoffs of changes that require the use of internal memory versus inspecting an immediately visible display.

Chapter **??** TODO

5

Visual Encoding Principles

The previous chapter presented a taxonomy of data types; now we discuss how to encode these types with a visual representation. This chapter begins with discussion of human perception as a system for making relative rather than absolute judgements. It continues with the image theory of marks and visual channels, and a discussion of channel types. The chapter presents a ranking of channels according to the data type that they are used to encode, and discusses the concepts of expressiveness and effectivenesss. It continues with ways to measure channel effectiveness in terms of accuracy, discriminability, and separability, and ability to provide visual popout. The chapter then discusses the characteristics of each channel, including planar spatial position, color, size, tilt and angle, shape, and stipple. It ends with a discussion of the difficulties of 3D depth coding.

5.1 Relative vs. Absolute Judgements

The human perceptual system is fundamentally based on relative judgements, not absolute ones. **Weber's Law** states that the amount of difference that we can detect is relative to the context between the two things, not an absolute quantity. [1] For instance, the amount of length difference we can detect is a percentage of the object's length.

This principle holds true for all sensory modalities. The fact that our senses work through relative rather than absolute judgements has far-ranging implications. When considering questions such as the accuracy and discriminability of our perceptions, we must distinguish between relative and absolute judgements. For example, when two objects are directly next to each other and aligned, we can make much more precise judgements than when they are not aligned and when they are separated with many other objects between them.

[1] Vocabulary note: More formally, Weber's Law is typically stated as the detectable difference in stimulus intensity I is a fixed percentage K of the object magnitude: $\delta I / I = K$.

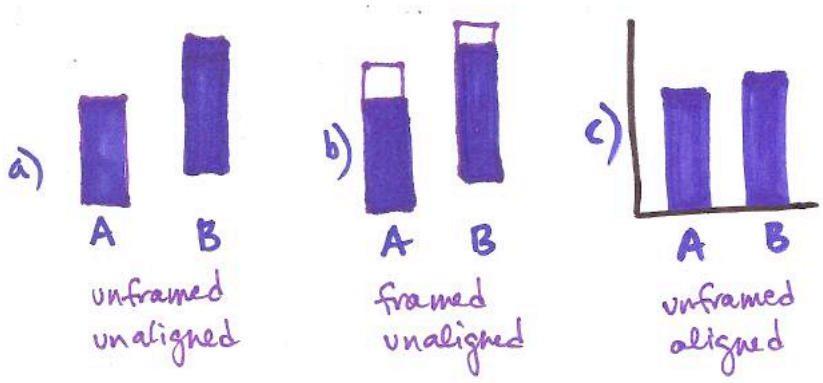


Figure 5.1: Weber's Law states that we judge based on relative, not absolute differences. (a) The lengths of unframed, unaligned rectangles of slightly different sizes are hard to compare. (b) Adding a frame allows us to compare the very different sizes of the unfilled rectangles between the bar and frame tops. (c) Aligning the bars also makes the judgement easy. Redrawn and extended from [Cleveland and McGill 84].

An example based on Weber's Law illuminates why position along a scale can be more accurately perceived than a pure length judgement of position without a scale. Figure 5.1 shows an example of a length judgement that is difficult to make with unaligned and unframed bars, but easier with framing or alignment so that they can be judged against a common scale. When making a judgement without a common scale, the only information is the length of the bars themselves. Placing a common frame around the bars provides another way to estimate magnitude: we can check the length of the unfilled bar. Bar B is only about 15% longer than Bar A, approaching the range where length differences are difficult to judge. But the unfilled part of the frame for Bar B is about 50% smaller than the one for Bar A, an easily discriminable difference. Aligning the bars achieves the same effect without the use of a frame.

Another example show that our perception of color and luminance is completely contextual, based on the contrast with surrounding colors. In Figure 5.2(a), the two labelled squares in a checkerboard appear to be quite different shades of grey. In Figure 5.2(b), superimposing a solid grey mask that touches both squares shows that they are identical. Conversely, Figure 5.3 shows two colorful cubes. In Figure 5.3a corresponding squares both appear to be red. In Figure 5.3b, masks show that the tile color in the image apparently illuminated by a yellowish light source is actually orange, and for the bluish light the tiles are actually purple. Our visual

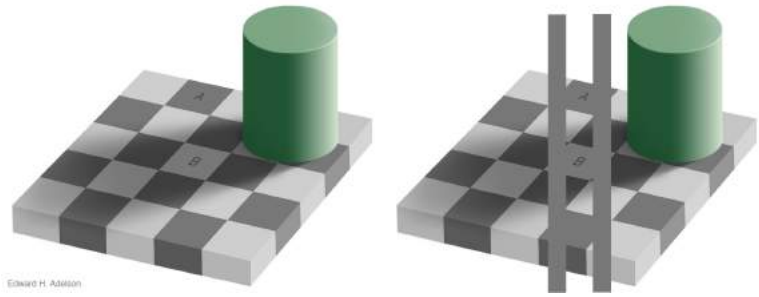
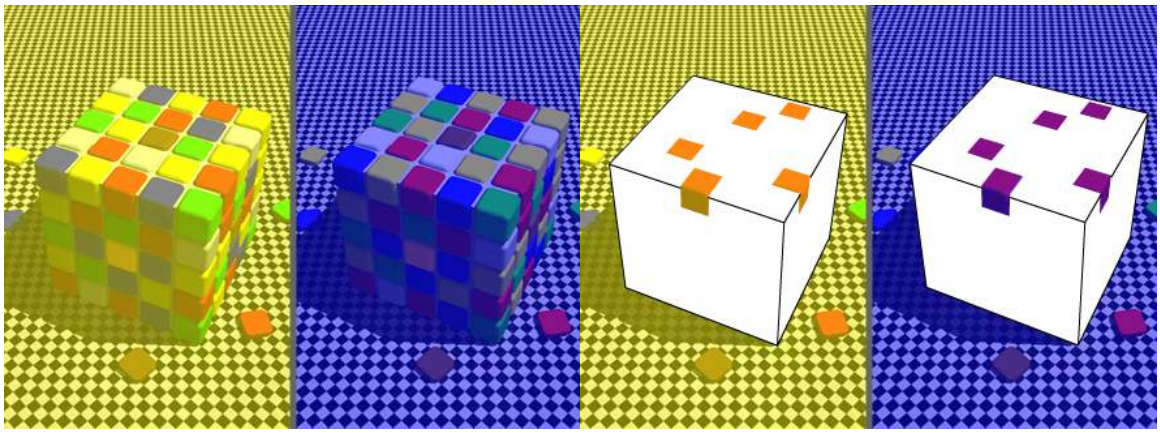


Figure 5.2: Lightness perception is based on relative, not absolute, judgements. (a) The two squares A and B appear quite different. (b) Superimposing a grey mask on the image shows that they are in fact identical. Courtesy of Edward H. Adelson.



(a) (b)

Figure 5.3: Color perception is also relative to surrounding colors and depends on context. (a) Both cubes have tiles that appear to be red. (b) Masking the intervening context shows that the colors are very different: with yellow apparent lighting, they are orange; with blue apparent lighting, they are purple.

system evolved to provide **color constancy** so that the same surface is identifiable across a broad set of illumination conditions, even though a physical light meter would yield very different readings. While the visual system works very well in natural environments, many of its mechanisms

work against simple approaches to visually encoding information with color.

5.2 Marks and Channels

We can describe a large part of the design space of visual encodings as an orthogonal combination of two aspects: graphical elements called marks, and visual channels to control their appearance.

5.2.1 Marks

A **mark** is a basic graphical element in an image. Marks are geometric primitive objects classified according to the number of spatial dimensions they require: points, lines, areas, or even volumes. That is, a zero-dimensional mark is a point, a one-dimensional mark is a line, a two-dimensional mark is an area, and a three-dimensional mark is a volume.

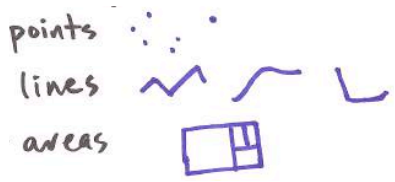


Figure 5.4: Marks are geometric primitives.

5.2.2 Channels

Visual channels are all of the parameters that control the appearance of marks, independent of the dimensionality of the geometric primitive. The set of available visual channels includes position, size, shape, orientation, hue, saturation, lightness, and many others; some are shown in Figure 5.5. A simple way to think about them is channels of information that flow from the retina of the eye to the brain. There are many visual channels that can encode information for a given mark. The rest of this chapter contains an extensive discussion of channels and their properties.

5.2.3 Using Marks and Channels

Figure 5.6 shows a progression of chart types, with each showing one more data attribute by using one more visual channel. A single attribute can be encoded with vertical spatial position. Bar charts are a common example of this encoding: the height of the bar conveys a quantitative value for

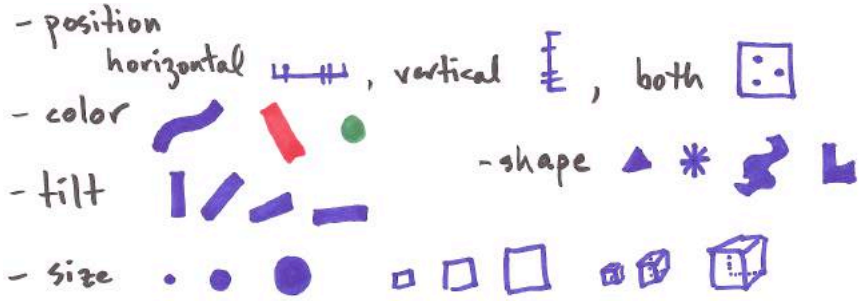


Figure 5.5: Visual channels control the appearance of marks.

Figure 5.6: Using marks and channels. (a) Bar charts encode one attribute using a line mark and the visual channel of vertical spatial position. (b) Scatterplots encode two attributes using point marks and both vertical and horizontal spatial position. (c) A third attribute can be encoded by adding color to the scatterplot. (d) The visual channel of size encodes a fourth attribute as well.

that attribute, as in Figure 5.6(a). A second, independent attribute can be encoded by adding the visual channel of horizontal spatial position. Scatterplots are exactly this visual encoding for the mark type of points, as in Figure 5.6(b). We cannot continue to add more spatial position encodings when creating drawings in two-dimensional space. Spatial position is not the only visual channel available to us for encoding that information visually. An additional data attribute can be encoded in a scatterplot format using the visual channel of hue (one aspect of color), as in Figure 5.6(c). Figure 5.6(d) shows the addition of a fourth independent data attribute encoded with the visual channel of size, which is area in this case of a large circular point.

In these examples, each attribute is encoded with a single channel. Multiple channels can be combined to redundantly encode the same attribute. The limitation of this approach is that more channels are "used up" so that not as many attributes can be encoded in total, but the benefit is that the attributes that are shown will be very easily perceived.

5.2.4 Channel Types

The human perceptual system has two fundamentally different kinds of sensory modalities. One kind of channel tells us information about identity and location: **what** something is. In contrast, another kind tells us magnitudes: **how much** of something there is.[2] For instance, we can tell *what* shape we see: a circle, or a triangle, or a cross. It does not make much sense to ask *how much* questions about magnitudes for shape. In contrast, we can ask about magnitudes with line length: how much longer is this line than that line? And *what* is not a productive question, since both objects are lines.

A few visual channels give us *what* information: what shape, what spatial region, what color. Most channels give us *how much* information, including length, area, volume, and tilt. Spatial position also conveys magnitudes, because we can consider how much further along an axis one item is compared to another.

The taxonomy of data attributes in Chapter 4 also encapsulates this very division. The *what* channels are the correct match for the *categorical* attributes that have no intrinsic order. The *how much* channels are the correct match for the *ordered* attributes, both *ordinal* and *quantitative*.

5.2.5 Mark Types

The discussion so far has been focused on table datasets, where a mark always represents an item. For network datasets, a mark might represent either an item – also known as a node – or a link. Link marks represent a relationship between items. The two link mark types are connection and containment. **Connection** marks show a pairwise relationship between two items, using a line. **Containment** marks can show hierarchical relationships using areas, and they can be nested within each other to do so at multiple levels.[3] While the visual representation of the area mark might be with a line that depicts its boundary, containment is fundamentally about the use of area. Links cannot be represented by points, even though individual items can be.

5.2.6 Expressiveness and Effectiveness

Two principles guide the use of visual channels in visual encoding: expressiveness and effectiveness.

The **expressiveness principle** dictates the visual encoding should express all of, and only, the information in the dataset attributes. For ex-

[2] Vocabulary note: In the psychophysics literature, the *what* channels are called metathetic, and the *how much* channels are called prothetic.

[3] Vocabulary note: Synonyms for containment are enclosure and nesting.

ample, ordered data should be shown in a way that our perceptual system intrinsically senses as ordered. Conversely, unordered data should not be shown in a way that perceptually implies an ordering that does not exist. Violating this principle is a common beginner's mistake in visualization.

The **effectiveness principle** dictates that the effectiveness of the channel should match the importance of the attribute. The most important attributes should be encoded with the most effective channels, and then decreasingly important attributes can be matched with less effective channels. The rest of this chapter is devoted to a detailed discussion of channel effectiveness and expressiveness.

5.2.7 Channel Rankings

Figure 5.7 presents effectiveness rankings for the visual channels broken down according to the two expressiveness types of ordered and categorical data. The rankings range from the most effective channels at the top to the least effective at the bottom. Ordered data should be shown with the *how much* channels. The most effective is aligned spatial position, followed by unaligned spatial position. Next is length, which is one-dimensional size, and then orientation considered as tilt or angle, and then area which is two-dimensional size. The next two channels, curvature and volume (3D size), are equally effective. The last three channels are also equally effective: color luminance (greyscale value), color saturation, and stipple density. Categorical data should be shown with the *what* channels. The most effective channel for categorical data is spatial region, with color hue as the next best one. The following shape and stipple pattern channels have similar effectiveness. While it is possible in theory to use a *how much* channel for categorical data or a *what* channel for ordered data, that choice would be a poor one because the expressiveness principle would be violated.

Figure 5.7 also summarizes the mark types according to their geometric dimension. When marks are used for items, they can be points, lines, or areas. When they are used for relationships between items, the only two options are lines for connection and areas for containment.

These rankings are my own synthesis of information drawn from many sources, including several previous frameworks, experimental evidence from a large body of empirical studies, and my own analysis. The further reading section at the end of this chapter contains pointers to the previous work. The following sections of this chapter discuss the reasons for these rankings at length, and provide definitions for all of these channels.

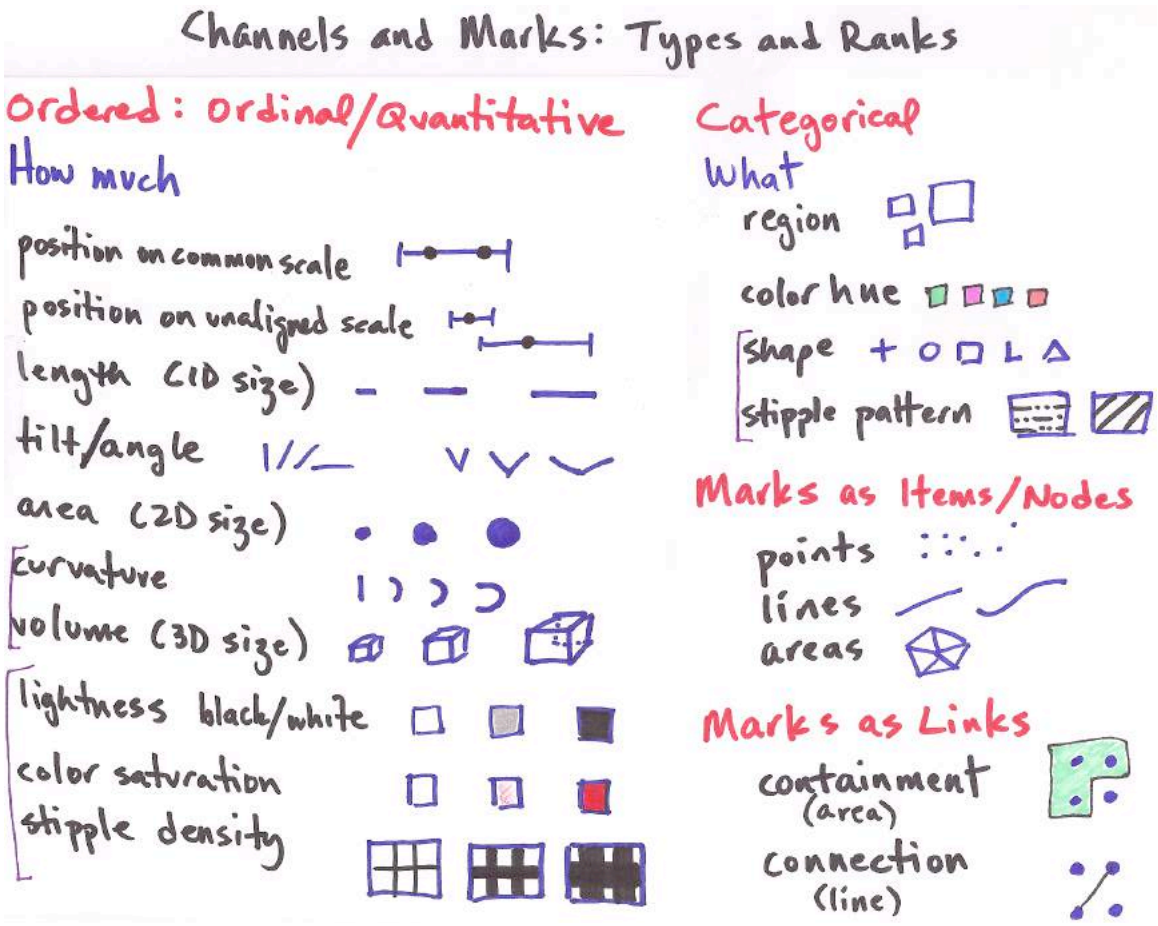


Figure 5.7: Channels ranked by effectiveness according to data and channel type. Ordered data should be shown with the *how much* channels, and categorical data with the *what* channels. Marks can represent individual items, or links between them.

5.3 Channel Effectiveness

To analyze the space of visual encoding possibilities we need to understand the characteristics of these visual channels, because many questions remain unanswered: How are these rankings justified? How did we decide to use those particular visual channels? How many more visual channels are there? What kinds of information and how much information can each

channel encode? Why are some channels better than others? Can all of the channels be used independently or do they interfere with each other?

This section addresses these questions by introducing the analysis of channels according to the criteria of accuracy, discriminability, separability, and the ability to provide visual popout.

5.3.1 Accuracy

The obvious way to quantify effectiveness is **accuracy**: how close is human perceptual judgement to some objective measurement of the stimulus? Some answers come to us from **psychophysics**, the subfield of psychology devoted to the systematic measurement of general human perception. We perceive different visual channels with different levels of accuracy; they are not all equally distinguishable. Our responses to the sensory experience of magnitude are characterizable by power laws, where the exponent depends on the exact sensory modality: most stimuli are magnified or compressed, with few remaining unchanged.

Figure 5.8 shows the psychophysical power law of Stevens [Stevens 75]. The apparent magnitude of all sensory channels follow a power function based on the stimulus intensity:

$$S = I^n, \tag{5.1}$$

where S is the perceived sensation and I is the physical intensity. The power law exponent n ranges from the sublinear .5 for brightness to the superlinear 3.5 for electric current. That is, the sublinear phenomena are compressed, so doubling the physical brightness results in a perception that is considerably less than twice as bright. The superlinear phenomena are magnified: doubling the amount of electric current applied to the fingertips results is a sensation that is much more than twice as great. We can see in Figure 5.8 that length has an exponent of $n = 1.0$, so our perception of length is a very close match to the true value. Here *length* means the length of a line segment on a 2D plane perpendicular to the observer. The other visual channels are not perceived as accurately: area and brightness are compressed, and while greyscale lightness and red-grey saturation are magnified.

Another set of answers to the question of accuracy comes from controlled experiments that directly map human response to visually encoded abstract information, giving us explicit rankings of perceptual accuracy for each channel type. For example, Cleveland and McGill's experiments on the magnitude channels [Cleveland and McGill 84] showed that position along a common scale is most accurately perceived, followed by position along nonaligned scales, followed by length and direction and angle judgements without an explicit scale. Area judgements are notably less accurate

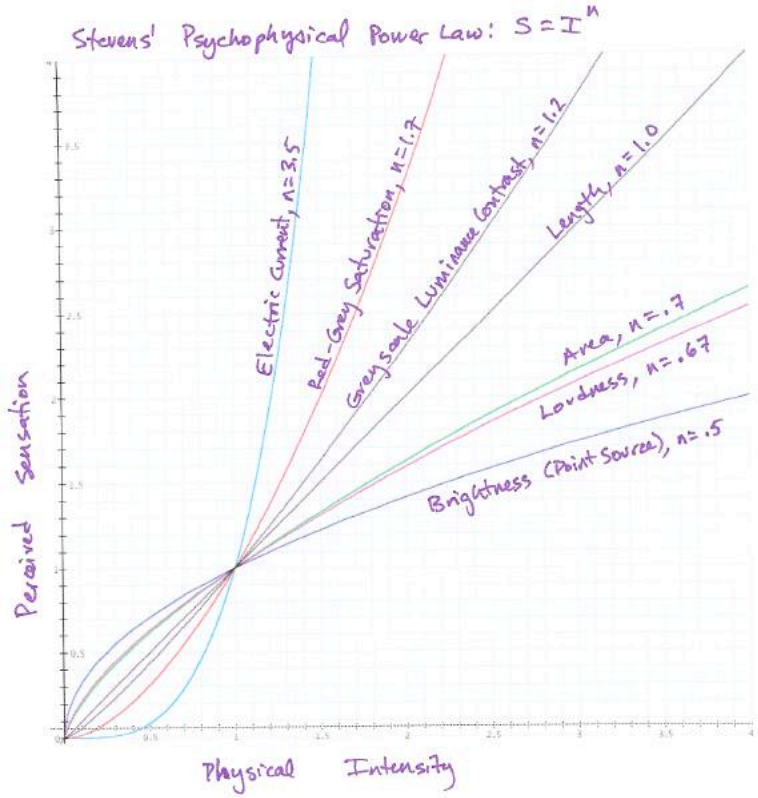


Figure 5.8: Stevens showed that the apparent magnitude of all sensory channels follow a power law $S = I^n$, where some sensations are perceptually magnified compared to their objective intensity (when $n > 1$) and some compressed (when $n < 1$). We see that length perception is completely accurate, whereas area is compressed and saturation is magnified. Data from Stevens [Stevens 75] (p. 15).

than all of these, with both volume and curvature judgements worse yet. Greyscale shading and color saturation were the least accurate judgements. These accuracy results for visual encodings dovetail nicely with the psychophysical channel measurements.

The rankings in Figure 5.7 are based on accuracy, which differs according to the type of the attribute that is beign encoded. While the next three criteria do not directly contribute to the effectiveness rankings, they are nevertheless important considerations for design and analysis.

5.3.2 Discriminability

The framed rectangles example in Section 5.1 is a good introduction to the larger question of **discriminability**: can items that we encode using different intensities in fact be distinguished as being different, for a given visual channel. Thus, we need to quantify the number of **bins** that are available for use within a visual channel, where each bin is a distinguishable step or level from the other.

For instance, some channels have a very limited number of bins. Consider line width: changing the line size only works for a fairly small number of steps. Increasing the width past that limit will result in a mark that is perceived as a polygon area rather than a line mark. A small number of bins is not a problem if the number of values to encode is also small. For example, Figure 5.9 shows an example of effective linewidth use. Line width can work very well to show three or four different values for a data attribute, but would be a poor choice for dozens or hundreds of values. The key factor is matching the ranges: the number of different values that need to be shown for the attribute being encoded must not be greater than the number of bins available for the visual channel used to encode it. If these do not match, then the visualization designer should either explicitly aggregate the attribute into meaningful bins, or use a different visual channel.

5.3.3 Separability

We cannot treat all visual channels as completely independent from each other, because some have dependencies and interactions with others. We must consider a continuum of potential interactions between channels for each pair, ranging from the orthogonal and independent **separable** channels to the inextricably combined **integral** channels. Visual encoding is straightforward with separable channels, but attempts to encode different information in integral channels will fail. People will not be able to access the desired information about each attribute; instead, an unanticipated combination percept will result.

Clearly, we cannot separately encode two attributes of information using vertical and horizontal spatial position, and then expect to encode a third attribute using planar proximity. In this case it is obvious that the third channel precludes the use of the first two. However, some of the inter-channel interference is less obvious.

Figure 5.10 shows pairs of visual channels at four points along this continuum. On the left is a pair of channels that are completely separable, position and hue. We can easily see that the points fall into two categories for spatial position, left and right. We can also separately attend to their

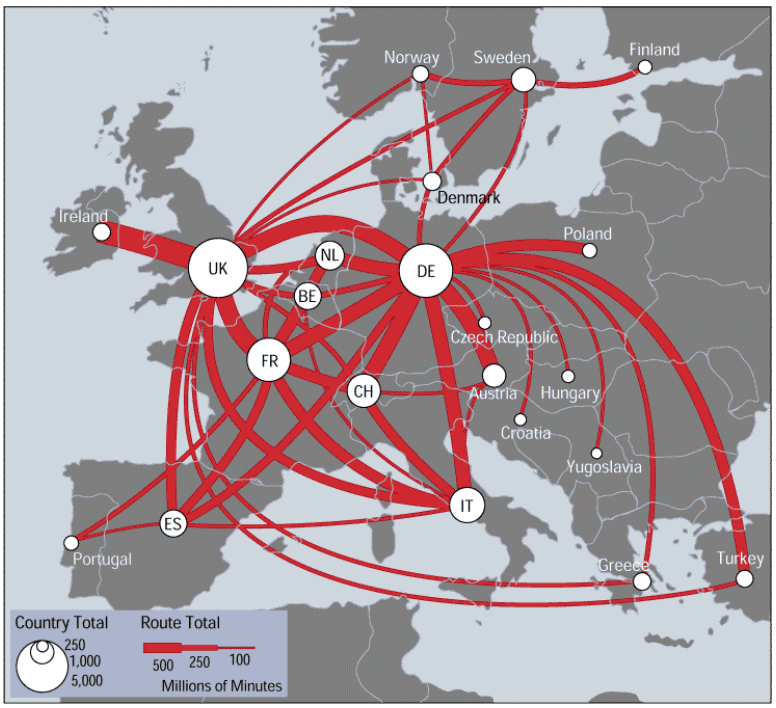


Figure 5.9: The linewidth has a limited number of discriminable bins.

hue, and distinguish the red from the black. It is easy to see that half the points fall into each of these categories for each of the two axes.

Next is an example of interference between channels, showing that size is not fully separable from color hue. We can easily distinguish the large half from the small half, but within the small half discriminating between the two colors is much more difficult. Size interacts with many visual channels, including shape.

The third example shows an integral pair. Encoding one variable with horizontal size and another with vertical size is ineffective because what we directly perceive is the planar size of the circles, namely their area. We cannot easily distinguish groupings of wide from narrow, and short from tall. Rather, the most obvious perceptual grouping is into three sets: small, medium, and large. The medium category includes the horizontally flattened as well as the vertically flattened.

The far right on Figure 5.10 shows the most inseparable channel pair, where the red and green channels of the RGB color space are used. The integrated percept is simply hue. While we can tell that there are four

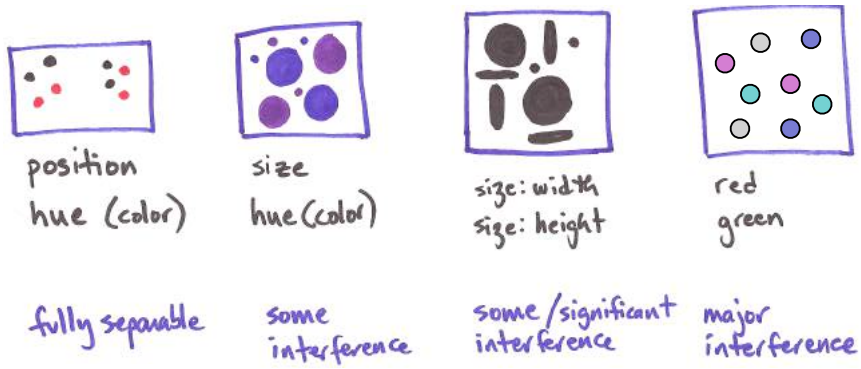


Figure 5.10: Pairs of visual channels fall along a continuum from fully separable to intrinsically integral. Color and location are separable channels well suited to encode different data attributes, for two different groupings that can be selectively attended to. However, size interacts with channel of color hue, which is harder to perceive for small objects. The horizontal size and and vertical size channels are automatically fused into an integrated perception of area, yielding three groups. Attempts to code separate information along the red and green axes of the RGB color space fail, because we simply perceive four different hues.

colors, even with intensive cognitive effort it is very difficult to try to recover the original information about high and low values for each axis. The RGB color system used to specify information to computers is a very different model than the color processing systems of our perceptual system, so the three channels are not perceptually separable. We will further discuss color in Section 5.4.2.

Integrality versus separability is not good or bad; the important idea is to match the characteristics of the channels to the information that is encoded. If the goal is to show the user two different data attributes, either of which can be attended to selectively, then separable channel pair of position and color hue is a good choice. If the goal is to show a single data attribute with three categories, then the integral channel pair of horizontal and vertical size is a reasonable choice.

5.3.4 Popout

Many visual channels provide visual **popout**, where a distinct item stands out from many others immediately.[4] Examples of popout are spotting a

[4]Vocabulary note: Visual popout is often called *preattentive* or *tunable detection*.

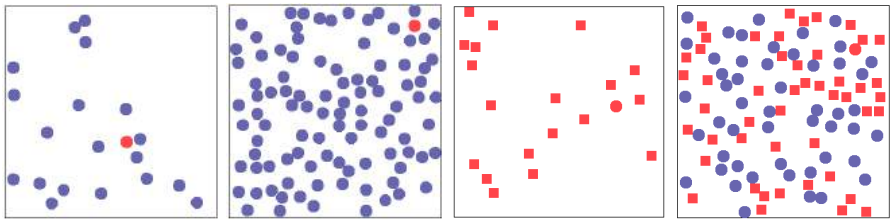


Figure 5.11: Visual popout. (a) The red dot pops out from the sea of blue dots. (b) The red dot can be spotted equally quickly against many blue dots. (c) The red dot also pops out from a sea of squares. (d) The red dot does not pop out from a sea of red and blue dots and squares, and can only be found by searching one by one through all the objects. Created with Christopher Healey's `http://www.csc.ncsu.edu/faculty/healey/PP/` software.

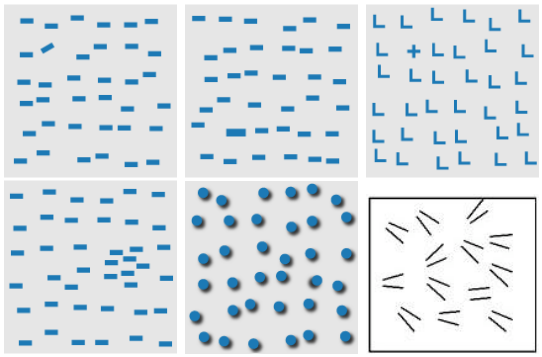


Figure 5.12: Many channels support visual popout, including tilt, size, shape, proximity, and shadow direction. However, parallel line pairs do not pop out from a sea of slightly tilted distractor object pair, and can only be detected through serial search.

red object from a sea of blue ones, or spotting one circle from a sea of squares. The great value of popout is that the time it takes us to spot the different object does not depend on the number of distractor objects. Our low-level visual system does massively parallel processing on these visual channels, without the need for the viewer consciously directly attention to items one by one. Figure 5.11 shows an example with one red dot in a sea of 20 blue dots, and then in a sea of 99 blue dots. In both cases, popout occurs with roughly the same speed. The red dot also pops out from a sea of red squares.

Popout occurs for many channels, not just color hue and shape. Figure 5.12 shows several examples: tilt, size, shape, proximity, and even lighting direction. Many other channels support popout, including several different kinds of motion such as flicker, motion direction, and motion velocity. However, a small number of potential channels do not support popout. The last subfigure in Figure 5.12 shows that parallelism is not preattentively detected; the exactly parallel pair of lines does not pop out from the slightly angled pairs. It can only be detected with **serial search**: checking each item, one by one. Thus, the amount of time it takes to find the target depends linearly on the number of distractor objects.

Although many different visual channels provide popout on their own, they cannot simply be combined. Figure 5.11 d shows that a red dot does not pop out automatically from a sea of objects that can be red or blue, and dots or squares. A very few channel pairs do support popout in some cases, but most do not. Popout is definitely not possible with three or more channels. As a general rule, visualization designers should only count on using popout for a single channel at a time.

Moreover, popout is not an all or nothing phenomenon. It depends on both the channel itself, and how different the target item is from its surroundings. The red dot pops out from the sea of blue dots in Figure 5.11(a) faster than from the sea of red squares in Figure 5.11(b). The difference between red and blue on the color hue channel is larger than the difference in shape between filled-in circles and filled-in squares.

In summary, all of the major channels commonly used in visual encoding that are shown in Figure5.7 do support popout individually, but not in combination with each other.

5.4 Channel Characteristics

The critical point of the previous section is although the same data attribute can be visually encoded with many different visual channels, the resulting information content in our heads after passing through the perceptual and cognitive processing pathways of our visual systems and brains is most definitely not equivalent. We now discuss the specific characteristics of the individual visual channels in terms of accuracy, discriminability, and separability. Discriminability is analyzed in terms of the number of bins. For simplicity, the display is analyzed as a square screen with 1000 pixels on each side.

5.4.1 Planar Position

The two ranked lists of channels in Figure 5.7 both have channels related to planar spatial position at the top in the the most effective spot. Aligned and unaligned spatial position are at the top of the list for ordered data, and spatial region is at the top of the list for categorical data. Moreover, the spatial channels are the only ones that appear on both lists; none of the others are effective for both data types.

These fundamental observations have motivated many of the visualization techniques illustrated this book, and underlie the framework of methods presented in Part IV. The choice of which attributes to encode with position is the most central choice in visual encoding. The attributes encoded with position will dominate the user's mental model compared to those encoded with any other visual channel. (A **mental model** is an internal mental representation used for thinking and reasoning.) In slogan form, remember *the power of the plane*.

Historically, visualization diverged into subfields based on this very question. The field of **information visualization**, or infovis, was concerned with problems where the spatial position in a visual encoding was *chosen* by the designer. The subfield of **scientific visualization**, or scivis, was concerned with problems where the spatial position was a *given* property because the data abstraction included inherently spatial data, and the task abstraction required understanding its geometric structure.

Vertical and horizontal position are combined into the shared category of *planar* because the differences between the up-down and side-to-side axes are relatively subtle. We do perceive height differences along the up-down axis as more important than horizontal position differences, no doubt due to the physical effects of gravity in real life. While the vertical spatial channel thus has a slight priority over the horizontal one, the aspect ratio of standard displays gives more horizontal pixels than vertical ones, so information density considerations sometimes override this concern. For the perceived importance of items ordered within the axes, reading conventions probably dominate. Most Western languages go from left to right and from top to bottom, but Arabic and Hebrew are read from right to left, and some Asian languages are read vertically.

5.4.2 Color

Color is a rich and complex topic, and here I only touch on the most crucial aspects that apply to visualization.

5.4.2.1 Color Vision Processes

The retina of the eye has two different kinds of receptors. The **rods** actively contribute to vision only in low-light settings and provide low-resolution black and white information. We will thus

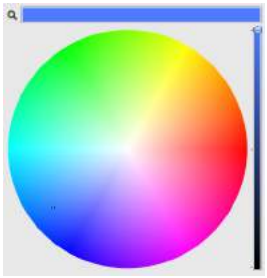


Figure 5.13: A common HSL/HSV colorpicker design, as in this example from Mac OSX, is to show a color wheel with fully saturated color around the outside and white at the center of the circle, and a separate control for the darkness.

not discuss them further in this book. The main sensors in normal lighting conditions are the **cones**. There are three types of cones, each with peak sensitivities at a different wavelength within the spectrum of visible light. The visual system immediately processes these signals into three **opponent color channels**: one from red to green, one from blue to yellow, and one from black and white encoding luminance information. The luminance channel conveys high-resolution edge information, while the red-green and blue-yellow channels are lower resolution. This split between luminance and the rest of what we would normally call *color* is a central issue in visual encoding design.

The theory of opponent color channels explains what we colloquially call **color blindness**, which affects around 8% of men in its most common form. A more accurate term is **color deficiency**, since the ability to differentiate color along the red-green channel is reduced or absent but the blue-yellow channel is still in full working order.[5]

5.4.2.2 Color Spaces The **color space** of what colors the human visual system can detect is three dimensional; that is, it can be adequately described using three separate axes. There are many ways to mathematically describe color as a space, and to transform colors from one such space into another. Some of these are extremely convenient for computer manipulation, while others are a better match with the characteristics of human vision.

The most common color space in computer graphics is the three-dimensional **RGB** system, where colors are specified as triples of red, green, and blue

[5]There is a type of color deficiency where the blue-yellow channel is impaired, but it is extremely rare.

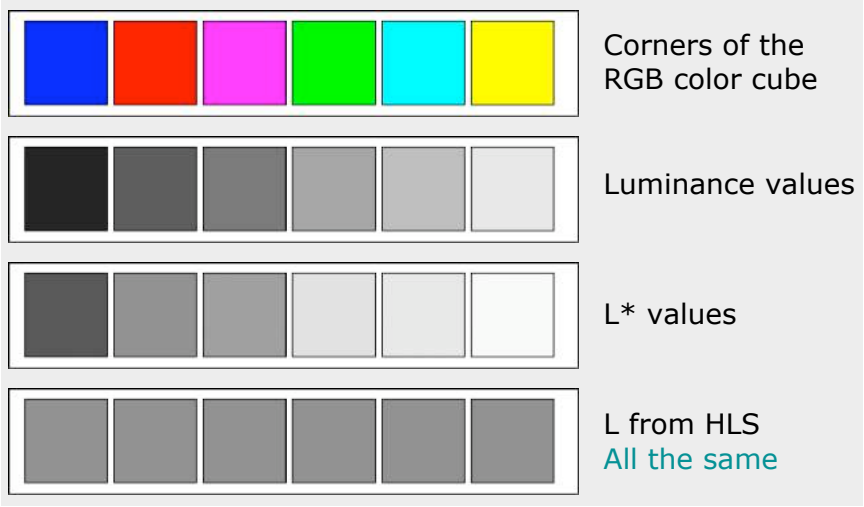


Figure 5.14: The HSL color space is only pseudo-perceptual: the lightness channel L is not the same as true luminance L^*. From [Stone 06].

values. Although this system is computationally convenient, it is a very poor match for the mechanics of how we see. The red, green, and blue axes of the RGB color space are not useful as separable channels; they give rise to the integral percept of a color. We do not discuss them further as channels in our analysis of visual encoding.

Another color space, **HSL**, is more intuitive and is heavily used by artists and designers. The **hue** axis captures what we normally think of as pure colors that are not mixed with white or black: red, blue, green, yellow, purple, and so on. The **saturation** axis is the amount of white mixed with that pure color. For instance, pink is a partially desaturated red. The **lightness** axis is the amount of black mixed with a color. A common design for color pickers is a disc with white at the center and the hue axis wrapped around the outside, with separate linear control for the amount of darkness versus lightness, as shown in Figure 5.13. The HSV space is very similar, where V stands for greyscale value and is linearly related to L.

Despite the popularity of the HSL space, it is is only pseudo-perceptual. There are several spaces that attempt to provide a more perceptually uniform space, including one known as $L^*a^*b^*$. This space has a single black and white luminance channel L^*, and the two color axes a^* and b^*. A critical difference between HSL and $L^*a^*b^*$is that the L greyscale lightness axis is significantly different from the L^* luminance axis, as shown in

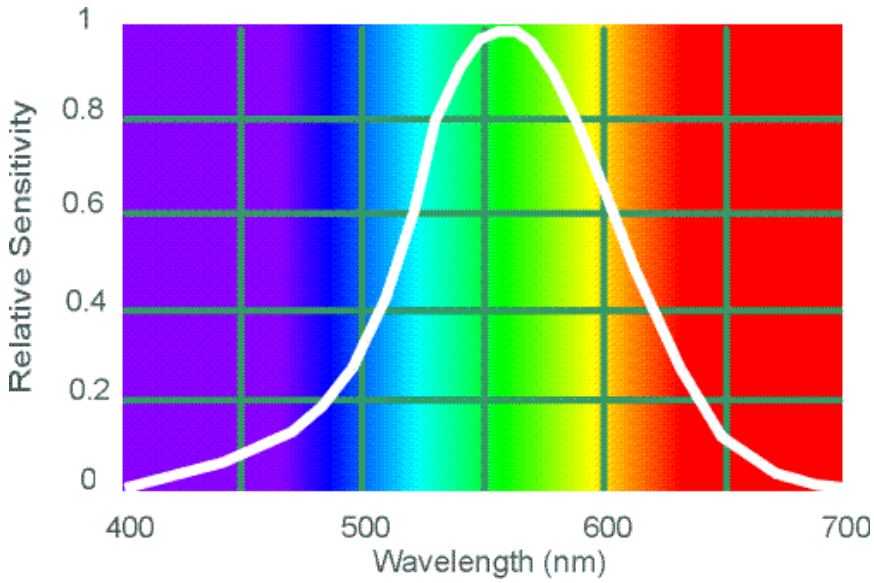


Figure 5.15: The spectral sensitivity of our eyes to luminance depends on the wavelength of the incoming light. From http://www.yorku.ca/eye/photopik.htm

Figure 5.14. Luminance is the channel that our eyes detect, not lightness. Figure 5.15 shows the roughly bell-shaped **spectral sensitivity** curve for daylight vision; clearly, the amount of luminance that we perceive depends on the wavelength.

The L^* axis is a nonlinear transformation of the luminance perceived by the human eye. As discussed in Section 5.3.1, doubling brightness does not result in a perception that is twice as bright. The L^* axis is designed to be perceptually linear, so that equally sized steps appear equal to our visual systems, based on extensive measurement and calibration over a very large set of observers. The color axes have also been designed to be as perceptually linear as possible. The $L^*a^*b^*$space is thus well-suited for many computations, including interpolation and finding differences between colors, that should not be executed in the perceptually nonlinear RGB or HSL spaces.

I distinguish luminance, hue and saturation as the three separable visual channels that pertain to color in the analysis of visual encoding. Luminance and saturation are *how much* channels, while hue is a *what* channel. When I use the generic term **color**, I mean the integral percept across all three of these channels at once. Color can thus be either a *how much* channel or

a *what* channel depending on how it is used. I consider each of the three channels in turn to analyze their capabilities and limitations independently.

5.4.2.3 Luminance Luminance is a *how much* channel suitable for ordered data types. However, one problem with using it for encoding a specific data attribute is that luminance is then "used up" and cannot be used for other purposes.

Luminance contrast is the only way we can resolve fine detail and see crisp edges; hue contrast or saturation contrast does not provide detectable edges. In particular, text is not readable without luminance contrast.

A good rule of thumb for design is the slogan "Get it right in black and white" [Stone 05, Stone 10]; that is, ensure that there is good luminance contrast for all of the fine-grained features.

Another consideration with luminance is our low accuracy for noncontiguous regions because of contrast effects. Thus, the number of discriminable steps is small, typically less then 6.

5.4.2.4 Saturation Saturation is also a *how much* channel suitable for ordinal data. Saturation shares the problem of low accuracy for noncontiguous regions. Moreover, saturation interacts strongly with the size channel: it is more difficult to perceive in small regions than in large ones. Both saturation and hue are difficult to detect in small regions.

When colored regions are large, as in backgrounds, a standard design guideline is to use low-saturation colors in the pastel range. For small regions, designers should use bright, highly saturated colors to ensure that the color coding is distinguishable. Saturation and hue are not separable channels for small regions for the purpose of categorical color coding.

Point and line marks typically occupy small regions. In this case, the number of discriminable steps for saturation is low: around three. For larger area marks, a few more steps may be distinguishable.

5.4.2.5 Hue Hue is a *what* channel, and is extremely effective for categorical data and showing groupings. It is the highest ranked channel after spatial position.

However, hue shares the same challenges as saturation in terms of interaction with the size channel: hue is harder to distinguish in small regions than large regions. It also shared the same challenges as saturation and luminance for separated regions: we can make fine distinctions in hue for contiguous regions, we have very limited discriminability between separated regions. A safe number of bins for categorical color is typically between 6 and 12, or fewer if the regions have small size. Moreover, this count includes the background and any default or neutral object colors.

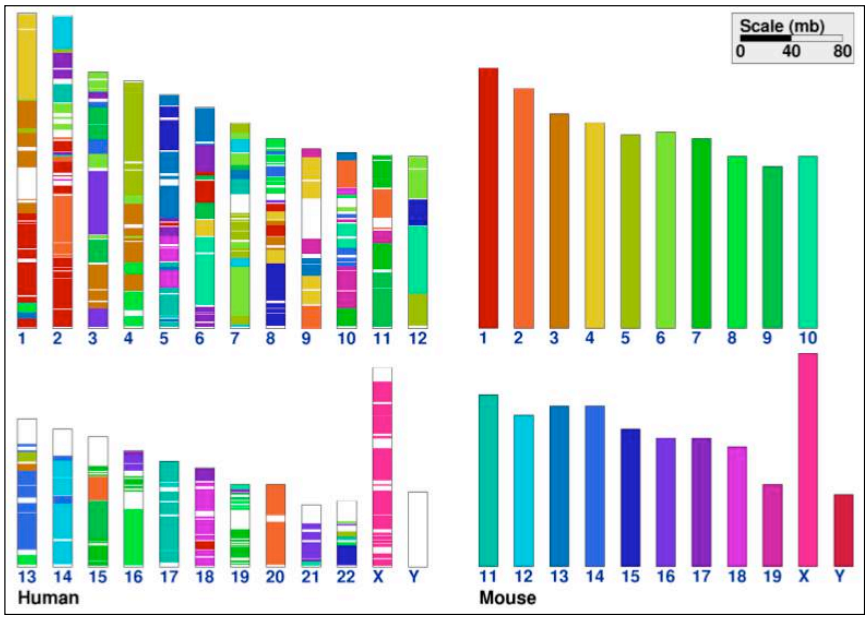


Figure 5.16: The number of discriminable bins for categorical color is limited to around 12 for noncontiguous small regions, as on the left, even though fine distinctions can be made for regions next to each other, as for the 21 chromosomes on the right. From [Sinha and Meller 07] Figure 2.

Figure 5.16 illustrates the problem of a discriminability mismatch, where categorical color is used as a visual encoding for 21 chromosomes. On the right, in the legend-style view where the subtly different marks are right next to each other, all of these colors are indeed distinguishable. On the left, where the marks are small and not contiguous, there are no more than 12 distinguishable bins. The seven greens collapse into around three bins, and so on.

Unlike luminance and saturation, hue does not have an implicit perceptual ordering, as shown in Figure 5.17. People can reliably order by luminance, always placing grey in between black and white. With saturation, people reliably place the less saturated pink between fully saturated red and zero-saturation white. However, when asked to create an ordering of red, blue, green, and yellow, people do not all give the same answer. People can and do learn conventions, such as green-yellow-red for traffic lights, or the order of colors in the rainbow, but these constructions are at a higher level than pure perception.

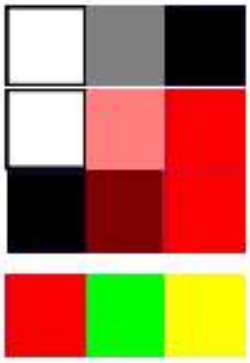


Figure 5.17: The luminance and saturation channels are automatically interpreted as ordered by our perceptual system, but the hue channel is not.

5.4.3 Size

Size is a *how much* channel suitable for ordered data. It interacts with most other channels: when marks are too small, encodings in another such as shape or tilt simply cannot be seen. Size interacts even more strongly with color hue and color saturation.

Length is one-dimensional size; more specifically, height is vertical size and width is horizontal size. Area is two-dimensional size, and volume is three-dimensional size. A larger-dimensional size coding clearly subsumes a smaller dimensional one: length and area cannot be simultaneously used to encode different dimensions. The combination of smaller-dimensional sizes is also integral rather than separable, as illustrated in Figure 5.10.

While our judgements of length are very accurate, our judgement of area is much less accurate. Stevens assigns a power law exponent of .7 for area, as shown in Figure 5.8. Volume is even more inaccurate.

5.4.4 Tilt/Angle

There are two ways consider orientation that are essentially the same channel. With tilt, an orientation is judged against the global frame of the display. With angle, the orientation of one line is judged with respect to another. While angle is somewhat less accurate than length and position, it is a significant step better than area, the next channel down on the effectiveness ranking.

We have very accurate perceptions of angles near the exact horizontal, vertical, or diagonal positions: we can tell 89 from 90 degrees, 44 from 45 degrees, and 1 from 0 degrees. However, we cannot tell 37 from 38 degrees.

The number of bins for angle is roughly dozens.

5.4.5 Shape

Shape is a catch-all word for a complex perceptual phenomenon. Vision scientists have identified many lower-level features that we can preattentively identify, including closure, curvature, termination, intersection, and others. For the purposes of analyzing visual encoding with marks and channels, I simplify by consider shape as a *what* channel that can be used with point marks. It cannot possibly apply to area marks because their shape is constrained by definition. It also does not apply to line marks; the relevant channel in that case is stipple pattern, as discussed below.

If the point size is sufficiently large, the number of discriminable bins for the shape channel is dozens or even hundreds. However, there is a strong interaction between shape and size. When the region in which the shape must be drawn is small, then far fewer discriminable bins exist. For example, given a limit of 10x10 pixels, around a dozen bins can be distinguished.

Shape can interfere other channels in the same way that size coding does, and can interfere with size coding itself. For example, filled-in shapes like disks are a good substrate for also encoding with color hue. Sparse line-oriented marks like crosses have less pixels available for color coding, so that channel will be impaired.

5.4.6 Stipple

Stippling means to fill in regions of drawing with short strokes. There are two separate channels that pertain to stipple: the pattern used, and the density of the strokes. A familiar example of a stipple pattern is the use of dotted or dashed lines. The pattern is the exact combination of dots, dashes, and other very small shapes. The density is the percentage of the region that is covered in ink rather than left blank.

Stippling is still in regular use for lines. Stippling was heavily used for area marks with older printing technology because it allows shades of grey to be approximated using only black ink; now that color and greyscale printing is cheap and pervasive, area stippling is less popular than in the past. [6] One benefit of stippling is that it is separable from the color hue and color saturation channels. However, stipple density channel interacts heavily with the luminance channel; they are not separable. The number of distinguishable bins for stipple depends heavily on the size of the region, with far fewer possibilities for lines than for areas.

[6] Vocabulary note: Hatching and cross-hatching are synonyms for stippling in two-dimensional areas.

5.4.7 Curvature

The curvature channel can only be used with line marks. It cannot be used with point marks that have no length, or area marks because their shape is fully constrained. The accuracy of and number of distinguishable bins for this channel is low, roughly equal to that of volume (3D size).

5.4.8 Motion

Several kinds of motion are also visual channels, including direction of motion, velocity of motion, and flicker frequency. The strength and weakness of motion is that it strongly draws attention; it is nearly impossible to ignore. The idea of using separable channels for visual encoding is that the viewer can selectively attend to any of the channels. While all of the motion channels are strongly separable from the non-motion channels, directing attention selectively to the non-motion channels may be difficult. Flicker and blinking are so difficult to ignore that they can be very annoying, and should usually be avoided.

5.5 History and Further Reading

The content in this chapter is my synthesis across the ideas of many previous authors. In particular, the rankings of channel effectiveness was influenced by previous work but does not come directly from any specific source.

The theory of visual marks and channels was proposed by French semiotician Bertin [Bertin 67]. The foundational work on the variable distinguishability of different visual channels, the categorization of channels as metathetic what/where and prothetic how much, and scales of measurement was done by pioneering psychophysicist Stevens [Stevens 57,Stevens 75]. Cleveland continued with the work of ranking of visual channels through measured-response experiments [Cleveland and McGill 84] and models [Cleveland 93a]. Mackinlay built on this work to propose design guidelines with rankings for visual channels according to data type, with the principles of expressiveness for matching channel to data type and effectiveness for choosing the channels by importance ordering [Mackinlay 86].

The rich literature on perception and color is absolutely worth a closer look, because this chapter only scratches the surface. Ware offers a broad, thorough, and highly recommended introduction to perception for information visualization in his two books [Ware 04, Ware 08]. His discussion includes more details from nearly all of the topics in this chapter, including color, planar vs. depth perception and stereoscopic vision, and the Gestalt

principles. Healey's detailed overview of preattentive processing is another good starting point [Healey 07]. Treisman authored some of the core work on untangling what low-level features are detected in early visual processing [Treisman and Gormican 88]. Stone's brief article and longer book are an excellent introduction to color [Stone 03].

6

Interaction Principles

Several principles of interaction are important when designing a visualization: classes of change, latency and feedback, the costs of interactivity, and spatial cognition.

6.1 Classes of Change

The fundamental point of interactivity is that the display is dynamic rather than static; that is, things change. We can categorize the kind of change that occurs in a display into four major types: a change of selection that triggers a different highlighting of dataset elements, a change of viewport that arises from navigating, a change of spatial ordering of the elements from sorting, and a change of the entire visual encoding.

6.1.1 Changing Selection

Selecting items is a fundamental piece of the interaction vocabulary. Many other higher-level interactions are handled as operations that affect the currently selected set, so we consider adding and removing items from that set as the outcome of selection actions. The important semantics revolve around the cardinality of the selection set membership; that is, how many items can be in the selection set: exactly one? many? is zero acceptable? is there a primary vs secondary selection?

For some operations there should only be one item selected at a time, so the semantics of selection is that choosing new item replaces the old one. While occasionally operation semantics require that there must always be a selected item, often it makes sense that the selection set could be empty. In this case, selection is often implemented as a toggle, where an single action such as clicking switches between select and deselect. For example, when a system is designed to show detail on demand for an item in a single fixed pane, that choice implies the detail selection set should have only one item at a time. A set size of zero can be easily accommodated by leaving the pane blank.

In other operations, it is meaningful to have many items selected at once, so there must be actions for adding items, removing items, and clearing the selection set. An example would be an interface that allows the user to assign items into any of four main groups, each of which is highlighted in a different color. In this example, there also needs to be some way to indicate which group the current selection actions should change.

In the previous example the selection set could contain multiple items but all items within the set were interchangeable. Some operations require a distinction between a primary and secondary selection, for example path traversal in a directed graph from a source to a target node. Once again, some operations require that these sets contain only one item apiece, others may support multi-item sets for one, the other, or both.

6.1.2 Changing Highlighting

We have completely separated the interaction question of selection from the visual encoding question of highlighting the selected items to distinguish them from the unselected ones. Of course, in an actual system design, selection is usually indicated with highlighting, but it is useful to consider the two as independent concerns from a design point of view. While color coding is one of the most common approaches, any of the visual channels discussed in Chapter 5 can be used to highlight items in the display.

6.1.3 Changing Viewpoint: Navigating

A single point of view is not enough to fully understand a sufficiently large or complex dataset. Many interactive visualization systems support a metaphor of navigation, analogous to navigation through the physical world. The principle that underlies all of the specific navigation techniques discussed in Part 9 is that this metaphor can be usefully invoked when the spatial layout stays fixed and what changes is the viewpoint from which the drawing is created. In the language of computer graphics, that change is a transformation of the camera or the viewport.

6.1.4 Changing Spatial Ordering: Sorting

Sorting is a change that reorders the spatial location of items in the display. For example, a common interaction with tables is to sort the rows according to the values in a chosen column. The same principle holds for more exotic visual encodings as well. The power of resorting lies in the privileged status of spatial position as the highest ranked visual channel, as discussed in Chapter 5.3. Reordering data spatially allows us to invoke the pattern finding parts of our visual system to quickly check whether the new configuration conveys new information.

6.1.5 Changing Visual Encoding

All of the previous changes can be considered as parameter changes to specific aspects of a fixed visual encoding. In some systems, the visual encoding itself is not fixed but can be changed by the user. Control of the encoding is particularly common in general-purpose systems designed to flexibly accommodate a large range of possible datasets, rather than special-purpose systems fine-tuned for a very particular task.

6.2 Latency and Feedback

The **latency** of interaction, namely how much time it takes for the system to respond to the user action, matters immensely. Latency is not simply a continuum, where the user's irritation level gradually rises as things take longer and longer. Rather, latency fundamentally affects human response to a system, and our response is best modelled as a series of discrete steps rather than a gradual change. Table 6.1 shows the three most critical bins and their time constants.

Time Constant	Value (in seconds)
perceptual processing	0.1
immediate response	1
unit tasks	10

Table 6.1: Human response to interaction latency changes dramatically at these time thresholds. Redrawn after Card [Card et al. 91], Table 3.

The perceptual processing time constant of one-tenth of a second is relevant for operations such as screen updates. The immediate response time constant of one second is relevant for operations such as visual feedback showing what item that user has selected with a mouse click, or the length of time for an animated transition from one layout to another. The unit task time constant of ten seconds is relevant for breaking down complex tasks into simpler pieces; a good granularity for the smallest pieces is this unit task time.

6.2.1 Visual Feedback

From the user's point of view, the latency of an interaction is the time between their action and some feedback from the system indicating that the operation has completed. In a visualization system, that feedback would most naturally be some visual indication of state change within the system itself, rather than cumbersome approaches such printing out status

indications at the command line or a popup dialog box confirmation that would interfere with the flow of exploration.

The most obvious principle is that the user should indeed have some sort of confirmation that the action has completed, rather than being left dangling wondering whether the action is still in progress, or whether the action never started in the first place (for example, because they missed the target and clicked on the background rather than the intended object). Thus, feedback such as highlighting a selected item is a good way to confirm that the desired operation has completed successfully. In navigation, feedback would naturally come when the user sees the new frame is drawn from the changed viewpoint. Visual feedback should typically take place within the immediate response latency class, of around one second.

Another principle is that if an action could take significantly longer than a user would naturally expect, some kind of progress indicator should be shown to the user. A good rule of thumb for significantly longer is crossing from one latency class into another, as shown in Figure 6.1.

6.2.2 Latency Classes

Successful interaction design for a visualization system depends on having a good match between the latencies of the low-level interaction mechanism, the visual feedback mechanism, the system update time, and the cognitive load of operation itself .

For example, consider the operation of seeing more details for an item and the latency difference between three different low-level interaction mechanisms for doing so. Clicking on the item is slowest, because the user must move the mouse towards the target location, stop the motion in the right place, and press down on the mouse. Mouseover hover, where the cursor is placed over the object for some short period of dwell time but no click is required, is faster because third step does not need to occur. Passover with no dwell time requirement, where the action is triggered by the cursor simply crossing the object, is of course the fastest because the second step is also eliminated and only the first step needs to take place.

For visual feedback, consider three different mechanisms for showing the information. One is showing the information on a fixed detail pane at the side of the screen. In order to see the information, the user's eyes need to move from the current cursor location to the side of the screen, so this operation has relatively high latency for making use of the visual feedback. On the other hand, from a visual encoding point of view, an advantage is that a lot of detail information can be shown without occluding anything else in the main display. A second feedback mechanism is a popup window at the current cursor location, which is faster to use since there is no need to move the eyes away from tracking the cursor. Since placing information

directly in the view might occlude other objects, there is a visual encoding cost to this choice. A third mechanism is a visual highlight change directly in the view, for instance by highlighting all neighbors within the graph that are one hop from the graph node under the cursor through a color change.

System update time is another latency to consider. With tiny datasets stored completely locally, update time will be negligible for any of these options. With larger datasets, the time to redraw the entire view could be considerable unless the rendering framework has been designed to deliver frames at a guaranteed rate. (**Rendering** is the term used in computer graphics for drawing an image.) Similarly, scalable rendering frameworks can support fast update for changing a few items or a small part of the display without redrawing the entire screen, but most graphics systems do not offer this functionality by default. Thus, designing systems to guarantee immediate response to user actions can require significant algorithmic attention.

With distributed datasets, obtaining details may require a round trip from the client to the server, possibly taking several seconds on a congested network. Latency can also be introduced on the input side; while mouse and keyboard information is typically reported very quickly, more exotic input approaches like video-based body tracking or eye tracking might require significant processing time before the output update cycle even begins.

When systems are designed so that all of these latencies are well matched, the user interacts fluidly and can stay focused on high-level goals such as building an internal mental model of the dataset. When there is a mismatch, the user is jarred out of a state of flow [Csikszentmihalyi 91] by being forced to wait for the system.

6.2.3 Tightly Coupled Interaction

6.3 Interactivity Costs

Interactivity has both power and cost. The benefit of interaction is that people can explore a larger information space than can be understood in a single static image. However, a cost to interaction is that it requires human time and attention. If the user must exhaustively check every possibility, use of the visualization system may degenerate into human-powered search. Automatically detecting features of interest to explicitly bring to the user's attention via the visual encoding is a useful goal for the visualization designer. However, if the task at hand could be completely solved by automatic means, there would be no need for a visualization in the first place. Thus, there is always a tradeoff between finding automatable aspects, and relying on the human in the loop to detect patterns.

6.4 Spatial Cognition

Spatial cognition is the study of how people acquire, organize, revise and use knowledge about spatial environments; that is, how we move around the world. [1] Spatial cognition is not yet as well understood as the low-level aspects of human vision, such as the psychophysical characterization of visual channels. Many visualization systems rely on implicit assumptions about spatial cognition, rather than being justified with explicitly articulated statements. Nevertheless, terms like **wayfinding**, meaning the ways that people orient themselves and navigate in physical space, have been widely adopted to describe our navigation in virtual environments. Similarly, the term **mental map** has been used to mean a person's internal model of spatial relationships at many levels, ranging from navigation in real-world cities [Lynch 60] to human interpretation of dynamically changing node-link graphs [Misue et al. 95].

There is considerable evidence that our spatial memory is not a straightforward representation of real-world territory, but rather has systematic distortions [Tversky 92]. Spatial information is stored hierarchically: we are likely to make the mistake that Reno is east of San Diego because of the relative locations of Nevada versus California. Spatial judgements are affected by perspective: distance estimates can be effected simply by asking people to imagine themselves at different locations. Spatial memory is organized around landmarks: we skew distance estimates near landmarks to be closer than to unfamiliar objects. When we learn our way through a new city, we progress from finding landmarks, to remembering explicit paths between them, to eventually reconstructing a fully indexable map. For instance, at first we might be able to get from the clock tower to the church, and from the church to the beach, but would not know the shortcut to get from the tower directly to the beach until gaining a better understanding of the city layout.

6.5 History and Further Reading

Card pioneered the discussion of latency classes for visualization and human-computer interaction [Card et al. 91]. Johnson has an excellent book chapter covering these ideas in his very accessible book on interface design [Johnson 10] (Chapter 12).

Interaction costs are discussed by Lam [Lam 08] and Yi et al. [Yi et al. 07].

Kevin Lynch's influential book Image of the City [Lynch 60] sparked many interpretations of how his framework of real-world navigation might

[1] Vocabulary note: Environmental cognition is a synonym for spatial cognition.

translate into virtual environments, including the visualization-specific derived guidelines of Ware (Chapter 10) [Ware 04].

7

Principles: Slogans and Guidelines

In this chapter, I present a series of design guidelines. Each guideline has a catchy title that distills it into a slogan, for memorability. These guidelines are not set in stone; they are my current attempt to synthesize the current state of knowledge into a more unified whole. Each section also contains a discussion of the limits of their applicability.

7.1 No Unjustified 3D

Many people have the intuition that if two dimensions are good, three dimensions must be better – after all, we live in a three-dimensional world. However, there are many difficulties in visually encoding information with the third spatial dimension, depth, which has important differences from the two planar dimensions.

The cues that convey depth information to our visual system include occlusion, perspective distortion, shadows and lighting, familiar size, stereoscopic disparity, and others. This section discusses the costs of these **depth cues** in a visual encoding context, and also the challenges of text legibility given current display technology. It then discusses in what situations the benefits of showing depth information could outweigh these costs, and the need for justification that the situation has been correctly analyzed.

In brief, 3D is easy to justify when the user's task involves shape understanding of inherently three-dimensional structures. In this case, which frequently occurs with inherently spatial data, the benefits of 3D absolutely outweigh the costs, and designers can use the many interaction techniques designed mitigate those costs.

In all other contexts, the use of 3D needs to be carefully justified. In most cases, rather than choosing a visual encoding using three dimensions of spatial position, a better answer is to visually encode using only two dimensions of spatial position. Often an appropriate 2D encoding follows

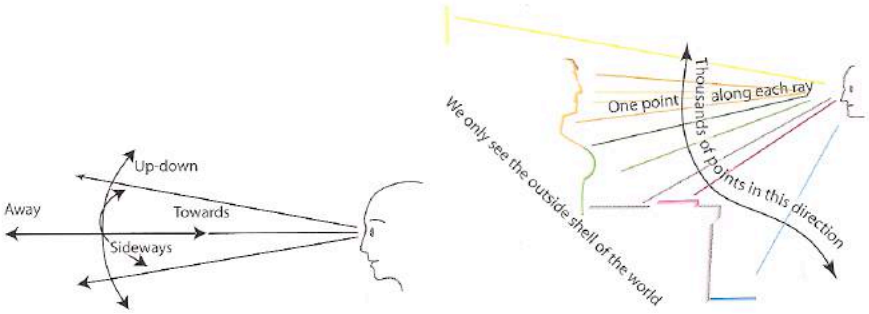


Figure 7.1: Seeing in 2.05D. (a) The sideways and up/down axes are fundamentally different from the towards-away depth axis. (b) Along the depth axis we can see only one point for each ray, as opposed to millions of rays for the other two axes. From [Ware 08], top and bottom figures on page 44.

from a different choice of data abstraction, where the original dataset is transformed by computing derived data.

7.1.1 The Power of the Plane, The Danger of Depth

All of the above discussion about spatial position has been about planar spatial position. The psychophysical power law exponents for accuracy are different for planar position judgements than for depth position judgements. Our highly accurate length perception capability, with the linear n value of 1.0, is only for planar spatial position. For depth judgements of visual distance, n was measured as 0.67 [Stevens 57]. Our judgement of length along the depth axis is even worse than for planar area. This phenomenon is not surprising when considered mathematically, because depth into the scene orthogonal to the picture plane is scaled nonlinearly in depth, as opposed to linearly in the horizontal and vertical dimensions, so distances and angles are distorted [St. John et al. 01].

Although many people would argue that we live in a three-dimensional world, from a perceptual point of view that intuition is misleading. We do not really live in 3D, or even $2\frac{1}{2}$D: we *see* in 2.05D [Ware 08]. Consider what we see when we look out at the world along a ray from some fixed viewpoint, as in Figure 7.1(a). There is a major difference between the towards-away depth axis and the other two axes, sideways and up-down. There are millions of rays that we can see along these two axes by simply moving our eyes, to get information about the nearest opaque object. This information is like a two-dimensional picture, often called the **image**

plane. In contrast, we can only get information at one point along the depth axis for each of these rays, as in Figure 7.1(b). This phenomenon is called **line-of-sight ambiguity** [St. John et al. 01]. In order to get more information about what is hidden behind the closest objects shown in the image plane, we would need to move our viewpoint or the objects. At best we could change the viewpoint by simply moving our head, but in many cases we would need to move our body to a very different position. Thus, the label of 2.05D accurately reflects the reality that the amount of information we have about the third dimension is a tiny fraction of what we have about the two-dimensional image plane.

7.1.2 Occlusion Hides Information

The most powerful depth cue is **occlusion**, where some objects cannot be seen because they are hidden behind others. The visible objects are interpreted as being closer than the occluded ones. The occlusion relationships between objects change as we move around; this **motion parallax** allows us to build up an understanding of the relative distances between objects in the world.

When people look at realistic scenes made from familiar objects, the use of motion parallax typically does not impose cognitive load or require conscious attention. In synthetic scenes, navigation controls that allow the user to change the 3D viewpoint interactively invoke the same perceptual mechanisms to provide motion parallax. In sufficiently complex scenes where a single fixed viewpoint does not provide enough information about scene structure, interactive navigation capability is critical for understanding 3D structure. In this case, the cost is time: interactive navigation takes longer than inspecting a single image.

The overarching problem with occlusion in the context of visual encoding is that presumably-important information is hidden, and discovering it via navigation has a time cost. In realistic environments, there is rarely a need to inspect all hidden surfaces. However, in a visualization context, the occluded detail might be critical. It is especially likely to be important when using spatial position as a visual channel for abstract, nonspatial data.

Moreover, if the objects have unpredictable and unfamiliar shapes, understanding the three-dimensional structure of the scene can be very challenging. In this case there can be appreciable cognitive load because people must use internal memory to remember the shape from previous viewpoints, and internally synthesize an understanding of the structure. This case is common when when using the spatial position channels for visual encoding. Figure 7.2 illustrates the challenges of understanding the topological structure of a node-link graph laid out in 3D, as an example of the un-

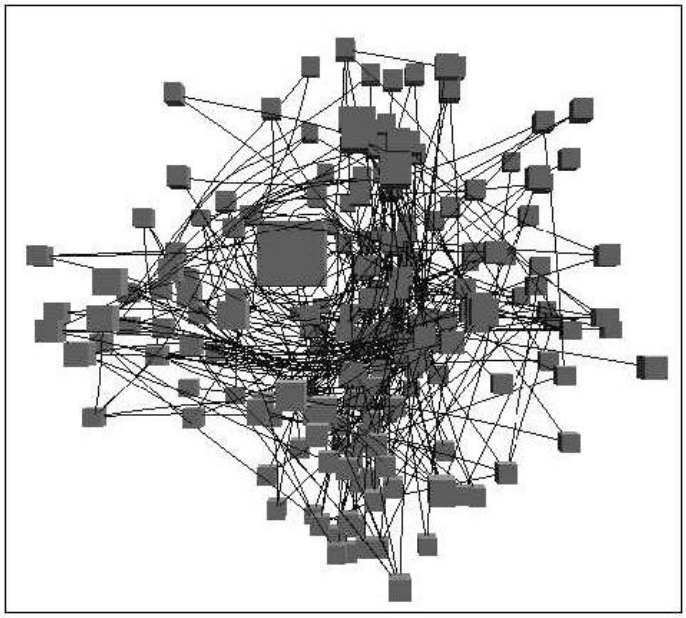


Figure 7.2: Resolving the 3D structure of the occluded parts of the scene is possible with interactive navigation, but that takes time and imposes cognitive load, even when sophisticated interaction techniques are used as in this example of a node-link graph laid out in 3D space. From [Carpendale et al. 96], Figure 21.

familiar structure that arises from visually encoding an abstract dataset. Synthesizing an understanding of the structure of the linkages hidden from the starting viewpoint shown here is likely to take a considerable amount of time. While sophisticated interaction techniques have been proposed to help users do this synthesis more quickly than with simple realistic navigation, lowering the time cost, visualization designers should always consider whether the benefits of 3D are worth the costs.

7.1.3 Perspective Distortion Undercuts Power of the Plane

Perspective distortion is the phenomenon that distant objects appear smaller and change their planar position on the image plane. Imagine a photograph looking along railroad tracks: although they are of course parallel, they appear to draw together as they recede into the distance. Although the tracks have the same width in reality, measuring with a ruler on the photograph itself would show that in the picture the width of the

Figure 7.3: With perspective distortion, the power of the planar spatial position channel is lost, as is the size channel. From [Mukherjea et al. 96], Figure 1.

nearby track is much greater than that of the distant track. [1]

One of the major breakthroughs of Western art was the Renaissance understanding of the mathematics of perspective to create very realistic images, so many people think of perspective as a good thing. However, in the context of visually encoding abstract data, perspective is a very bad thing! Perspective distortion is one of the main dangers of depth because the power of the plane is lost; it completely interferes with visual encodings that use the planar spatial position channels, and also the size channel. For example, it is more difficult to judge bar heights in a 3D bar chart than in multiple horizontally aligned 2D bar charts. Foreshortening makes direct comparison of bar heights difficult.

Figure 7.3 shows another example where size coding in multiple dimensions is used for bars that recede into the distance in 3D on a ground plane. The result of the perspective distortion is that the bar sizes cannot be directly compared as a simple perceptual operation.

7.1.4 Other Depth Cues

7.1.4.1 Familiar/Relative Size In realistic scenes, one of the depth cues is the size of familiar objects. We roughly know the size of a car, so when we see one at a distance we can estimate the size of a nearby unfamiliar object. If all objects in the scene are visually encoded representations of abstract information, we do not have access to this strong depth cue.

7.1.4.2 Shadows and Shading The depth cues of shadows and surface shading also communicate depth and three-dimensional structure information. Cast shadows are useful for resolving depth ambiguity because they allow us to infer the height of an object with respect to a ground plane. Shading and self-shadowing show the three-dimensional shape of an object. One problem with using these lighting-based cues when visualizing abstract data is that they create visual clutter that distracts the viewer's attention from the meaningful parts of the scene that represent information. Another

[1] Vocabulary note: This phenomenon is also known as *foreshortening*.

problem is that cast shadows, regions of self-shadowing, or highlights could be mistaken by the viewer for true marks, in the image theory sense of a geometric primitives that are the substrate for the visual channels showing attribute information. Cast shadows could also cause problems by occluding true marks. The final problem is that surface shading effects interfere with the color channels: highlights can change the hue or saturation, and shadows change the luminance.

7.1.4.3 Stereo Stereoscopic depth is a cue that comes from the disparities between two images made from two camera viewpoints slightly separated in space, just like our two eyes are. In contrast, all of the previous discussion pertained to pictoral cues from a single camera. Although many people assume that stereo vision is the strongest depth cue, it is in fact a relatively weak one compared to the others listed above, and contributes little for distant objects. Stereo depth cues are most useful for nearby objects that are at roughly the same depth, providing guidance for manipulating things within arm's reach.

Stereo displays, which deliver slightly different image each for each of our two eyes, do help people better understand resolve depth. Conveniently, they do not directly interfere with any of the main visual channels. Stereo displays do indeed improve the accuracy of depth perception compared to single-view displays – but even still depth cannot be perceived with the accuracy of planar position. Of course, stereo cannot solve any of the problems associated with perspective distortion.

7.1.4.4 Atmospheric Perspective The relatively subtle depth cue of atmospheric perspective, where the color of distant objects is shifted towards blue, would conflict with color encoding.

7.1.5 Tilted Text Isn't Legibile

Another problem with the use of 3D is dramatically impaired text legibility with most standard graphics packages that use current display technology [Grossman et al. 07]. Text fonts have been very carefully designed for maximum legibility when rendered on the grid of pixels that makes up a 2D display, so that characters as little as nine pixels high are easily readable. Although hardware graphics acceleration is now nearly pervasive, so that text positioned at arbitrary orientations in 3D space can be rendered *quickly*, this text is usually not rendered *well*. As soon as a text label is tilted in any way off of the image plane, it typically becomes blocky and jaggy. The combination of more careful rendering and very high-resolution displays of many hundred of dots per inch may solve this problem in the future, but legibility is a major problem today.

7.1.6 3D Benefits

The great benefit of using 3D comes when the viewer's task fundamentally requires understanding the three-dimensional geometric structure of objects or scenes. In almost all of these cases, a 3D view with interactive navigation controls to set the 3D viewpoint will allow users to construct a useful mental model of dataset structure more quickly than simply using several 2D axis-aligned views. For these tasks, all of the costs of using 3D discussed above are outweighed by the benefit of helping the viewer build a mental model of the 3D geometry.

For example, although people can be trained to comprehend blueprints with a top view and two side views, synthesizing the information contained within these views to understand what a complex object looks like from some arbitrary 3D viewpoint is a difficult problem that incurs significant cognitive and memory load. The 2D blueprint views are better for the task of accurately discriminating the sizes of building elements, which is why they are still heavily used in construction. However, there is considerable experimental evidence that 3D outperforms 2D for shape understanding tasks [St. John et al. 01].

Most tasks that have inherently 3D spatial data after the abstraction stage fall into this category. Some classical examples are fluid flow over an airplane wing, a medical imaging tomography dataset of the human body, or molecular interaction within a living cell.

7.1.7 Justification and Alternatives

The question of whether to use two or three channels for spatial position has now been extensively studied. When computer-based visualization began in the late 1980s, and interactive 3D graphics was a new capability, there was a lot of enthusiasm for 3D representations. As the field matured, researchers began to better appreciate the costs of 3D approaches when used for abstract datasets [Ware 01]. By now, the use of 3D for abstract data requires careful justification. In many cases, a different choice at the abstraction or visual encoding levels would be more appropriate.

A good example is a system designed to browse time series data. The specific dataset it shows below is the amount of power used in an office building over the course of the day, with measurements taken daily over one full year [van Wijk and van Selow 99]. The authors compare a straightforward 3D representation with a carefully designed approach using linked 2D views, which avoids the problems of occlusion and perspective distortion. Figure 7.4(a) shows a 3D representation created directly from the original time-series data, where each cross-section is a 2D time series curve showing power consumption for one day, with one curve for each day of the

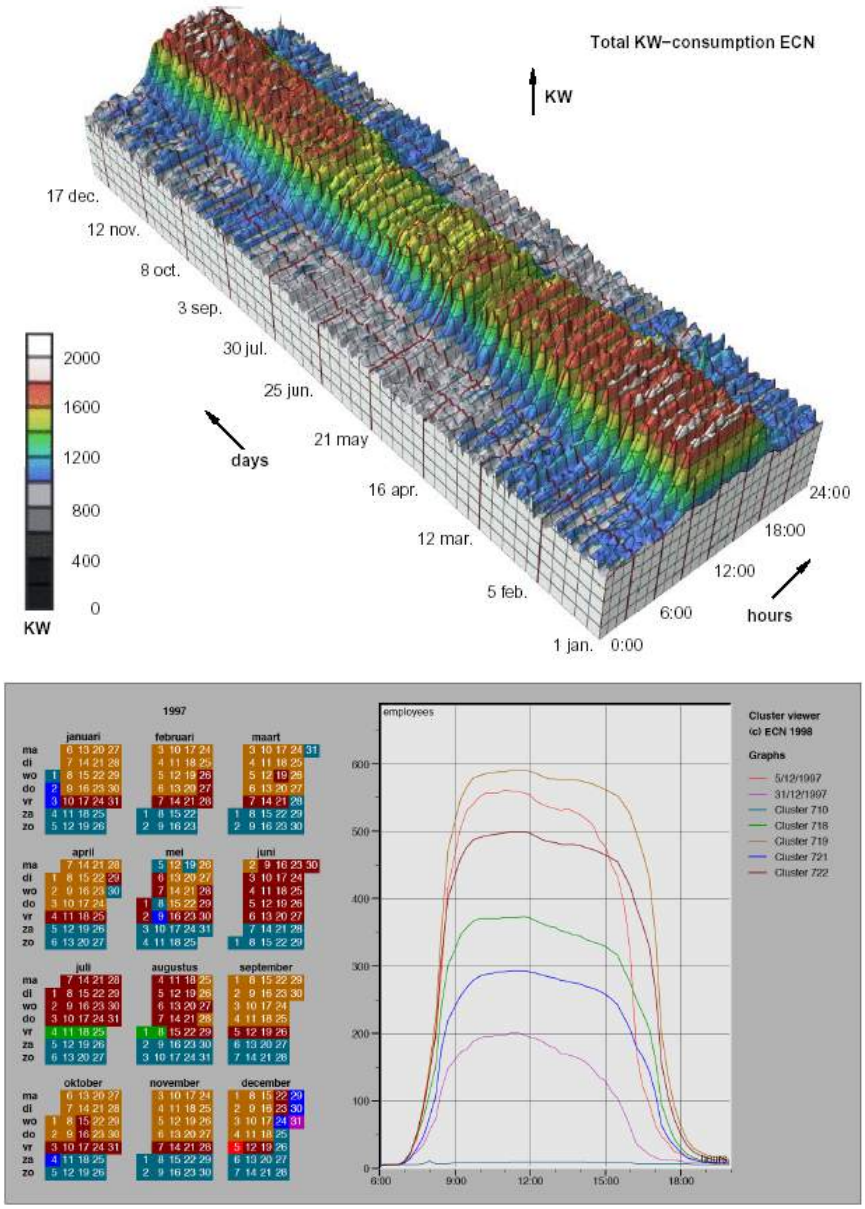


Figure 7.4: 3D versus 2D. (a) A 3D representation of this time series dataset introduces the problems of occlusion and perspective distortion. (b) The linked 2D views of derived aggregate curves and the calendar allow direct comparison and show more fine-grained patterns. From [van Wijk and van Selow 99], Figures 1 and 4.

year along the extruded third axis. We can only see large-scale patterns such as the higher consumption during working hours and the seasonal variation between winter and summer.

In their final visualization they choose a different data abstraction by creating a derived dataset using **agglomerative clustering**, where the most similar curves are merged together into a cluster that can represented by the average of the curves within it. This process is repeated until there is a hierarchical structure of clusters. Figure 7.4(b) shows a single aggregate curve for each of the highest-level groups in the clustering in the window on the right of the display. There are few enough of these aggregate curves is that they can all superimposed in the same 2D image without excessive visual clutter. Direct comparison between the curve heights at all times of the day is easy because there is no perspective distortion or occlusion.

On the left side of Figure 7.4(b) is a calendar view, where the pattern of which days belong to which clusters is shown with color coding that matches the curves on the right side. Calendars are a very traditional and successful way to show temporal patterns. The same large-scale patterns of seasonal variation between summer and winter are still visible, but smaller-scale patterns that are difficult or impossible to spot in the top view are also revealed. This related dataset shows the number of people in the building, rather than the power consumption. In this Dutch calendar, weeks are vertical strips with the weekend at the bottom. We can identify weekends and holidays as the nearly flat teal curve where nobody is in the building, and normal weekdays as the topmost tan curve with a full house. Summer and Fridays during the winter are the brown curve with one hundred fewer people, and Fridays in the summer are the green curve with nearly half of the employees gone. The blue and magenta curves show days between holiday times where most people also take vacation. The red curve shows the unique Dutch holiday of Santa Claus day, where everybody gets to leave work an hour early. This design is an example of linked multiple views, which will be discussed in more detail in Chapter 8.5.

While unbridled enthusiasm for 3D is no longer common, there are indeed situations where its use is justifiable even for abstract data. Figure 7.5 shows an example that is similar on the surface to the previous one, but in this case 3D is used with care and the design is well justified [Lopez-Hernandez et al. 10]. In this system for visualizing oscilloscope time-series data, the user starts by viewing the data using the traditional eye diagram where the signal is wrapped around in time and shown as many overlapping traces. Users can spread the traces apart using the metaphor of opening a drawer. This drawer interface does use 3D, but with many constraints. Layers are orthographically projected and always face the viewer. Navigation complexity is controlled by automatically zooming and framing as the user adjusts the drawer's orientation.

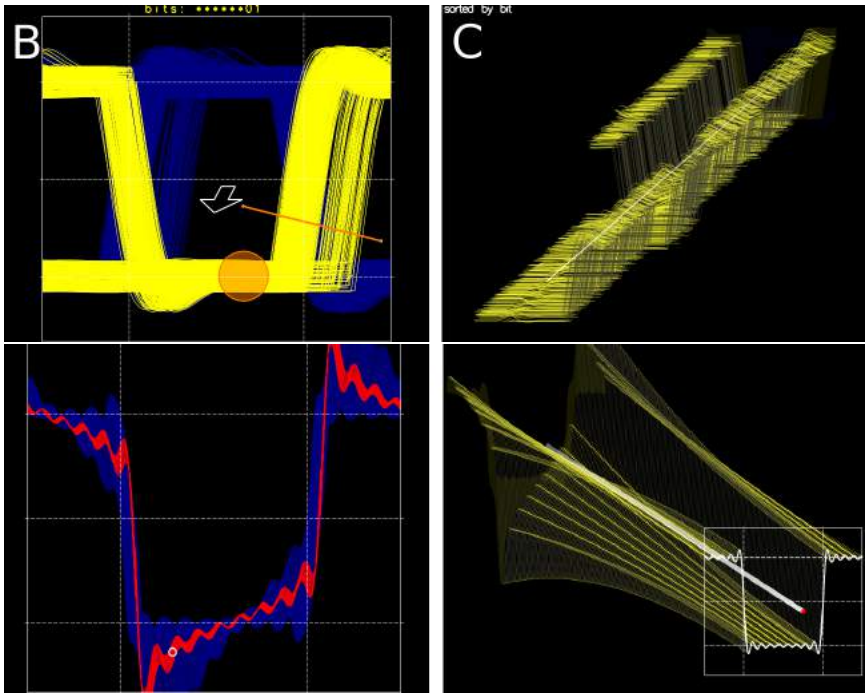


Figure 7.5: Careful use of 3D. **Top**: The user can evolve the view from the traditional overlapping eye diagram with the metaphor of opening a drawer. **Bottom**: The interaction is carefully designed to avoid the difficulties of unconstrained 3D navigation. From [Lopez-Hernandez et al. 10], Figures 3 and 7.

7.1.8 Empirical Evidence

Empirical experiments are critical in understanding user performance, especially because of the well-documented dissociation between stated preference for 3D and actual task performance [Andre and Wickens 95]. Experimental evidence suggests that 3D interfaces are better for shape understanding, whereas 2D are best for relative position tasks: those that require judging the precise distances and angles between objects [St. John et al. 01]. Most tasks involving abstract data do not benefit from 3D; for example, an experiment comparing 3D cone trees to an equivalent 2D tree browser found that the 3D interaction had a significant time cost [Cockburn and McKenzie 00].

Designing controlled experiments that untangle the efficacy of specific interfaces that use 3D can be tricky. Sometimes the goal of the experi-

menter is simply to compare two alternative interfaces that differ in many ways; in such cases it is dangerous to conclude that if an interface that happens to be 3D outperforms another that happens to be 2D, it is the use of 3D that made the difference. In several cases, earlier study results that were interpreted as showing benefits for 3D were superseded by more careful experimental design that eliminated uncontrolled factors. For example, the 3D Data Mountain interface for organizing web page thumbnail images was designed to exploit human spatial cognition, and was shown to outperform the standard 2D Favorites display in Internet Explorer [Robertson et al. 98]. However, this study left open the question of whether the benefit was from the use of 3D, or the use of the data mountain visual encoding, namely a spatial layout allowing immediate access to every item in each pile of information. A later study compared two versions of Data Mountain, one with 3D perspective and one in 2D, and no performance benefit for 3D was found [Cockburn and McKenzie 01]. Another empirical study that found no benefits for 3D landscapes created to reflect the density of a 2D point cloud, compared to simply showing the point cloud in 2D, is discussed in Section **??**.

7.2 Eyes Over Memory

Using our eyes to switch between different views that are visible simultaneously has much lower cognitive load than consulting our memory to compare a current view to what was seen before. Many interaction techniques implicitly rely on the internal use of memory, and thus impose cognitive load on the viewer. Consider navigation within a single view, where the display changes to show the scene from a different viewpoint. Maintaining a sense of orientation implicitly relies on using internal resources, either by keeping track of past navigation choices (*I zoomed into the nucleus*) or by remembering past views (*earlier all the stock options in the tech sector were in the top corner of the view*). In contrast, having a small overview window with a rectangle within it showing the position and size of the current camera viewport for the main view is a way to show that information through an external representation easily consulted by moving the eyes to that region of the screen, so that it can be read off by the perceptual system instead of remembered.

7.2.1 Animation vs. Side-By-Side Views

Some animation-based techniques also impose significant cognitive load on the viewer because of implicit memory demands. Animation is an overloaded word that can mean many different things considered through the

lens of visualization encoding and interaction. We distinguish between these three definitions:

- narrative storytelling, as in popular movies;
- transitions from just one state to another;
- video-style playback of a multiframe sequence: play, pause, stop, rewind, and step forward/back.

Some people have the intuition that because animation is a powerful storytelling medium for popular movies, it should also be suitable in a visualization context. However, the situation is quite different. Successful storytelling requires careful and deliberate choreography to ensure that action is only occurring in one place at a time and the viewer's eyes have been guided to ensure that they are looking in the right place. In contrast, a dataset animation might have simultaneous changes in many parts of the view.

Animation is extremely powerful when used for transitions between two dataset configurations because it helps the user maintain context. There is considerable evidence that animated transitions can be more effective than jump cuts, by helping people track changes in object positions or camera viewpoints. These transitions are most useful when only a few things change; if the number of objects that change between frames is large, people will have a hard time tracking everything that occurs.

Although jump cuts are hard to follow when only seen once, giving the user control of jumping back and forth between just two frames can be effective for detecting whether there is a localized change between two scenes. This *blink comparator* technique was used by the astronomer who found Pluto.

Finally, we consider animations as sequences of many frames, where the viewer can control the playback using video-style controls of play, pause, stop, rewind, and sometime single-step forward or backwards frame by frame. We distinguish animation from true interactive control, for example navigation by flying through a scene. With animation the user does not directly control what occurs, only the speed at which the animation is played.

The difficulty of multi-frame animations is that making comparisons between frames that do not adjoin relies on internal memory of what previous frames looked like. If changes only occur in one place at a time, the demands on attention and internal memory are small. However, when many things change all over the frame, and there are many frames, we have a very difficult time in tracking what happens. Giving people the ability to

pause and replay the animation is much better than only seeing it a single time straight through, but does not fully solve the problem.

For tasks requiring detailed comparison across many frames, seeing all the frames at once side by side can be more effective than animation. The number of frames must be small enough that the details within each can be discerned, so this approach is typically suitable for dozens but not hundreds of frames with current display resolutions. The action should be also segmented into meaningful chunks, rather than keyframes that are randomly chosen. Many visualization techniques that use multiple views exploit this observation, especially small multiples 8.5.2.

7.2.2 Change Blindness

The underlying reason that eyes beat memory is the design of our visual system, which works by querying the world around us using our eyes. Our visual system works so well that most people have the intuition that we have detailed internal memory of the visual field that surrounds us. However, we do not. Our eyes dart around, gathering information just in time for our need to use it, so quickly that we do not typically notice this motion at a conscious level.

Change blindness is the phenomenon that we fail to notice even quite drastic changes if our attention is directed elsewhere. For example, experimenters set up a real-world interaction where somebody was engaged by a stranger who asked directions, only to be interrupted by people carrying a door who barged in between them. The experimenters orchestrated a switch during this visual interruption, replacing the questioner with another person. Remarkably, most people did not notice, even when the new questioner was dressed completely differently – or was a different gender than the old one!

Although we are very sensitive to changes at the focus of our attention, we are surprisingly blind to changes when our attention is not engaged. The difficulty of tracking complex and widespread changes across multi-frame animations is one of the implications of change blindness for visualization.

7.3 Resolution Over Immersion

Immersive environments emphasize simulating realistic interaction and perception as closely as possible through technology such as stereo imagery delivered separately to each eye to enhance depth perception, full six-degree-of-freedom head and position tracking so that the displays respond immediately to the user's physical motion of walking and moving the head around. The most common display technology is head-mounted displays,

or small rooms with rear-projection displays on walls, floor, and ceilings. Immersion is most useful when a sense of presence is an important aspect of the intended task. With current display hardware, there is a tradeoff between **resolution**, the number of available pixels divided by the display area, and **immersion**, the feeling of presence in virtual reality. The price of immersion is resolution; these displays cannot show as many pixels as state-of-the-art desktop displays of the equivalent area. The number of pixels available on a computer display is a limited resource that is usually the most critical constraint in visualization design, as discussed in Chapter 1.9. Thus, it is extremely rare that immersion is worth the cost in resolution.

Another price of immersion is the integration of visualization with the rest of a user's typical computer-based workflow. Immersive display environments are almost always special-purpose settings that are a different physical location than the user's workspace, requiring them to leave their usual office and go to some other location, whether down the hall or in another building. In most cases users stand rather than sit, so working for extended periods of time is physically taxing compared to sitting at a desk. The most critical problem is that they do not have access to their standard working environment of their own computer system. Without access to the usual input devices of mouse and keyboard, standard applications such as web browsing, email reading, text and spreadsheet editing, and other data analysis packages are completely unusable in most cases, and very awkward at best. In contrast, a visualization system that fits into a standard desktop environment allows integration with the usual workflow and fast task switching between visualization and other applications.

A compelling example of immersion is the use of virtual reality for phobia desensitization; somebody with a fear of heights would need a sense of presence in the synthetic environment in order to make progress. However, this example is not an application of visualization, since the goal is to simulate reality rather than to visually encode information. The most likely case where immersion would be helpful for visualization is when the chosen abstraction includes 3D spatial data. Even in this case, the designer should consider whether a sense of presence is worth the penalties of lower resolution and no workflow integration. It is very rare that immersion would be necessary for nonspatial, abstract data. As we discuss in Section 7.1, using 3D for visual encoding of abstract data is the uncommon case that needs careful justification. The use of an immersive display in this case would require even more careful justification.

7.4 Function First, Form Next

The best visualization designs should shine in terms of both form and function; that is, they should be both beautiful and effective. Nevertheless, in this book, I focus on function.

My rationale is that given an effective but ugly design, it's possible to refine the form to make it more beautiful while maintaining the base of effectiveness. Even if the original designer of the visualization has no training in graphic design, collaboration is possible with people who do have that background.

In contrast, given a beautiful and ineffective design, you will probably need to toss it out and start from scratch. Thus, I don't advocate a "form first" approach, because progressive refinement is usually not possible. My argument mirrors the claims I made in the first chapter about the size of the visualization design space and the fact that most designs are ineffective.

Equally important is the point that I don't advocate "form never": visual beauty does indeed matter, given that visualization makes use of human visual perception. Given the choice of two equally effective systems, where one is beautiful and one is ugly, people will prefer the better form. Moreover, good visual form enhances the effectiveness of visual representations.

I don't focus on teaching the principles and practice of graphic design in this book because they are covered well by many other sources. I focus on the principles of visualization effectiveness because of the lack of other resources.

7.5 Get It Right In Black And White

Maureen Stone has advocated the slogan **Get It Right In Black And White** as a design guideline for effective use of color [Stone 10]. Specifically, the advice is to encode the most important attribute with the luminance channel to ensure adequate luminance contrast, and consider the hue and saturation channels as secondary sources of information. This slogan is clearly motivated by visual encoding principles discussed in Section 5.4.2.

7.6 Further Reading

St John et al. discuss the issues of 2D versus 3D in depth [St. John et al. 01], with references to many previous studies in the human factors and air traffic control literature including the extensive work of Wickens.

Lasseter's influential paper on incorporating the principles of hand-drawn animation into computer graphics discusses the importance of choreography to guide the viewer's eyes during narrative storytelling [Lasseter 87].

Tversky et al. present a meta-review of animation, showing that many seemingly promising study results are confounded by attempts to compare incommensurate situations [Tversky et al. 02]. They find that small multiples are better than animation if equivalent information is shown and the segmentation is carefully chosen. Plumlee and Ware [Plumlee and Ware 06] also discuss the costs of multiple views versus navigating within a single view.

For more on animated transitions, see Heer et al. [Heer and Robertson 07] and Robertson et al. [Robertson et al. 08].

For more on the fascinating phenomenon of change blindness, see Simons [Simons 00].

A very accessible place to start for basic graphic design guidelines is The Non-Designer's Design Book by Robin Williams [Williams 08].

Part IV

Methods

This part of the book covers the fundamental methods that serve as the building blocks of visualization. Their categorization provides a general lens that can be used to analyze previously proposed techniques and as a framework for designing a new technique. The taxonomy has two top-level categories: view composition, covered in Chapter 8, and data reduction, covered in Chapter 9.

Chapter 8 covers the fundamental ways to create a view. The taxonomy begins with the use of space and color channels. Space can be used to express values; to separate, align, and order regions; and to use given spatial data. Spatial layouts can be rectilinear, parallel, or radial; they can also be spacefilling and/or dense. Marks can represent items or links. Colormaps can encode a single categorical or ordered attribute; bivariate colormaps are possible, but tricky to use effectively. Views can be combined in three main ways: side by side, superimposed on top of each other, or with a single view that changes over time. There are four main questions to consider when coordinating multiple views: whether encoding channels are shared; whether data is shared, a subset, or a disjoint partition; whether navigation is synchronized, and whether views are linked with explicit marks.

Chapter 9 covers methods to reduct the amount of data to show. It begins with filtering and aggregation methods for both items and attributes. It continues with navigation including both geometric and semantic zooming and camera-oriented attribute reduction. It concludes the family of methods for combining focus and contextual information within a single view by a combination of filtering and aggregation. These focus+context methods include both the use of selective filtering and of geometric distortion.

Chapter **??** TODO

8

Making Views

This chapter covers the analysis framework for methods of making views, summarized in Figure 9.1. It starts with ways to use space, continues with the use of color, and ends with combining and coordinating views.

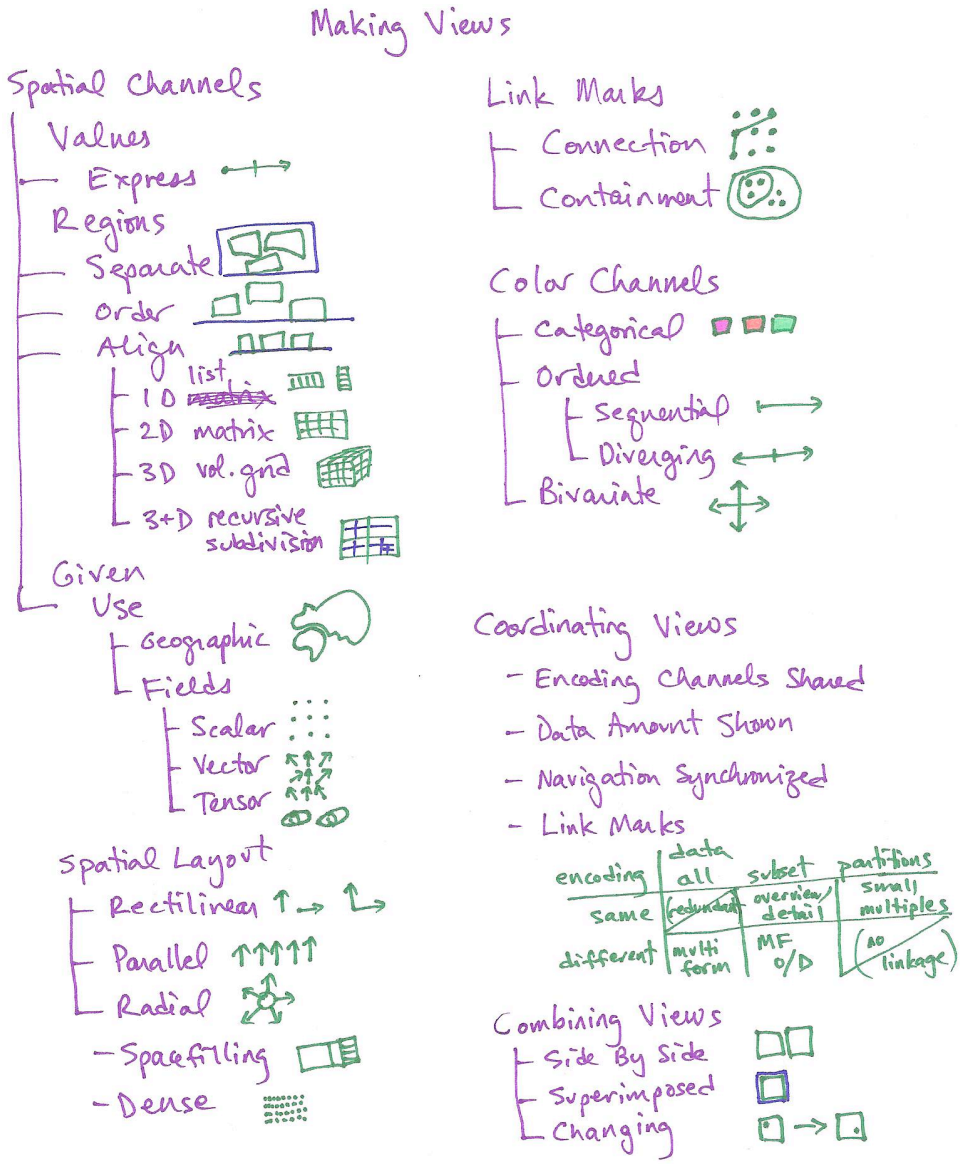


Figure 8.1: Framework of methods for making views.

8.1 Using Space

The most central question in the view-making methods framework is about how space is used. This centrality is motivated by many of the visual encoding principles discussed in Chapter 5, particularly the expressiveness and effectiveness principles. The visual channels related to spatial position are the only ones that are effective for encoding all attribute types, and moreover they are the most effective channels for both ordered and categorical attributes.

The methods framework distinguishes five different ways to use space: to express values; to separate, order, and align regions; and to use it for given spatial attributes.

These uses depend on the semantics of the attributes: whether they are categorical or ordered, and whether they are keys – independent attributes that can be used as a unique index to look up items in a table – or values – dependent attributes that are the values in the cells of a table.

See Section 4.4.1 for more on keys and values.

8.1.1 Quantitative Values: Express

Using space to express quantitative attributes is a straightforward use of the spatial position channel to visually encode data. For quantitative attributes, which must always have value semantics, the attribute is mapped to spatial position along an axis. In the simple case of a single attribute, each item is encoded with a mark at some position along the axis. Additional attributes might also be encoded on the same mark with other non-spatial channels such as color and size. In the more complex case, a composite object with internal structure is drawn; these **glyphs** are made up of multiple marks, with subregions that are visually encoded differently, to show multiple attributes at once.

Example: Scatterplots

Scatterplots encode two quantitative value variables using both the vertical and horizontal spatial position channels, and the mark type is necessarily a point. Figure 8.2 shows an example of demographic data, plotting infant mortality on the vertical axis against life expectancy on the horizontal axis. Scatterplots are often augmented with color coding to show an additional attribute. Size coding can also portray yet another attribute; size-coded scatterplots are sometimes called **bubble plots**.

Scatterplots are effective for the general tasks of overview and characterizing distributions, and specifically for finding outliers and extreme values. Scatterplots are also highly effective for judging the correlation

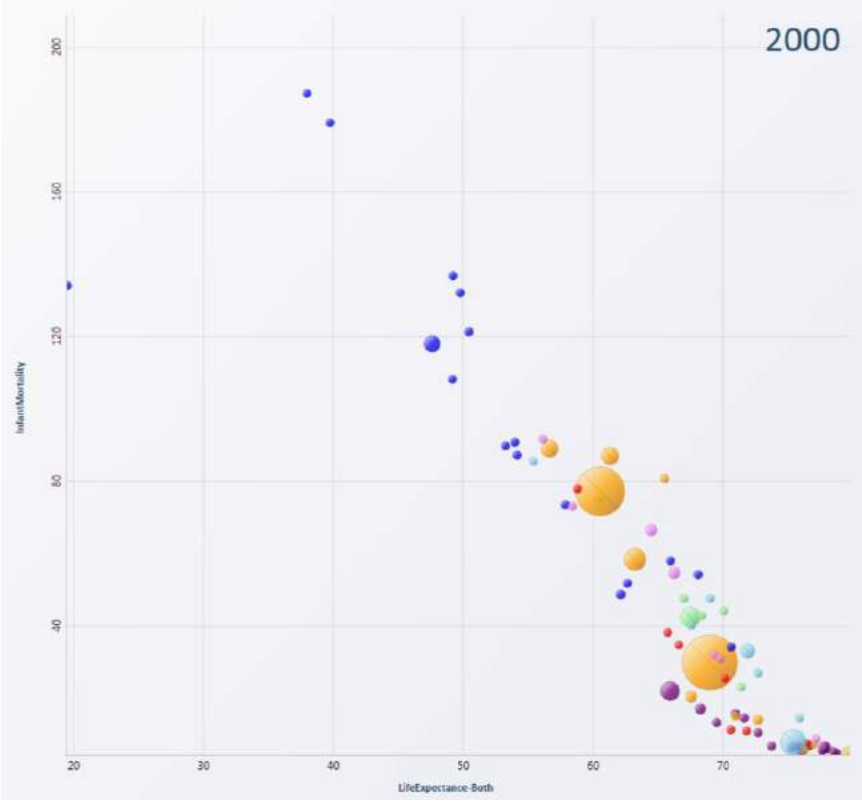


Figure 8.2: This scatterplot chart uses one point mark for each country, with horizontal and vertical spatial position encoding the primary information of life expectancy and infant mortality. The color channel is used for the country category and the size channel for country population. From [Robertson et al. 08], Figure 1c.

between two variables, because that task corresponds the easy perceptual judgements of noticing whether the points form a line along the diagonal. The stronger the correlation, the closer the points fall along a perfect diagonal line; positive correlation is an upward slope, and negative is downward. Figure 8.2 shows a highly negatively correlated dataset. When judging correlation is the primary intended task, the derived data of a calculated regression line is often superimposed on the raw scatterplot of points, as in Figure 1.1.

The table below summarizes this analysis in terms of the data types and semantics, and the visual encoding. All of the subsequent technique

and system examples will end with a similar summary table.

Technique	**scatterplot**
Data Types	table: 2 quantitative value attributes
View Comp.	mark: point, channels: express value with horizontal spatial position, express value with vertical spatial position

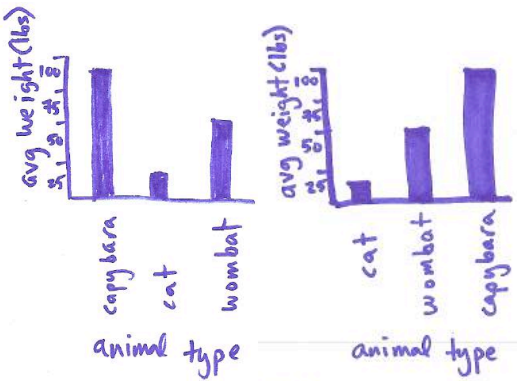
. . . . . .

8.1.2 Categorical Regions: Separate, Order, and Align

The use of space to encode categorical attributes is more complex than the simple case of quantitative attributes. The problem is that spatial position is an ordered *how much* visual channel, whereas categorical attributes have unordered *what* semantics. Although the separation of space into distinct regions can be considered as an unordered *what* channel, those regions must be given spatial positions on the plane in order to draw any specific picture.

The problem becomes easier to understand by breaking down the distribution of regions into three operations: separating into regions, aligning the regions, and ordering the regions. The separation is done by the categorical attribute, but the alignment and ordering is done by some other attribute that is ordered. The separation, done according to the categorical attribute, must always occur. Alignment of the regions is not strictly required but is very common. The ordered attribute used to control the alignment *between* regions is sometimes, but not always, the same one that is used to encode the spatial position of items *within* the region. Finally, the attribute to order the regions must have ordered semantics, and thus it cannot be the categorical one that was used to do the separation. To summarize, regions are separated by a categorical attribute, almost always aligned by the value attribute used for encoding in the other spatial direction, and then ordered by that or some other value attribute.

Choosing an ordering when encoding categorical attributes goes beyond the creation of a static visual encoding. The interaction method of **reordering** is a powerful way to dynamically explore datasets, where the user interactively chooses which attribute to use for ordering spatial positions. It can be used with any categorical attribute. In contrast, reordering does not make sense for ordered attributes, which have a given order already.



(a) (b)

Figure 8.3: Bar charts showing one key attribute, `species`, and one value attribute, `weight`. They use aligned vertical spatial position with line marks distributed along the horizontal spatial axis. (a) Marks ordered alphabetically according to species name. (b) Marks ordered by the weight attribute used for bar heights.

8.1.3 Keys

Some datasets only have attributes with value semantics. However, it would be rare to visually encode a dataset that has only key attributes: the keys are always used in the context of one or more value attributes that need to be encoded. Keys are necessarily categorical or ordinal, not quantitative, thus they can be used to define a region of space for each item in which one or more value attributes are shown. The unique values for a categorical or ordered attribute are called **levels**, to avoid the confusion of overloading the term *value*.

8.1.3.1 1 Key: List With a single key, separating into regions using that key yields one region per item. The regions are typically aligned in a one-dimensional list, either horizontal or vertical. The view itself covers a two-dimensional area: the aligned list of items stretches across one of the spatial dimensions, and the region in which the values are shown stretches across the other.

Example: Bar charts

The well-known bar chart technique is a simple initial example. Figure 8.3 shows a bar chart of approximate weights on the vertical axis for

each of three animal species on the horizontal axis. Analyzing the visual encoding, **bar charts** use a line mark and encode a quantitative value attribute with one spatial position channel. The other attribute shown in the chart, animal species, is a key categorical attribute. Each line mark is indeed in a separate region of space. These line marks are all aligned within a common frame, so that the highest-accuracy aligned position channel is used rather than the lower-accuracy unaligned channel. In Figure 8.3a the regions are ordered alphabetically by species name. Formally, the alphabetical ordering of the names should be considered as a derived attribute. This frequent default choice does have the benefit of making lookup by name easy, but it often hides what could be meaningful patterns in the dataset. Figure 8.3b shows this dataset with the regions ordered by the values of the same value attribute that is encoded by the bar heights, animal weight. This kind of data-driven ordering makes it easier to see dataset trends.

Technique	**bar chart**
Data Types	table 1 quantitative value attrib, 1 key categorical attrib
View Comp.	line mark. aligned position: express value attrib. position: key attrib

8.1.3.2 2 Keys: Matrix Datasets with two keys are often aligned in a two-dimensional matrix where one key is distributed along the rows and the other along the columns, so a rectangular cell in the matrix is the region for showing the item values.

Example: Cluster Heatmap

The **heatmap** technique is one of the simplest uses of the matrix alignment: each cell is fully occupied by an area mark encoding a single quantitative value attribute with color. Heatmaps are often used with bioinformatics datasets; Figure **??** shows an example where the keys are genes and experimental conditions, and the quantitative value attribute is the activity level of a particular gene in a particular experimental condition as measured by a microarray. This heatmap uses diverging colormaps, as discussed in Section 8.3. The genes are a categorical attribute; experimental condition might be categorical or might be ordered, for example if the experiments were done at successive times.

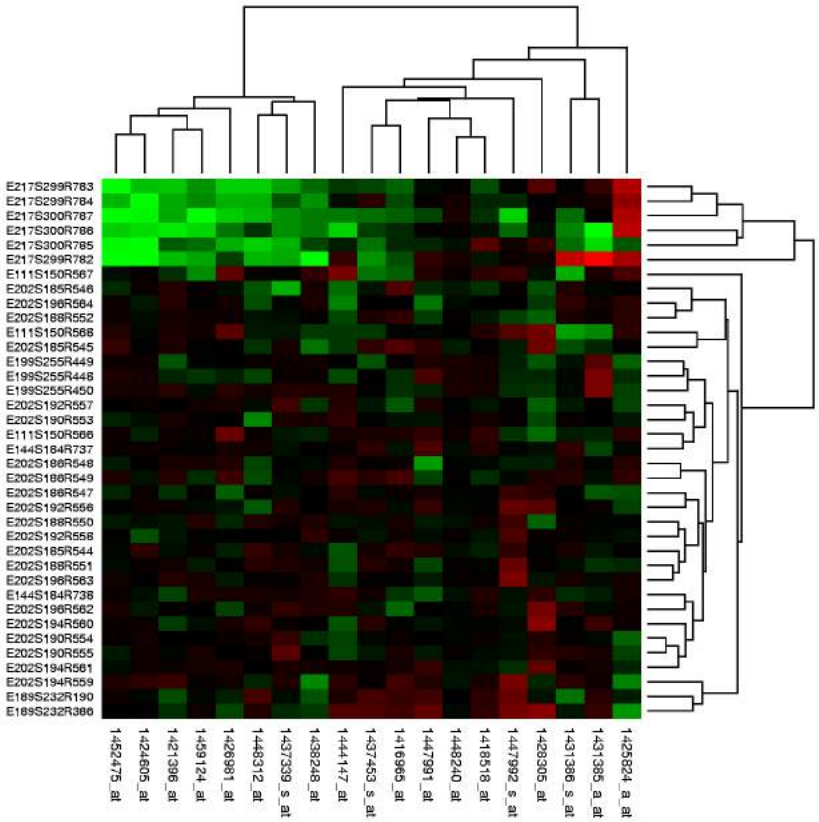


Figure 8.4: The heatmap shows a compact overview of a quantitative value attribute by visually encoding it with diverging colormap, with small area marks that are cells within a matrix of two key attributes. The cluster heatmap includes trees drawn on the periphery to show explicitly how the core heatmap matrix was reordered according to the derived data of hierarchical clusterings on its rows and columns.

The benefit of heatmaps is that visually encoding quantitative data with color using small area marks is very compact, so they are good for providing overviews with high information density. For simplicity, all of the scalability analysis in this book related to screen-space limits will assume a standard display size of 1000x1000, with a total of one million pixels. The area marks in a heatmap are often several pixels on a side for easy distinguishability, so

a matrix of 200x200 with 40,000 items is easily handled. The limit is area marks of a single pixel, for a dense heatmap showing one million items.

The cluster heatmap technique combines the basic heatmap with matrix reordering, where two attributes are reordered in combination. [1] The goal of matrix reordering is to group similar cells in order to check for large-scale patterns between both attributes, just as the goal of reordering a single attribute is to see trends across a single one. A **cluster heatmap** is the juxtaposed combination of a heatmap and two trees showing how the ordering of the cells in the heatmap matrix was obtained. The trees show the derived data of two hierarchical clusterings derived from the original table, one for the matrix rows and one for the columns.

Technique	**heatmap**
Data Types	table 2 categorical key attributes (genes, conditions) 1 quantitative value attribute (activity level for gene in condition)
View Comp.	space: area marks in 2D matrix alignment. color: diverging colormap.
Scalability	items: 1M attribs: 3
Technique	**cluster heatmap**
Derived Data	2 cluster hierarchies for table rows and columns
View Comp.	space: area marks in 2D matrix alignment, ordered by both cluster hierarchies. dendrogram: parent-child relationships in tree with connection line marks.

Example: Network Matrix Views

A 2D alignment is also used when visually encoding networks as **matrix views**, where all of the nodes in the network are laid out along the vertical and horizontal edges of a square region and links between two nodes are indicated by coloring an area mark in the cell in the matrix that is the intersection between their row and column. That is, the network is transformed into the derived dataset of a table with two key attributes that are separate full lists of every node in the network, and one value attribute for each cell

[1] Vocabulary note: Synonyms for matrix reordering are matrix permutation and seriation.

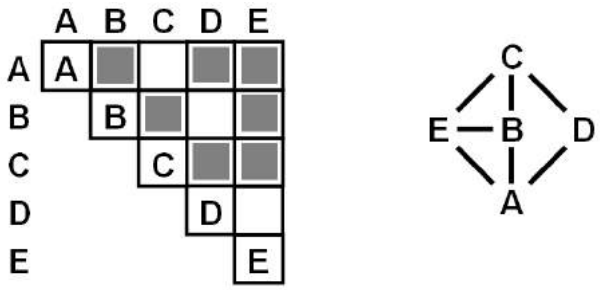


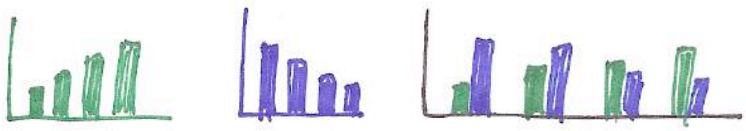
Figure 8.5: Comparing matrix and node-link views of a five-node network. a) Matrix view. b) Node-link view. From [Henry et al. 07], Figure 3b and 3a.

the records whether a link exists between those the nodes that index the cell. Network matrix views can also show weighted networks, where each link has an associated quantitative value attribute, by encoding with an ordered channel such as color luminance or size.

For undirected networks where links are symmetric, only half of the matrix needs to be shown, above or below the diagonal, because a link from node A to node B necessarily implies a link from B to A. For directed networks, the full square matrix has meaning, because links can be asymmetric. Figure 8.5 shows a simple example of an undirected network, with a matrix view of the five-node dataset in Figure 8.5a and a corresponding node-link view in Figure 8.5b.

Technique	**network matrix view**
Data Types	network
Derived Data	table: network nodes as keys, link status between two nodes as values
View Comp.	space: area marks in 2D matrix alignment

8.1.3.3 Multiple Keys: Partition and Subdivide When a dataset has only one key, then it is straightforward to use that key to separate into one region per item. When a dataset has multiple keys, there are several possibilities for separation. Given two keys, A and B, you could first separate by A and then by B, or you could first separate by B and then by A. A third option is that you might separate into regions by only one key, and then draw something more complex in the region. Regions are not limited to showing a single mark; they can be used for a glyph with internal structure,



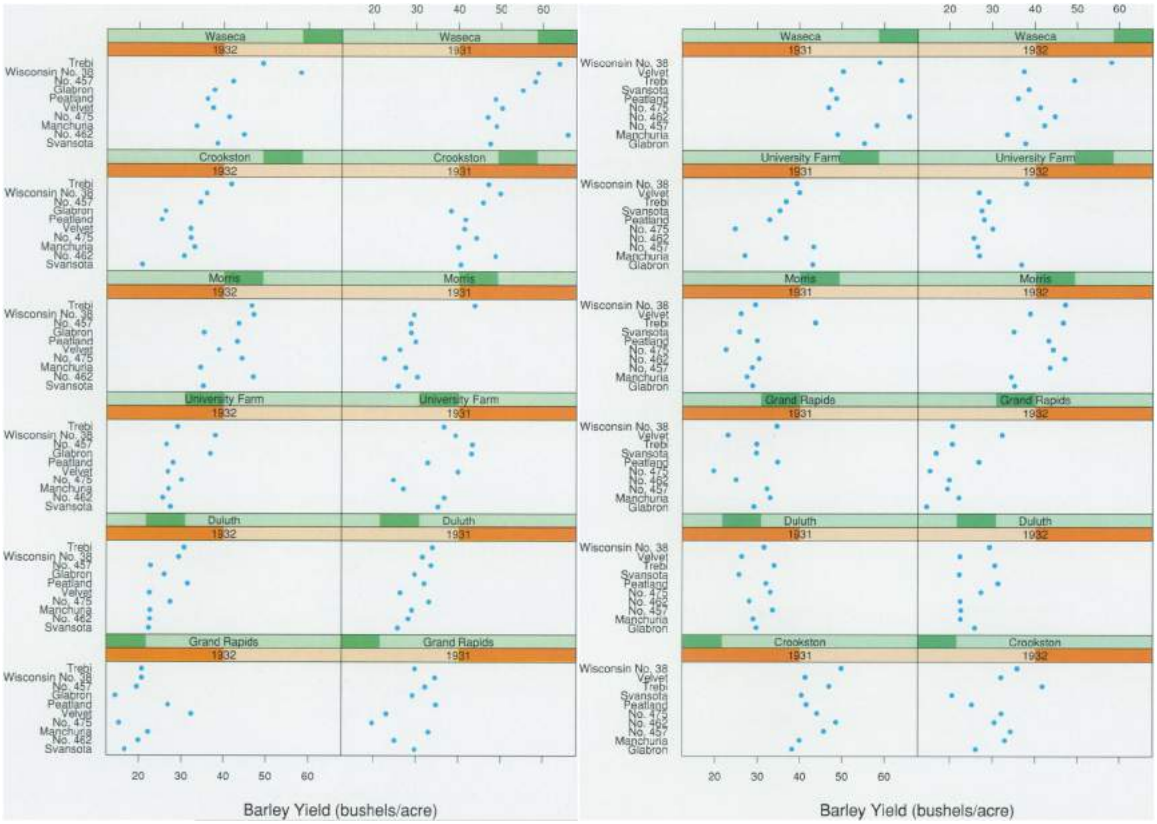
(a) (b)

Figure 8.6: Partitioning choices example with bar charts. (a) Two aligned bar charts: Separate by one key, A, into a horizontal list of two regions, with a bar chart in each. Within each chart, separate by another key, B, into regions with one mark each. (b) Single bar chart with grouped bars: separate by key B into regions with two marks each, within each region draw two marks.

or even an entire plot or chart. This question of how to **partition** a multi-attribute dataset into meaningful groups has major implications for what kind of patterns are visible and should be driven by the intended task.[2] The two design choices when partitioning are the attribute precedence in terms of which splits happen earlier versus later, and when to stop splitting and show the data as grouped marks rather than continuing to partition further.

Returning to bar charts as a simple first example, consider a dataset with two keys, `year` with 4 levels and `animal` with 2 levels, and one value attribute, `weight`. Figure 8.6 shows two different partitioning choices. In both, `animal` is color coded, with `cat` in green and `dog` in blue. In Figure 8.6a the dataset is first partitioned by `animal` into two regions and a separate chart is drawn in each region. Within each chart the second key `year` is used to separate the available space into one region for each item, where a single line mark is drawn. Partitioning first by `animal` groups the years most closely, making it easy to see the trend across years for each animal: the cats are gaining weight and the dogs are losing weight. In Figure 8.6b single bar chart is drawn where the only separation is by `year`. Each region contain a group of two marks rather than just a single one: a pair of bars that groups together both animals for that year. This choice makes it easy to compare the weights between the two animals at each year, but it is harder to see the trends for each animal.

[2]Vocabulary note: Partioning and grouping are inverse terms; the term *partitioning* is natural when considering starting from the top and gradually refining; the term *grouping* is more natural when considering a bottom-up process of gradually consolidating. Conditioning is a synonym for partitioning that is used heavily in the statistics literature.



(a) (b)

Figure 8.7: Trellis features partitioning and orderering. (a) With main-effects ordering, the plots are ordered by median values within the plots for the sites, and the shared vertical axis within each plot is ordered by median values within the varieties. The Morris site in the third row is a visible outlier that does not fit the general trends. (b) With a simple alphabetical ordering of plots and axes, no trends are visible so no outliers can be detected. From [Becker et al. 96], Figures 1 and 3.

Example: Dotplots and Trellis

In Trellis [Becker et al. 96], the methods of partitioning a multi-attribute dataset into individual plots and ordering them within a 2D matrix alignment are the main emphasis of the system. Figure 8.7 shows a dataset

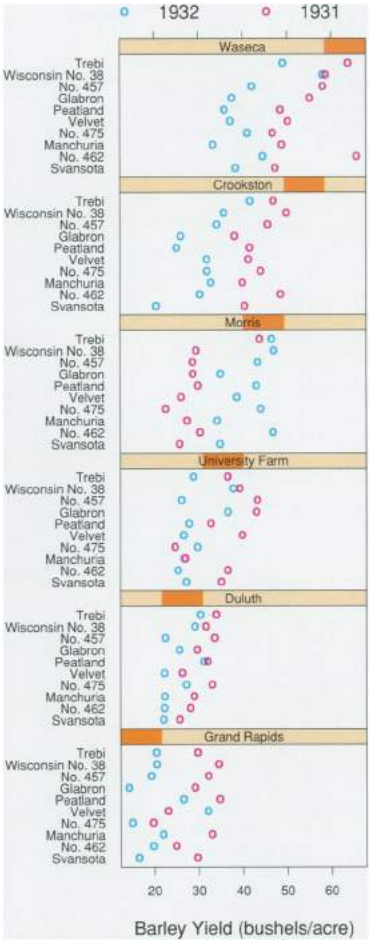


Figure 8.8: A second Trellis plot combines the years into a single plot with year encoded by color, showing strong evidence of an anomaly in the data. From [Becker et al. 96], Figure 2.

of barley yields shown with dotplots. A **dotplot** is a visual encoding of one quantitative attribute using spatial position against one categorical attribute, but using point marks rather than the line marks of a bar chart. This dataset is a multidimensional table with three categorical attributes that act as keys. The `site` attribute has six unique **levels**, the locations where the barley was grown. The `variety` attribute for the type of barley grown has ten levels. The `year` attribute has only two levels, and although it technically is a number it is treated as categorical rather than ordered.

The dataset also has a fourth quantitative attribute, the `yield`.

In this figure, the partitioning is by `year` for the matrix columns and by `site` for the rows. Within the individual dotplots the vertical axis is separated by `variety`, with `yield` as the quantitative value expressed with horizontal spatial position. The ordering technique used is **main-effects ordering**, where the derived attribute of the median value is computed for each group created by the partitioning and used to order them. In Trellis, main-effects ordering can be done at every partitioning scale. In Figure 8.7a the matrix rows are ordered by the medians for the site, and the rows within each dotplot are ordered by the medians for the varieties.

The value of main-effects ordering is that outliers countervailing to the general trends are visible. The Morris plots in the third row do not match up with the others, suggesting that perhaps the years had been switched. Figure 8.7b shows a trellis where the vertical ordering between and within the plots is alphabetical. This display does not provide any useful hints of outliers versus the trends, since no particular general trend is visible at all. Main-effects ordering is useful because it is a data-driven way to spatially order information, so that both general trends and outliers can be spotted.

Figure 8.8 shows another plot with a different structure to further investigate the anomaly. The plots are still partitioned vertically by `site`, but no further. Both years are thus included within the same plot, and distinguished from each other by color. The switch in color patterns in the third row shows convincing evidence for the theory that the Morris data is incorrect.

Dot plots use a point mark, and also use one planar position channel. Figure **??**(c) shows a dot plot of cat weight over time with the ordered variable of year on the horizontal axis, and the quantitative weight of a specific cat on the vertical axis.

Technique	**Dotplot**
Data Types	table 1 quantitative value attrib, 1 key attrib
View Comp.	point mark. space: aligned position to express value attrib, position to separate/order key attrib
System	**Trellis**
Data Types	multidimensional table: 3 categorical key attributes 1 quantitative value attribute
Derived Data	medians for each partition
View Comp.	space: partitioned by any combination of keys into regions in 2D matrix alignment, dotplot within each region.

8.1.4 Value Attributes

The methods introduced in the previous section are not just for key attributes; they can all be used for value attributes as well. An attribute can be categorical without being a key; that attribute can still be used to separate into regions and partition the dataset according to the levels for that attribute.

When dealing with key attributes, it is possible to partition the data down to to the item level, since each item is uniquely indexed by the combination of all keys. With a value attribute, multiple items can all share the same value, so the final division might be a group of items rather than just a single one. A partitioning attribute is typically a categorical variable that has only a limited number of unique values. It can also be a derived attribute, for example created by a transformation from a quantitative attribute by dividing it up into a limited number of bins.

Partitioning is an action on a dataset; the question of how to separate the resulting groups into different regions of space remains. The previous section discussed the simple cases of a list and a matrix. In general, space can be subdivided into regions, and a recursive subdivision allows these regions to nest inside each other. The regions created by the subdivision can also be ordered and aligned, just as in the simpler cases.

Partitioning is powerful and general idea, especially when used hierarchically with a succession of attributes to slice and dice the dataset up into pieces of possible interest. This choice of which attributes to partition by

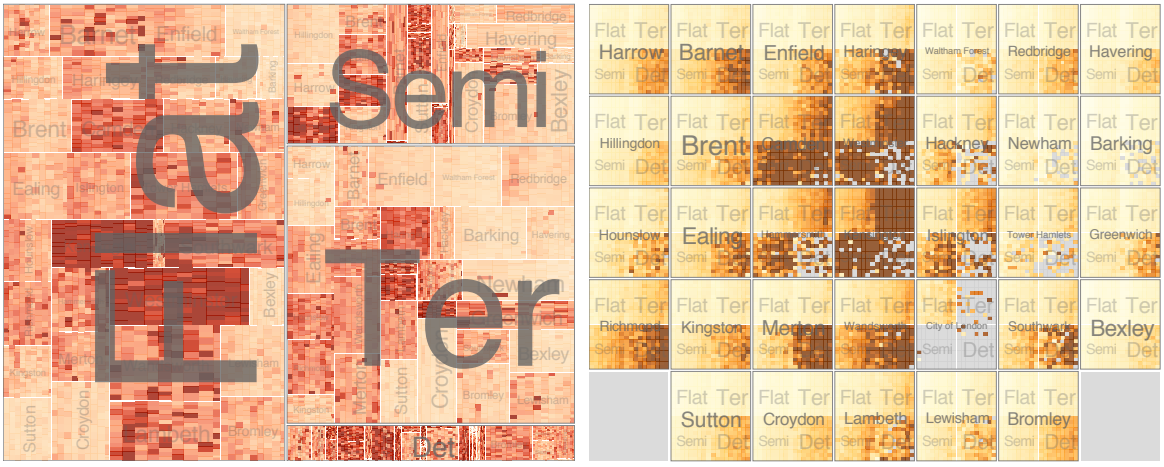


Figure 8.9: This dataset of London property transactions can be conditioned into small multiples in many different ways. (a) The first split into subsets is by house type. (b) The first split is into neighborhoods. From [Slingsby et al. 09], Figures 7a and 2c.

versus which to directly encode by, and the order of partitioning, has a profound effect on what aspects of the dataset are visually salient. [3]

Example: HIVE

Partitioning can be used in an exploratory way, where the user can reconfigure the display to see different choices of partioning and encoding variables. Figure 8.9(a) shows a dataset of property transactions in the London area, with attributes of residence type (flats, attached terrace houses, semi-detached houses, and fully detached houses), house price, neighborhood, and time of sale in terms of year and month. The dataset is first partitioned by residence type into rectangular views, which are collected together into a large rectangular region. Within each view, it is further partitioned by geographic location into neighborhoods, and finally by time of sale. The visual encoding for each view is a treemap, which can be considered as a collection of spatially contiguous rectangle marks that can be color coded, and the rectangles can be further subdivided recursively to show hierarchical structure with containment. The color coding is by price variation within the group. This example shows that the patterns

Treemaps are discussed in detail in Section 8.2.2.

[3] Vocabulary note: Hierarchical partitioning is also known as **dimensional stacking**

of the top-level rectangles, which are the small-multiple views for the four different house types, are very different. In contrast, the coloring within each second-level rectangle representing a neighborhood is more consistent; that is, houses within the same neighborhood tend to have similar prices.

Figure 8.9(b) shows another configuration of the same dataset. This example is partitioned first by neighborhood, then by residence type, then by time of sale in terms of year, and finally by month. The color coding is by average price within the group. In this display it is easy to spot expensive neighborhoods, which are the views near the center. It is also easy to see that detached houses, in the lower right corner of each view, are more expensive than the other types.

8.1.5 Given: Use

For spatial fields, datasets with spatial semantics, the usual choice is to use the given spatial layout. In this case, some of the design choices discussed in the previous section do not apply, because the position channel cannot be used to encode attributes, so other channels must be used. The common case is that spatial position is the attribute of primary importance because the central tasks revolve around understanding spatial relationships. In these cases, the right visual encoding choice is to use the provided spatial position as the substrate for the visual layout, rather than to visually encode other attributes with marks using the spatial position channel. This choice may seem obvious from common sense alone. It also follows from the effectiveness principle discussed in Section 5.2.6, since the most effective channel of spatial position is used to show the most important aspect of the data, namely the spatial relationships between elements in the dataset. Of course, it is possible that datasets with spatial attribute semantics might not have the task involving understanding of spatial relationships as the primary concern. In these cases, the question of which other attributes to encode with spatial position is once again on the table.

With spatial fields, the spatial attributes are usually called **dimensions** and act as keys for a multidimensional table. The standard set of methods are intended for the **2D** case of two spatial attributes and the **3D** case of three spatial attributes. The taxonomy of these methods is divided according to the number of value attributes for each item: scalar fields have one, vector fields have two, and tensor fields have three or more.

Multivariate spatial field terminology is also discussed in Section 4.4.1.

8.1.5.1 1 Value: Scalar Fields

Example: Isosurfaces

TODO

Example: Direct Volume Rendering

TODO

8.1.5.2 2 Values: Vector Fields

Example: Arrows

TODO

Example: Streamlines

TODO

8.1.5.3 3+ Values: Tensor Fields

Example: Ellipsoids

TODO

Example: Superquadric Glyphs

TODO

8.1.5.4 Geographic Data

Example: Choropleth Maps

TODO

8.1.6 Spatial Layout

Another choice with the use of space is whether to use rectilinear, parallel, or radial layout. Any of these layouts can also be spacefilling and/or can be dense.

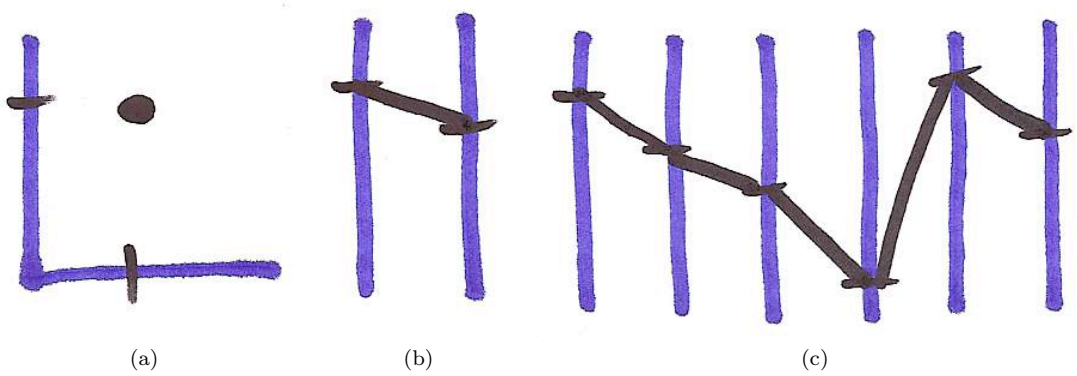
8.1.6.1 Rectilinear Layouts In a **rectilinear** layout, regions or items are distributed along two perpendicular axes, horizontal and vertical spatial position, that range from minimum value on one side of the axis to a maximum value on the other side. Rectilinear layouts are heavily used in visualization design, and occur in many common statistical charts. All of the examples above use rectilinear layouts.

8.1.6.2 Parallel Layouts The rectilinear approach of a scatterplot, where items are plotted as dots with respect to perpendicular axes, is only usable for two data attributes when high-precision planar spatial position is used. Even if the low-precision visual channel of a third spatial dimension is used, despite the potential drawbacks discussed in Section 7.1, then only three data attributes can be shown using spatial position channels. Although additional channels can be used for visual encoding, as discussed in Section 8.1, there are major limits on the number of channels that can combined effectively in a single view. Of course, many tables contain far more than three quantitative attributes.

The technique of **parallel coordinates** is an approach for visualizing many quantitative attributes at once using spatial position. As the name suggests, the axes are placed parallel to each other, rather than perpendicularly at right angles. While an item is shown with a dot in a scatterplot, with parallel coordinates a single item is represented by a jagged line that zigzags through the parallel axes, crossing each axis exactly once at the location of the item's value at for the associated attribute. [4] Figure 8.10 shows an example: a single item is shown with a black dot in the scatterplot in Figure **??** the left, but with a black line in the parallel coordinates view on the middle. The thin purple lines mark the location of the item values on each axis. The black parallel coordinate line passes through these locations, in contrast to the scatterplot dot that does not touch them.

One original claim by the designers of parallel coordinates was that they can be used for the task of checking for correlation between attributes. In scatterplots, the visual pattern showing correlation is the tightness of the diagonal line formed by the item dots, tilting upwards for positive correlation and downwards for negative correlation. If the attributes are not correlated, the points fall throughout the two-dimensional region rather than tightly along the diagonal. With parallel coordinates, correlation is also visible, but through different kinds of visual patterns, as illustrated in Figure 8.11. If two neighboring axes have high positive correlation, the line segments are mostly parallel. If two axes have high negative correlation, the line segments mostly cross over each other at a single spot between the axes. The pattern in between uncorrelated axes is a mix of crossing angles.

[4] Vocabulary note: In graphics terminology, it is a *polyline*: a connected set of straight line segments.



(a) (b) (c)

Figure 8.10: Parallel coordinates, simple example. a) In a scatterplot, an item with two attributes is represented by a dot. b) With parallel coordinates, an item with two attributes is represented by a line segment. c) Parallel coordinates allow an item with multiple attributes to be represented with a jagged line across multiple parallel axes.

However, in practice, parallel coordinates are more often used for other tasks including outlier detection, selection, and general overview. Scatterplots are easier to use for the finding correlation task.

Parallel coordinates visually encode data using two dimensions of spatial position. Of course, any individual axis requires only one spatial dimension, but the second dimension is used to lay out multiple axes. The scalability is high in terms of the number of quantatitive attribute values that can be discriminated, since the high-precision channel of planar spatial position is used. The exact number is roughly proportional to the screen space extent of the axes, in pixels. The scalability is moderate in terms of number of attributes that can be displayed: dozens is common. As the number of attributes shown increases, so does the width required to display them, so a parallel coordinates display showing many attributes is typically a wide and flat rectangle. Assuming that the axes are vertical, then the amount of vertical screen space required to distinguish position along them does not change, but the amount of horizontal screen space increases as more axes are added. One limit is that there must be enough room between the axes to discern the patterns of intersection or parallelism of the line segments that pass between them.

The basic parallel coordinates techniques scales to showing hundreds of items, but not thousands. If too many lines are overplotted, the resulting occlusion yields very little information. Figure 8.12 contrasts the technique used successfully with 13 items and 7 attributes, versus ineffectively with

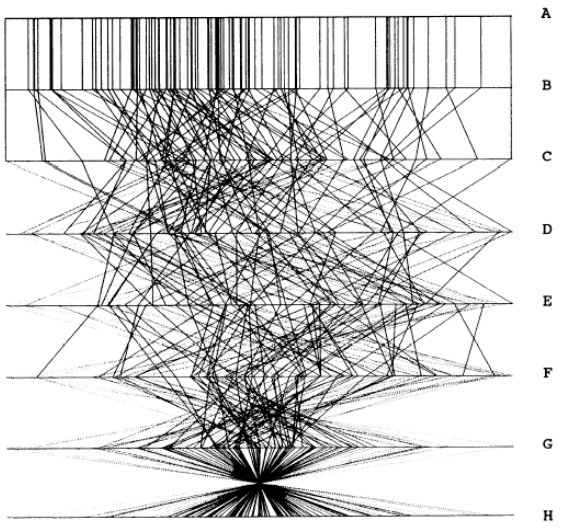


Figure 3. Parallel Coordinate Plot of Six-Dimensional Data Illustrating Correlations of $\rho = 1, .8, .2, 0, -.2, -.8,$ and -1.

Figure 8.11: Parallel coordinates were designed to show correlation between neighboring axes. At the top, parallel lines show perfect positive correlation. At the bottom, all of the lines cross over each other at a single spot in between the two axes, showing perfect negative correlation. At the middle, the mix of crossings shows uncorrelated data. From [Wegman 90], Figure 3.

over 16,000 items and 5 attributes. In the latter case, only the minimum and maximum values along each axis can be read; it is impossible to see trends, anomalies, or correlations.

The patterns made easily visible by parallel coordinates have to do with the pairwise relationships between neighboring axes. The crucial limitation of parallel coordinates is thus how to determine the order of the axes. Most implementations allow the user to interactively reorder the axes. However, exploring all possible configurations of axes through systematic manual interaction would be prohibitively time-consuming as the number of axes grows, because of the exploding number of possibilities.

Another limitation of parallel coordinates is training time; first-time users do not have intuitions about the meaning of the patterns they see, which must thus be taught explicitly. Parallel coordinates are often used one of several multiple views showing different visual encodings of the same dataset, as discussed in Chapter **??**, rather than as the only encoding. The

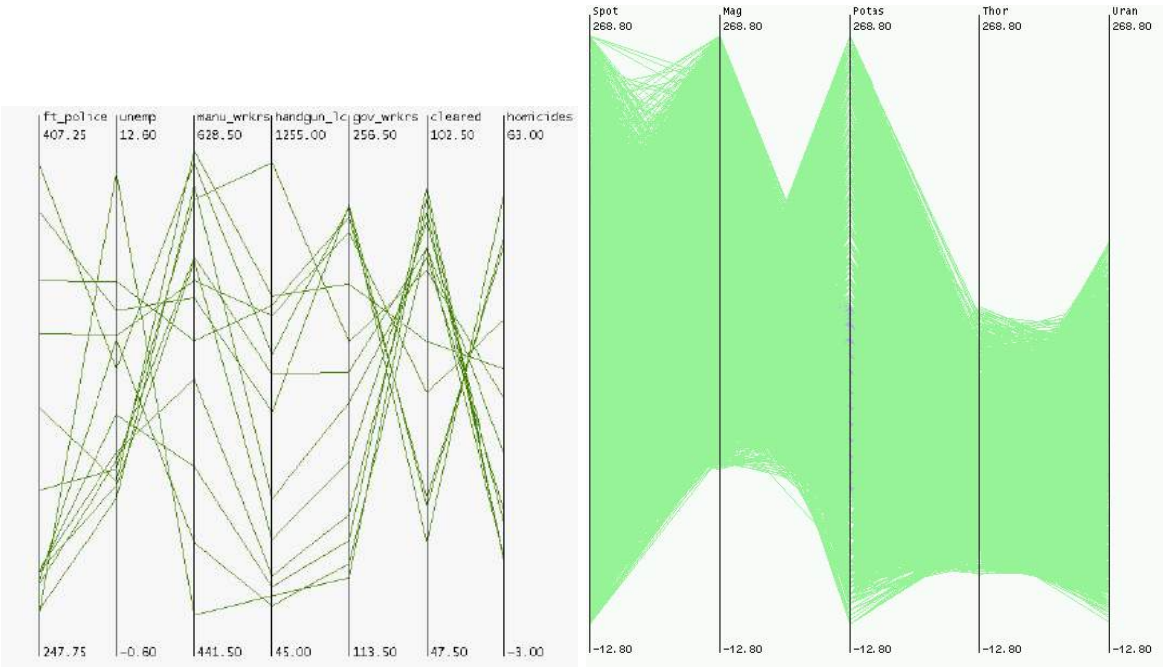


Figure 8.12: Parallel coordinates scale to dozens of attributes and hundreds of items, but not to thousands of items. a) Effective use with 13 items and 7 attributes. b) Ineffective use with over 16,000 items and 5 attributes. From [Fua et al. 99], Figures 1 and 2.

combination of more familiar views such as scatterplots with a parallel coordinates view accelerates learning, particularly since linked highlighting reinforces the mapping between the dots in the scatterplots and the jagged lines in the parallel coordinates view.

Technique	**parallel coordinates**
Data Types	table: k dependent attribs
View Comp.	parallel, 2 position channels: positions along aligned axes connected with lines position for separating axes
Abstract Tasks	find trends, outliers, extremes, correlation
Scalability	attributes: dozens along secondary axis items: hundreds

(a) (b) (c)

Figure 8.13: Radial vs. rectilinear layouts. a) Rectilinear bar chart. b) Radial star plot. c) Radial multipodes plot. From [Booshehrian et al. 11], Figure 4.

Figure 8.14: Radial layouts. a) Radial layouts use polar coordinates, with one spatial position and one angle channel. b) Rectlinear layouts use two perpendicular spatial position channels. c) Transforming rectilinear to radial maps two parallel bounding lines to a point at the center and a circle at the perimeter.

8.1.6.3 Radial Layouts: Position and Angle In a **radial** spatial layout, items are distributed around a circle using the angle channel in addition to one or more linear spatial channels, in contrast to the rectilinear layouts that use only two spatial channels.

Figure 8.13 shows the same five-attribute dataset encoded with a rectilinear bar chart and two radial alternatives, a star plot and a multipodes plot. In a star plot, the one-pixel wide line marks all emerge from a center point, with the attribute encoded as the spatial position along each axis. In the multipodes plot, line marks are distributed along a center circle, making short lines easier to distinguish at the expense of requiring more screen space. Simiarly, the lines are also wider, providing more room for color coding the marks with an additional attribute.

The star and multipodes plots show that a layout can be both radial and parallel; a star plot is the radial version of a parallel coordinate plot. In contrast, a radial plot cannot be rectilinear.

Technique	**star plots**
Data Types	table
View Comp.	length coding along point marks at 1D spatial position along axis + 1D spatial position for aligned axes

Mathematically, the natural coordinate system in radial layouts is **polar coordinates**, where one dimension is measured as an angle from a starting line and the other is measured as a distance from a center point, as shown

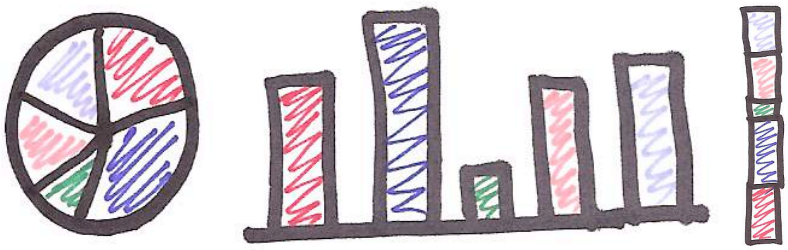
in Figure **??**. From a strictly mathematical point of view, rectilinear and radial layouts are equivalent under a particular kind of transformation: a box bounded by two sets of parallel lines is transformed into a disc with where one line is collapsed to a point at the center and the other line wraps around to meet up with itself, as in Figure **??**.

However, from a perceptual point of view, they are not equivalent at all. The change of visual channel has two major consequences from visual encoding principles alone. First, the angle channel is less accurately perceived than a rectilinear spatial position channel. Second, the nonmonotonic angle channel is inherently cyclic, because the start and end point are the same, as opposed to the inherently linear nature of a position channel. The expressiveness and effectiveness principles suggest some guidelines on the use of radial layouts. Radial layouts may be more effective than rectilinear ones in showing the periodicity of patterns, but encoding nonperiodic data with the periodic channel of angle may be misleading. Radial layouts imply an asymmetry of importance between the two attributes, and would be inappropriate when the two attributes have equal importance.

A first empirical study on radial versus rectilinear grid layouts by Diehl et al. focused on the abstract task of memorizing positions of objects for a few seconds [Diehl et al. 10]. They compared performance in terms of accuracy and speed for rectilinear grids of rows and columns versus radial grids of sectors and rows. (The study did not investigate the effect of periodicity.) In general, rectilinear layouts outperformed radial layouts: perception speed was faster, and accuracy tended to be better. However, their results also suggest the use of radial layouts can be justified when one attribute is more important than the other. In this case, the more important attribute should be encoded in the sectors and the less important attribute in the rings.

The most commonly used radial statistical graphic is the pie chart, shown in Figure **??**. Pie charts encode a single attribute with area marks and the angle channel. Despite their popularity, pie charts are clearly problematic when considered according to the visual channel properties discussed in Section 5.3. Angle judgements on area marks are less accurate than length judgements on line marks. The wedges vary in width along the radial axis, from narrow near the center to wide near the outside, making the area judgement is particularly difficult. Figure **??** shows a bar chart with the same data, where the perceptual judgement required to read the data is the high-accuracy position along a common scale channel.

A critical property of pie charts is that they show the relative contribution of parts to a whole. The sum of the wedge angles must add up to the 360° of a full circle, matching normalized data such as percentages where the parts must add up to 100%. However, this property is not unique to pie charts; a single stacked bar can also be used to show this property with



(a) (b) (c)

Figure 8.15: Pie chart vs bar chart accuracy. a) Pie charts require angle and area judgements. b) Bar charts require only high-accuracy length judgements for individual items, but contributions of parts to the whole are difficult to see. c) A single stacked bar can show relative contributions of parts to a whole.

the more accurate channel of length judgements, as in Figure **??**.

Stacking is discussed further in Section **??**.

8.1.7 Spacefilling

A **spacefilling** layout has the property that it fills all available space in the view, as the name implies. Any of the three geometric possibilties discussed above can be spacefilling. Spacefilling layouts typically use area marks for items or containment marks for relationships, rather than line or connection marks, or point marks. Examples of spacefilling layouts using containment marks are the treemaps in Figure 8.25g and the nested circle tree in Figure 8.25f. Examples of spacefilling layouts using area marks and the spatial position channels are the concentric circle trees in Figure 8.25e and the icicle tree of Figure 8.25c.

One advantage of space-filling approaches is that they maximize the amount of room available for color coding, increasing the chance that the colored region will be large enough to be perceptually salient to the viewer. A related advantage is that the available space representing an item is often large enough to show a label embedded within it, rather than needing more room off to the side.

In contrast, one disadvantage of spacefilling views is that the designer cannot make use of **white space** in the layout; that is, empty space where there are no explicit visual elements. Many graphic design guidelines pertain to the careful use of white space for many reasons including readability, emphasis, relative importance, and visual balance.

Spacefilling layouts are typically aimed at the goal of achieving high information density. However, the property that a layout is spacefilling is

by no means a guarantee that is using space efficiently. More technically, the definition of spacefilling is that the total area used by the layout is equal to the total area available in the view. There are many other possible metrics for analyzing the space efficiency of a layout. For instance, for trees, proposed metrics include the size of the smallest nodes and the area of labels on the nodes [McGuffin and Robert 10].

8.1.8 Dense

A **dense** layout uses small and densely packed marks to provide an overview of as many items as possible with very high information density.[5] Point marks are only a single pixel in size, and line marks are only a single pixel wide. The small size of the marks implies that only the planar position and color channels can be used in visual encoding; size and shape are not available, nor are others like tilt, curvature, or stipple that require more room than is available. Section 11.1.1 provides a detailed discussion of VisDB, a dense display for multidimensional tables using point marks.

Dense displays using line marks have become popular for showing overviews of software source code. In these displays, the arrangement of the marks is dictated by the order and length of the lines of code, and the coloring of the lines encodes an attribute of interest.

Example: Tarantula

Figure 8.16 shows the Tarantula system, a software engineering tool for visualizing test coverage [Eagan et al. 01,Jones et al. 02]. Most of the screen is devoted to a large pixel-oriented view providing overview of source code using one-pixel wide lines, color coded to show whether it passed, failed, or had mixed results when executing a suite of test cases. Although of course the code itself cannot be read from the low-resolution overview, this view does convey some information about the code structure. Indentation and line length are preserved, creating visible landmarks that help orient the reader. The layout wraps around to create multiple horizontal columns out of a single long linear list. The small source code view in the lower left corner is a detail view showing a few lines of source code at a legible size; that is, a high-resolution view with the same color coding and same spatial position. The pixel-oriented display scales to around 10K lines of code, handling around 1K vertical pixels and ten columns.

The dataset used by Tarantula is an interesting complex combination of the software source code, the test results, and derived data. The original dataset is the software source code itself, which can be considered as a

[5] Vocabulary note: A synonym for *dense* is *pixel-oriented*.

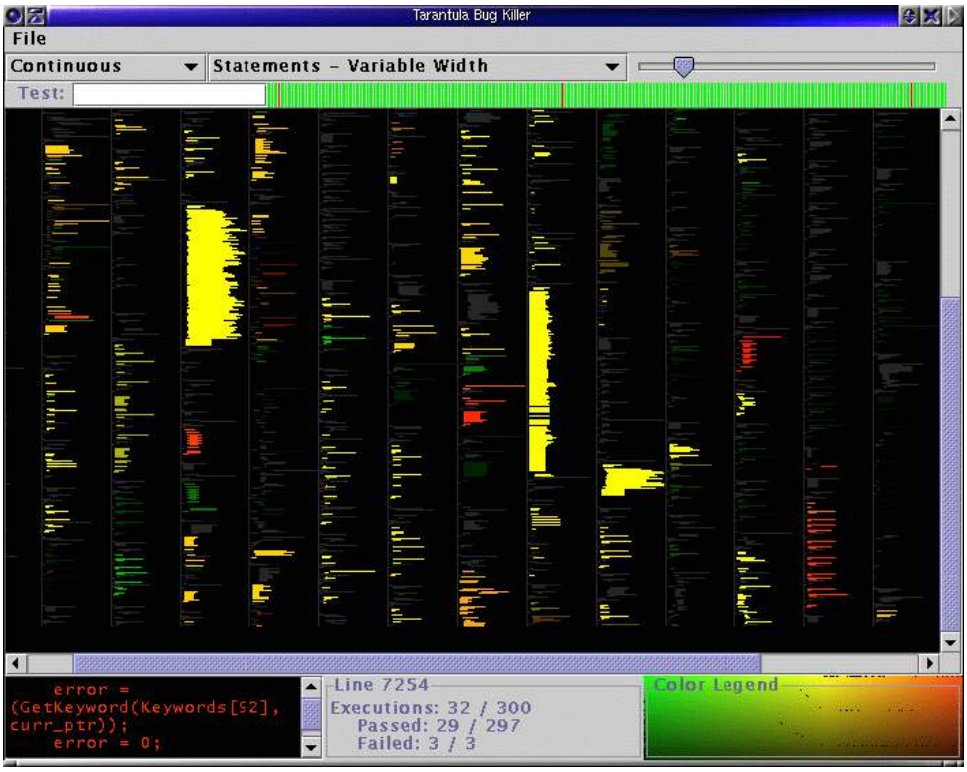


Figure 8.16: Tarantula shows a pixel-oriented overview of source code with lines color coded by execution status of a software test suite. Courtesy of John Stasko, from [Jones et al. 02] Figure 4.

special case of the text data type that is divided into numbered lines. The original dataset also includes the tests, where each test has the categorical attribute of test or fail, and is associated with a set of specific lines of the source code. Tarantula computes two derived quantitative attributes that are encoded with hue and brightness. The brightness encodes the percentage of coverage by the test cases, where dark lines represent low coverage and bright ones are high coverage. The hue encodes the relative percentage of passed versus failed tests.

System	**Tarantula**
Data Types	text (source code, test results log)
Derived Data	2 quantitative attributes (test execution results)
Multiple Views	same encoding, same dataset, global overview with detail showing subset of data, different resolutions, linking with color
View Comp.	pixel-oriented spatial position and line length from text ordering color channels of hue and brightness
Reduction	overview: none detail: filter to local neighborhood of selection
Domain Task	locating faults in source code, overview of testing results and coverage
Scalability	lines of text: 10K

8.2 Link Marks

The use of link marks is an alternative to directly encoding information with spatial position. The two main methods are using line marks for connection, and area marks for containment.

8.2.1 Connection

The most common visual encoding method for tree and network data is with **node-link diagrams**, where nodes are drawn as point marks and the links connecting them are drawn as line marks. This method uses connection marks to indicate the relationships between items. Figure 8.17 shows three examples of trees laid out as node-link diagrams. Figure 8.17a shows a tiny tree of 24 nodes laid out with a triangular vertical node-link layout, with the root on the top and the leaves on the bottom. In addition to the connection marks, it uses vertical spatial position channel to show the depth in the tree. The horizontal spatial position of a node does not directly encode any attributes. It an artifact of the layout algorithm's calculations to ensure maximum possible information density while guaranteeing that there are no edge crossings or node overlaps [Buchheim et al. 02].

Figure 8.17b shows a small tree of a few hundred nodes laid out with a spline radial layout. This layout uses essentially the same algorithm for density without overlap, but the visual encoding is radial rather than rectilinear: the depth of the tree is encoded as distance away from the

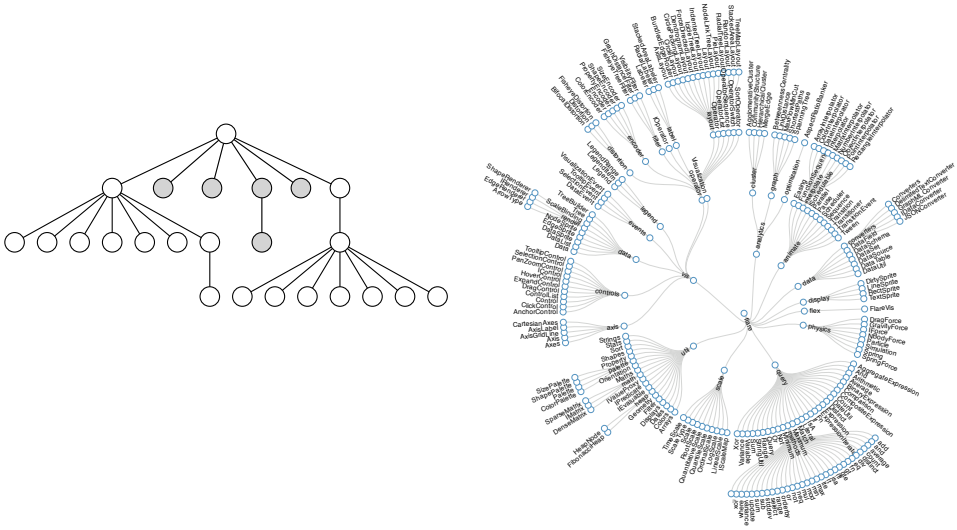


Figure 8.17: Node-link layouts of small trees. a) Triangular vertical for tiny tree. From [Buchheim et al. 02], Fig. 2d. b) Spline radial layout for small tree. Courtesy of Michael Bostock, made with D3 [Bostock et al. 11], from http://mbostock.github.com/d3/ex/tree.html.

center of the circle. Also, the links of the graph are drawn as smoothly curving **splines** rather than as straight lines.

Figure 8.18a shows a large tree of 3238 nodes laid out as a rectangular horizontal node-link diagram, with the root on the left and the leaves stretching out to the right. The edges are colored with a blue to yellow continuous colormap according to the Strahler centrality metric discussed in Section 4.6.2.1. The spatial layout is fundamentally the same as the triangular one, but from this zoomed-out position the edges within a subtree form a single perceptual block where the spacing in between them cannot be seen. Figure 8.18b shows the same tree laid out with the BubbleTree algorithm [Grivet et al. 06]. BubbleTree is also a radial rather than rectilinear approach, but subtrees are laid out in full circles rather than partial circular arcs. Spatial position does encode information about tree depth, but as relative distances to the center of the parent rather than as absolute distances in screen space.

Figure 8.25 shows many different visual encodings of the same tree

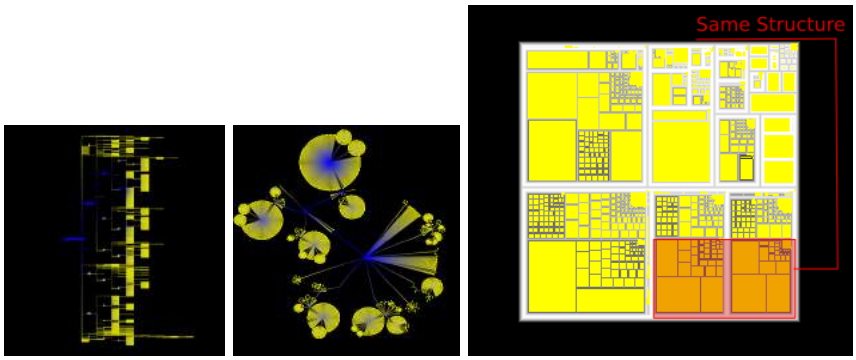


Figure 8.18: Three layouts of a larger tree of 3238 nodes. a) Rectangular horizontal node-link layout for large tree. b) Bubble-Tree node-link layout for same tree. Treemaps layout, showing hierarchical structure with containment rather than connection. Courtesy of David Auber, made with Tulip [Auber et al. 12], from http://tulip.labri.fr/Documentation/3_7/userHandbook/html/ch06.html.

dataset. Three of them use connection: the vertical rectilinear layout of Figure 8.25a, the horizontal rectilinear layout of Figure 8.25b, and the radial layout of Figure 8.25c. There are many alternatives beyond connection for showing hierarchical structure, as discussed in later sections.

Networks are also very commonly represented as node-link diagrams, using connection. Nodes that are directly connected by a single link are perceived as having the tightest grouping, while nodes with a long path of multiple hops between them are less closely grouped. The number of **hops** within a path – the number of individual links that must be traversed to get from one node to another – is a network-oriented way to measure distances. Whereas distance in the 2D plane is a continuous quantity, the network-oriented distance measure of hops is a discrete quantity. The connection marks support path tracing via these discrete hops.

Node-link diagrams in general are well suited for tasks that involve understanding the network **topology**: the direct and indirect connections between nodes in terms of the number of hops between them through the set of links. Examples of topology tasks include finding all possible paths from one node to another, finding the shortest path between two nodes, finding all the adjacent nodes one hop away from a target node, and finding nodes that act as a bridge between two components of the network that would otherwise be disconnected.

Node-link network layout techniques typically do not directly use spatial position to encode attribute values. The algorithms are designed to

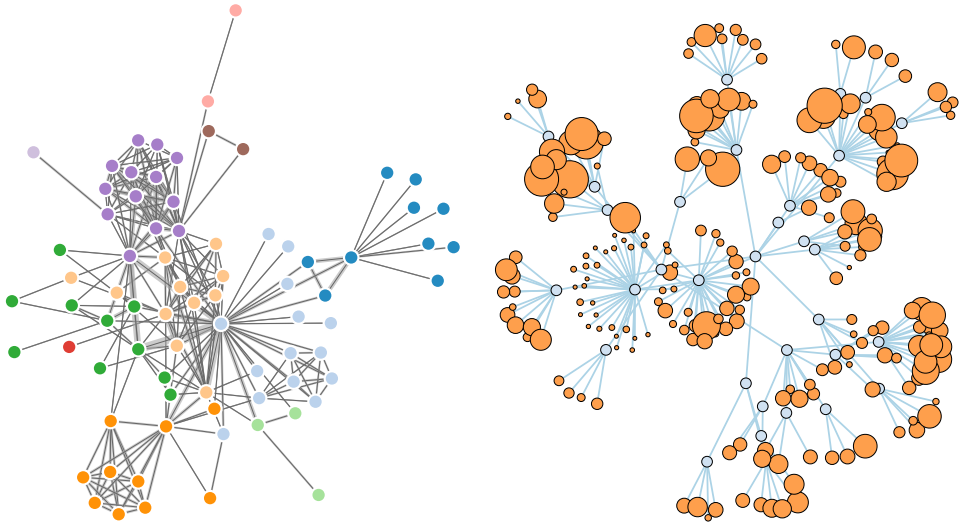


Figure 8.19: Node-link layouts of small networks. a) Force-directed placement of small network of 75 nodes, with size coding for link attributes. b) Larger network, with size coding for node attributes. Courtesy of Michael Bostock, made with D3 [Bostock et al. 11], from http://mbostock.github.com/d3/ex/force.html and http://bl.ocks.org/1062288.

minimize the number of distracting artifacts such as edge crossings and node overlaps, so the spatial location of the elements is a side effect of the computation rather than directly encoding attributes. Figure 8.19a shows a node-link layout of a graph, using the technique of force-directed placement discussed in more detail in Section 8.2.1. Size and color coding for nodes and edges is also common. Figure 8.19a shows size coding of edge attributes with different line widths, and Figure 8.19b shows size coding for node attributes through different point sizes.

Node-link diagrams are most often laid out within a two-dimensional planar region. While it is algorithmically straightforward to design 3D layout algorithms, it is rarely an effective choice because of the many perceptual problems discussed in Section 7.1, and thus should be carefully justified.

Example: Force-Directed Placement

One of the most widely used approaches for node-link network layout using connection marks is **force-directed placement**, where the network elements are positioned according to a simulation of physical forces where nodes push away from each other while links act like springs that draw their endpoint nodes closer to each other.[6] Force-directed placement algorithms start by placing nodes randomly within a spatial region, and then iteratively refining their location according to the pushing and pulling of these simulated forces to gradually improve the layout. One strength of this approach is that a simple version is very easy to implement. Another strength is that it is relatively easy to understand and explain at a conceptual level, using the analogy of physical springs.

Technique	**force-directed placement**
Data Types	network
View Comp.	connection
Abstract Tasks	understanding topological structure following paths
Scalability	nodes: dozens/hundreds edges: hundreds node/edge density: $E < 4N$

Analyzing the visual encoding created by force-directed placement is somewhat subtle. Spatial position does not directly encode any attributes of either nodes or links; the placement algorithm uses it indirectly. A tightly interconnected group of nodes with many links between them will often tend to form a visual clump, so spatial proximity does indicate grouping through a strong perceptual cue. However, some visual clumps are simply artifacts: nodes have been pushed near each other because they were repelled from elsewhere, not because they are closely connected in the network. Thus, proximity is sometimes meaningful but sometimes arbitrary; this ambiguity can lead to misleading interpretations by the user. This situation is a specific instance of the general problem that occurs in all techniques where spatial position is implicitly chosen rather than deliberately used to encode information.

As with any kind of computational optimization, the algorithm's search can get stuck in **local minimum** energy configuration that is not the globally best answer. The layout is thus **nondeterministic** because it will look different each time the algorithm is run, rather than a **deterministic** approach such as a scatterplot or a bar chart that yields an identical layout each time for a specific dataset. Most methods that use randomness share

[6] Vocabulary note: Force-directed placement is also known as spring embedding, energy minimization, or nonlinear optimization.

Figure 8.20: Force-directed placement simulates repelling forces between nodes with attracting forces along the edges, like springs. a) Small graphs can be laid out quickly and fairly readably. b) Large graphs typically degenerate into "hairballs" once the number of nodes reaches hundreds or thousands.

this weakness. The problem with nondeterministic visual encodings is that spatial memory cannot be exploited across different runs of the algorithm. The identity of "the stuff in the upper left corner" changes each time. Moreover, the randomness can lead to different proximity relationships each time, where the distances between the nodes reflect the randomly chosen initial positions rather than the intrinsic structure of the network in terms of how the links connect the nodes.

A major weakness of force-directed placement is scalability, both in terms of the visual complexity of the layout and the time required to compute it. Force-directed approaches yield readable layouts quickly for tiny graphs with dozens of nodes. Figure 8.20 shows an example. However, the layout quickly degenerates into a "hairball" with even a few hundred nodes, where the tasks of path following or understanding overall structural relationships become very difficult, and essentially impossible with thousands of nodes or more. Moreover, force-directed placement algorithms are notoriously brittle: they have many parameters that can be tweaked to improve the layout for a particular dataset, but different settings are required to do well for another.

In the simplest force-directed algorithms, the nodes never settle down to a final location; they continue to bounce around if the user does not explicitly intervene to halt the layout process. While seeing the force-directed layout iteratively improve the layout can be interesting while the layout is actively improving, continual bouncing can be distracting and should be avoided if a force-directed layout is being used in a multiple-view context where the user may want to attend to other views without having motion-sensitive peripheral vision invoked. More sophisticated algorithms automatically stop by determining that the layout has reached a good balance between the forces.

Many recent approaches to scalable network drawing are **multi-level** methods, where the original network is transformed by **coarsening** it into a derived hierarchy of successively simpler networks that nevertheless attempt to capture the most essential aspects of the original's structure. The combination of the original network with the derived hierarchy of coarsened networks is a compound graph, as defined in Section 8.6. By laying out

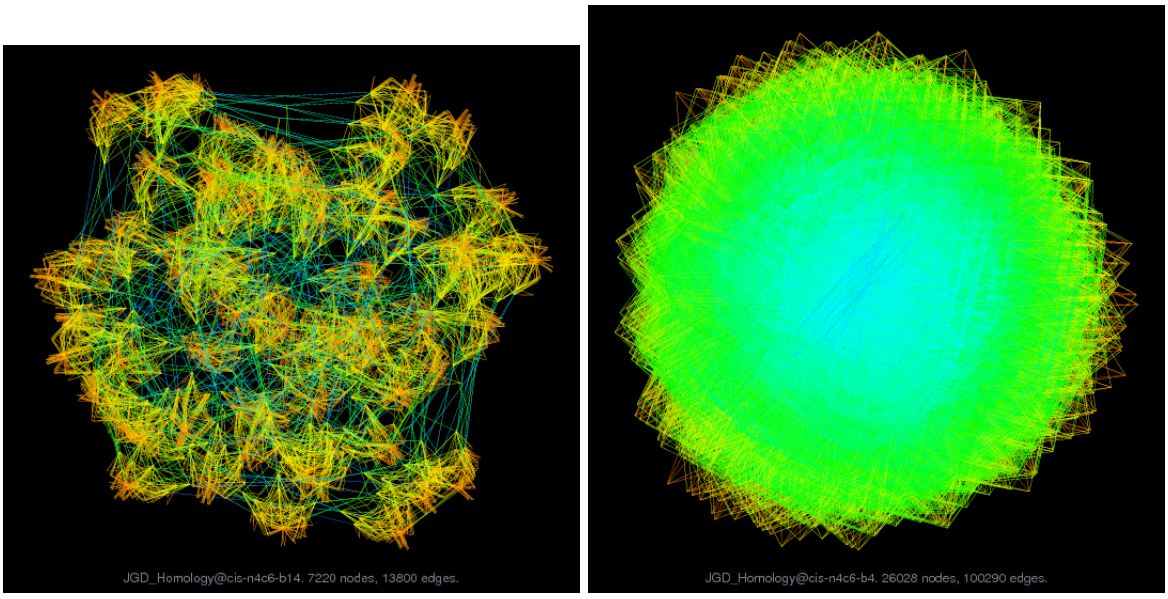
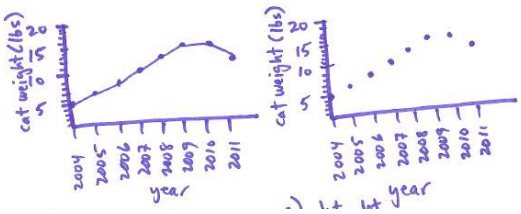


Figure 8.21: Multi-level graph drawing with sfdp [Hu 05]. a) Cluster structure is visible for a large network of 7220 nodes and 13,800 edges. b) A huge graph of 26,028 nodes and 100,290 edges is a "hairball" without much visible structure. From Yifan Hu's Gallery of Large Graphs, JDG_Homologycis-n4c6, b14 and b4. http://www.research.att.com/~yifanhu/GALLERY/GRAPHS/index1.html

the simplest version of the networks first, and then improving the layout with the more and more complex versions, both the speed and quality of the layout can be improved. These approaches do better at avoiding the local minimum problem.

Example: sfdp

Figure 8.21a shows a network of 7220 nodes and 13,800 edges using the multi-level sfdp algorithm [Hu 05], where the edges are colored by length. Significant cluster structure is indeed visible in the layout, where the dense clusters with short orange and yellow edges can be distinguished from the long blue and green edges between them. However, even these sophisticated techniques hit their limits with sufficiently large networks and fall prey to the hairball problem. Figure 8.21b shows a network of 26,028 nodes and 100,290 edges, where the sfdp layout does not show much visible



(a) (b)

Figure 8.22: Line charts versus dotplots. (a) Dotplots use a point mark to show the value for each item. (b) Line charts use point marks connected by lines between them.

structure. The enormous number of overlapping lines leads to overwhelming visual clutter caused by occlusion.

Technique	**sfdp (multi-level force-directed placement)**
Data Types	network
Derived Data	cluster hierarchy atop original network
View Comp.	connection
Abstract Tasks	path following, topological structure
Scalability	nodes: 1K - 10K edges: 1K - 10K node/edge density: E < 4N

Example: Line Charts

A **line chart** is like a dotplot where additional lines connect the point marks together. Figure 8.22 shows a dotplot and a line chart for the same dataset side by side, plotting the weight of a cat over several years. The trend of constantly increasing weight, followed by loss after a veterinarian-imposed diet regime in 2010, is emphasized by the connecting lines.

Line charts, dotplots, and bar charts all show one value attribute and one key attribute with a rectilinear spatial layout. All of these chart types are often augmented to show a second categorical attribute using color or stipple channels. They use one spatial position channel to express a quantitative attribute, and use the other direction for a second key attribute. The difference is that line charts also use connection marks to emphasize the or-

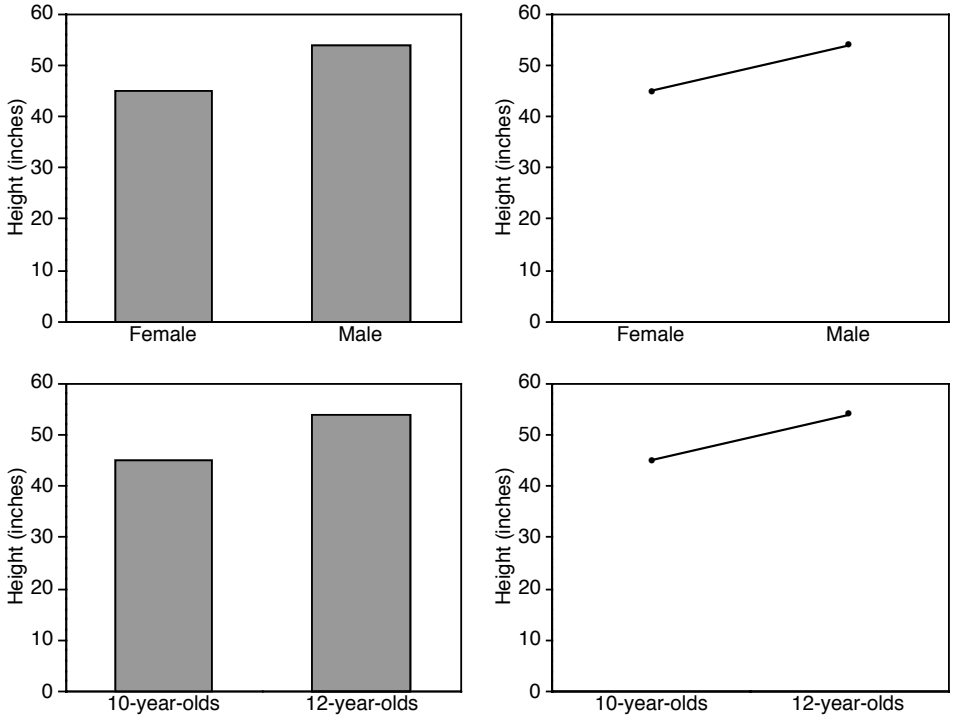


Figure 8.23: Bar charts and line charts both encode a single attribute. Bar charts encourage discrete comparisons while line graphs encourage trend assessments. Line charts should not be used for categorical data, as in the upper right, because their implications are misleading. From [Zacks and Tversky 99], Figure 2.

dering of the items along the key axis by explicitly showing the relationship between one item and the next, and imply trend relationships.

Technique	**line chart**
Data Types	table: 1 quantitative value attribute, 1 ordered key attribute
View Comp.	point marks, connection marks, 1 position channel expressing quantitative value, 1 position channel separating ordered key

Thus, line charts should be used for ordered keys but not categorical keys. A line chart used for categorical data violates the expressiveness

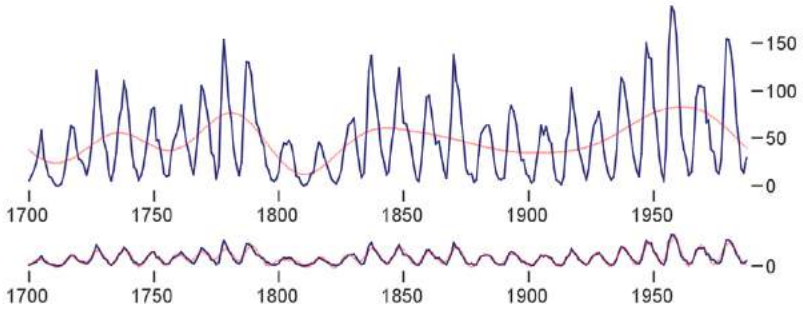


Figure 8.24: The multiscale banking to 45° technique exploits our orientation resolution accuracy at the diagonal. (a) An aspect ratio close to 4 emphasizes low-frequency structure. (b) An aspect ratio close to 22 shows higher-frequency structure: cycle onset is mostly steeper than the decay. From [Heer and Agrawala 06], Figure 5.

principle, since it visually implies a trend where one cannot exist. This implication is so strong that it can override common knowledge. Zacks and Tversky studied how people answered questions about the categorical data type of gender versus the quantitative data type of age, as shown in Figure 8.23 [Zacks and Tversky 99]. Line charts for quantitative data elicited appropriate trend-related answers such as "Height increases with age." Bar charts for quantitative data elicited equally appropriate discrete-comparison answers such as "Twelve year olds are taller than ten year olds". However, line charts for categorical data elicited inappropriate trend answers such as "The more male a person is, the taller he/she is".

When designing a line chart, an important question to consider is its **aspect ratio**: the ratio of width to height of the entire plot. While many standard charting packages simply use a square or some other fixed size, in many cases this default choice hides dataset structure. The relevant perceptual principle is that our ability to judge angles is more accurate at exact diagonals than at arbitrary directions. We can easily tell that an angle like 43° is off from the exact 45° diagonal, whereas we cannot tell 20° from 22°. The technique of **banking to 45°** computes the best aspect ratio for a chart in order to maximize the number of line segments that fall close to the diagonal. Multiscale banking to 45° automatically finds a set of informative aspect ratios using techniques from signal processing to analyze the line graph in the frequency domain, with the derived variable of the power spectrum. Figure 8.24 shows the classic sunspot example dataset. The aspect ratio close to 4 shows the low-frequency oscillations in the maximum values of each sunspot cycle. The aspect ratio close to

22 shows that many cycles have a steep onset followed by a more gradual decay. The blue line graphs the data itself, while the red line is the derived locally weighted regression line showing the trend.

8.2.2 Containment

Two of the visual encodings in Figure 8.25 use containment: the treemap in Figure 8.25g consisting of nested rectangles, and the nested circles of Figure 8.25f.

Containment marks are very effective at showing complete information about hierarchical structure, rather than pairwise relationships between two items at once. The **treemap** technique is an alternative to node-link tree drawings, where the hierarchical relationships are shown with containment rather than connection. All of the children of a tree node are enclosed within the area allocated that node, creating a nested layout. The size of the nodes is be mapped to some attribute of the node. Figure 8.18c is a treemap view of the same dataset as Figure 8.18a and b, a computer file system, where two identical subdirectories are highlighted. Here, node size encodes file size. Containment marks are not as effective as the pairwise connection marks for network tasks focused on topological structure, such as tracing paths through the network, but it is much more effective for showing tasks that pertain to understanding attribute values at the leaves of the tree. They are often used when hierarchies are shallow rather than deep. Treemaps are very effective for spotting the outliers of very large attribute values, in this case large files.

Technique	**treemap**
Data Types	tree
View Comp.	area marks, containment, rectilinear
Abstract Tasks	understanding attributes at leaf nodes
Scalability	leaf nodes: 1M edges: 1M

Although connection and containment marks that depict the link structure of the network explicitly are very common ways to encode networks, they are not the only way. In most of the trees in Figure 8.25, the spatial position channel is explicitly used to show the tree depth of a node. However, three layouts show parent-child relationships with spatial position as well. The rectilinear icicle tree of Figure 8.25c and the radial concentric circle tree of Figure 8.25e show tree depth with one spatial dimension and parent-child relationships with the other. Similarly, the indented outline tree in of Figure 8.25h shows parent-child relationships with relative vertical position, in addition to tree depth with horizontal position.

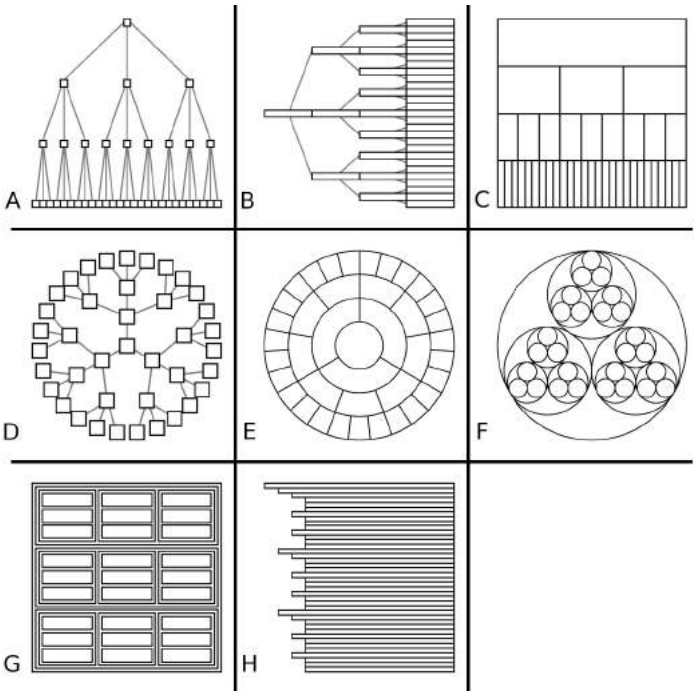
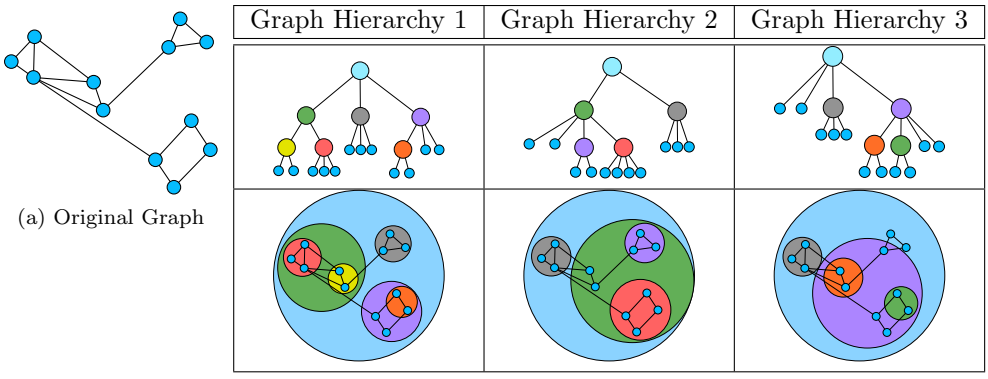


Figure 8.25: Eight visual encodings of the same tree dataset, using different combinations of visual channels. a) Rectilinear vertical node-link, using connection, with vertical spatial position showing tree depth. b) Rectilinear horizontal layered node-link, using connection, with horizontal spatial position showing tree depth. c) Icicle, with vertical spatial position and size showing tree depth, and horizontal spatial position showing link relationships. d) Radial node-link, using connection, with radial spatial position showing tree depth. e) Concentric circles, with spatial position and size showing tree depth and radial spatial position showing link relationships. f) Nested circles, using radial containment, with nesting level and size showing tree depth. g) Treemap, using rectilinear containment, with nesting level and size showing tree depth. g) Indented outline, using spatial position channels, with horizontal spatial position showing tree depth and . From [McGuffin and Robert 10], Figure 1.

Example: GrouseFlocks

Containment is often used for exploring multi-level graphs, where the



(a) Original Graph

(b) Graph Hierarchies

Figure 8.26: GrouseFlocks uses containment to show graph hierarchy structure. (a) Original graph. (b) Several alternative hierarchies built from the same graph. The hierarchy alone is shown in the top row. The bottom row combines the graph encoded with connection with a visual representation of the hierarchy using containment. From [Archambault et al. 08], Figure 3.

original dataset is augmented by a derived hierarchy that groups nodes of the original graph together. In the `sfdp` example above, there was a specific approach to coarsening the network that created a single derived hierarchy. That hierarchy was used only to accelerate force-directed layout, and was not shown directly to the user. In the GrouseFlocks system, users can explore multiple different possible hierarchies. Figure 8.26 shows how the use of containment marks for the associated hierarchy and connection marks for the original graph links can be combined in a single view, as in the bottom row. In contrast, if the hierarchy is shown with connection links, as in the top row, it is difficult to combine it with the layout of the original graph.

System	**GrouseFlocks**
Data Types	network
Derived Data	cluster hierarchy atop original network
View Comp.	connection marks for original network, containment marks for cluster hierarchy

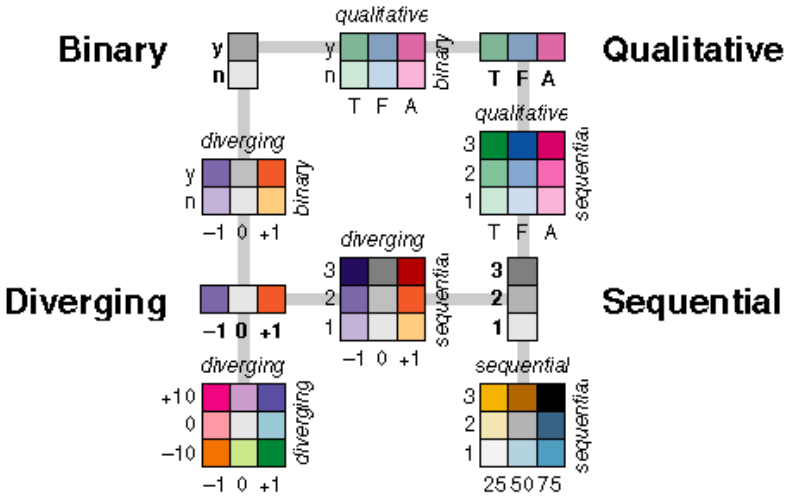


Figure 8.27: The colormap taxonomy mirrors the data type taxonomy: categorical versus ordered, and within ordered sequential and diverging. Redrawn after [Brewer 99].

8.3 Using Color

A **colormap** specifies a mapping between colors and data values; that is, a visual encoding with color.

Figure 8.27 shows the taxonomy of colormaps; it is no coincidence that it mirrors the taxonomy of data types. Colormaps can be categorical or ordered, and ordered colormaps can be either sequential or diverging. Of course, it is important to match colormap to data type characteristics. Colormaps for ordered data should use of the *how much* channels of luminance and saturation, since the *what* channel of hue does not have an implicit ordering.

Colormaps can either be a **continuous** range of values, or **segmented** into discrete bins of color.[7] Continuous colormaps are heavily used for showing quantitative attributes, especially those associated with inherently spatial fields. Segmented colormaps are suitable for categorical data. For ordinal data, segmented colormaps would emphasize its discrete nature,

[7]Vocabulary note: There are many synonyms for segmented: quantized, stepped, binned, discretized, and discrete. Colormapping is also called pseudo-coloring, especially in earlier literature. An early synonym for colormap is color ramp.

Figure 8.28: A suggested set of discriminable categorical colors. From [Ware et al. 02], Figure 4.21.

while continuous would emphasize its ordered nature.

8.3.0.1 Categorical Colormaps **Categorical colormaps** use color to encode categories and groupings.[8] They are very effective when used appropriately; for categorical data, they are the next best channel after spatial position. As discussed in Section 5.4.2.5, the number of discriminable colors for coding small separated regions is limited to around a dozen. In this case, the best choice is usually to consider color as an integral *what* channel, rather than trying to encode completely separate variables with the three subchannels of hue, saturation, and luminance. The best choices are the fully saturated and easily nameable colors: red, blue, green, yellow, white, black. A next set of good possibilities when more colors are needed are orange, brown, pink, magenta, and purple. Figure 8.28 shows Ware's suggested set [Ware et al. 02](p. 126). However, when there is a hierarchical structure for the categories, it is possible to create families of colors that are perceptually grouped together by considering the interaction of these three channels. One way is to fix a base set of hues fixed and vary the lightness across a small number of levels, as in the categorical-sequential examples at the top of Figure 8.27.

When coding large areas, use pale low-saturation colors rather than highly saturated colors. For example, Figure 8.29 compares a 10-element low-saturation map to an 8-element high-saturation map. The low-saturation map is well suited for large regions, leaving fully saturated colors for small road marks. In contrast, the 8-element map that uses highly saturated colors is much too bright for the large areas shown here, but would be a good fit for small marks. Both of these examples were created with ColorBrewer2, a useful resource for colormap construction available online at `www.colorbrewer2.org`.

8.3.0.2 Ordered Colormaps: Sequential and Diverging **Sequential** colormaps range from a minimum value to a maximum value. If only the luminance channel is used, the result is a greyscale ramp. When they incorporate hue, one end of the ramp is a specific hue at full saturation and brightness. If saturation is the variable, the other end is pale or white;

[8] Vocabulary note: Categorical colormaps are also known as qualitative colormaps.

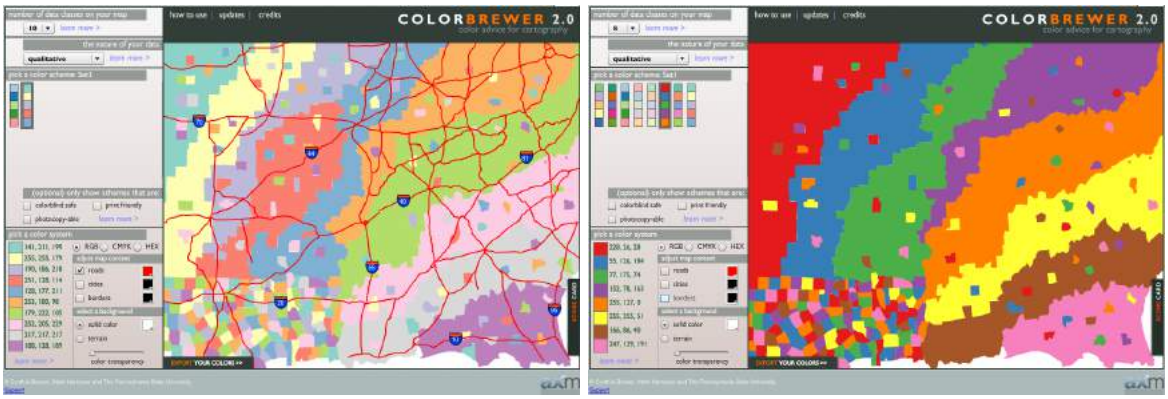


Figure 8.29: Saturation and area. (a) The 10-element low-saturation map works well with large areas. (b) The 8-element high-saturation map would be better suited for small regions, and works poorly for these large areas. Created with ColorBrewer2, `http:www.colorbrewer2.org`.

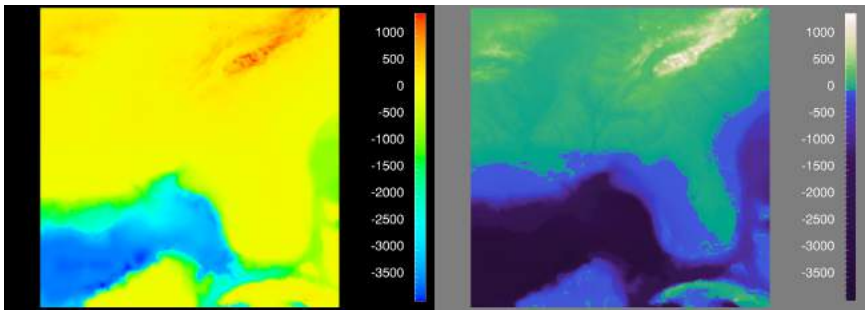


Figure 8.30: Colormap design has many potential pitfalls. (a) Problems with the common continuous rainbow colormap include perceptual nonlinearity and the mismatch of using the coarse-grained *what* hue channel to show fine-grained ordered magnitude information. (b) A colormap that combines monotonically increasing luminance with multiple hues for semantic categories, with a clear segmentation at the zero point, shows more structure. From [Rogowitz and Treinish 98], Figure 1.

when luminance is the varying quantity, the other end is dark or black. **Diverging** colormaps have two hues at the endpoints and a neutral color as a midpoint, such as white, high-luminance yellow, grey, or black.

Colormapping is a powerful and flexible technique, but colormap design has many pitfalls for those who have not fully assimilated the principles

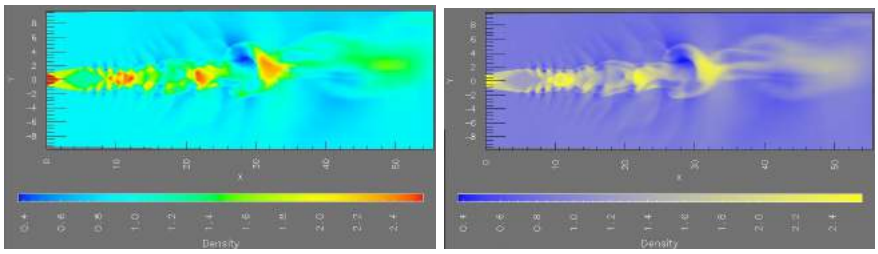


Figure 8.31: Colormaps and spatial frequency. (a) Using many hues, as in this rainbow colormap, emphasizes fine-grained structure. (b) Using only two hues, the blue-yellow colormap emphasizes large-scale structure. From [Bergman et al. 95], Figures 2 and 1.

discussed in Section 5.4.2. Figure 8.30 shows an example. A rather unfortunate default in many software packages is the continuous rainbow colormap, which suffers from three serious problems at the perceptual level. First, hue is used to indicate order, despite being a *what* channel without an implicit perceptual ordering. Second, the scale is not perceptually linear: steps of the same size at different points in the colormap range are not perceived equally by our eyes. Consider the range of 1000 units in the Figure 8.30(a) example. While the range from -2000 to -1000 has three distinct colors, cyan and green and yellow, a range of the same size from -1000 to 0 simply looks yellow throughout. Third, fine detail cannot be perceived with the hue channel; the luminance channel would be a much better choice.

On the other hand, one advantage of the rainbow colormap is that people can easily discuss specific subranges because the differences are easily nameable: "the red part vs. the blue part vs. the green part". In colormaps that just use saturation or luminance changes, it is not easy to verbally distinguish sections – we cannot easily demarcate the "sort-of-bright red vs. the not-quite-so-bright red parts".

One way to get the best of both worlds is to design colormaps that use monotonically increasing luminance to show ordering in conjunction with varying hues for easy segmentation into categorical regions. Figure 8.30(b) shows the same data with a more appropriate colormap, where the luminance increases monotonically. Hue is used to create a semantically meaningful categorization: the viewer can discuss structure in the dataset, such as the dark blue sea, the cyan continental shelf, the green lowlands, and the white mountains.

The question of how many unique hues to use in colormaps for ordered data depends on whether the goal is to emphasize large-scale structure or fine detail. In the Florida example above, the goal is to see fine detail,

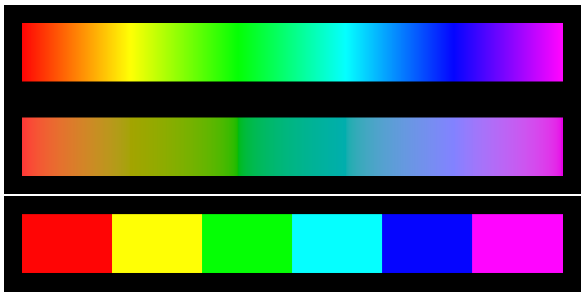


Figure 8.32: Avoiding perceptually nonlinearity in colormaps. (a) The standard rainbow colormap is perceptually nonlinear. (b) The perceptually linear rainbow is less bright, with a decreased dynamic range [Kindlmann 02]. (c) The segmented rainbow works well for categorical data when the number of categories is small.

and many hues were used. To show large-scale structure in the data, then smoothly varying between only a few hues is best. Figure 8.31 shows an example comparing a rainbow colormap to one where each end of the colormap is a different fully saturated hue, and they meet at a middle point of grey. There are only two hue-based semantic categories, the blue side versus the yellow side. In the language of signal processing, the emphasis is on the low-frequency components of the data.

It is possible to create a perceptually linear rainbow colormap, but at the cost of losing part of the dynamic range because the fully saturated colors are not available for use. Figure 8.32 shows an example [Kindlmann 02]. The perceptually linear rainbow is so much less bright than the standard one that it seems almost dingy, so this solution is not common.

Rainbows are not always bad; a segmented rainbow colormap is a fine choice for categorical data with a small number of categories. Figure 8.32 also shows this example. Segmented rainbows could also be used for ordered data; while not ideal, at least the perceptual nonlinearity problem is solved because the colormap range is explicitly discretized into bins. Using a segmented colormap on quantitative data is equivalent to transforming the datatype from quantitative to ordered. This choice is most legitimate when task-driven semantics can be used to guide the segmentation into bins. The intuition behind the technique is that it is better to deliberately bin the data explicitly, rather than relying on the eye to create bins that are of unequal size and often do not match meaningful divisions in the underlying data.

8.3.0.3 Bivariate Colormaps The safest use of the color channel is to visually encode a single variable; these colormaps are known as **univariate**.

Figure 8.27 includes several colormaps that encode two separate variables, called **bivariate**. When one of the attributes is categorical and the other is sequential with only two levels, the result is fairly comprehensible. When both are categorical, results will be poor [Wainer and Francolini 80], and thus there are no such map in the Figure8.27 taxonomy. While encoding combinations of two sequential or diverging variables is possible, empirical studies show that many people have difficulty in interpreting the meaning of these colormaps.

8.3.0.4 Colorblind-safe Design Colormap designers should take the common problem of red-green colorblindness into account. It is a sex-linked inherited trait that affects 8% of males and a much smaller proportion of females, 0.5%. As discussed in Section 5.4.2, in the common forms of color blindness the ability to sense along the red-green opponent color axis is limited or absent. The problem is not limited to simply telling red apart from green; many pairs that are discriminable to people with normal color vision are confused, including red from black, blue from purple, light green from white, and brown from green.

On the theoretical side, the safest strategy is to avoid using only the hue channel to encode information: design categorical colormaps that vary in luminance or saturation, in addition to hue. Clearly, avoiding colormaps that emphasize red-green, especially divergent red/green ramps, would be wise. In some domains there are strong conventions with the use of red and green, so those user expectations can be accommodated by ensuring luminance differences between reds and greens. On the practical side, an excellent way to ensure that a design uses colors that are distinguishable for most users is to use simulation tools such as VisCheck (`http://www.vischeck.com`).

8.4 Combining Views

Many visualization systems use multiple views rather than just a single one. A **view** is a single layout. [9] That is, rather than combining all information into a single, complex, monolithic view, the visualization system uses multiple, simpler views.

A very powerful visualization method is to show multiple views side by side. As the slogan *Eyes Over Memory* argues, moving our eyes between two views that are simultaneously visible has lower cognitive load than consulting our memory to compare a current view to what was seen before.

For more on the ideas behind the slogan *Eyes Over Memory*, see Section **??**.

[9] Vocabulary note: Other synonyms for view include *display*, *window*, *panel*, and *pane*.

The obvious and significant cost to side-by-side views is the screen real estate required to show these multiple windows at the same time. When two views are shown side by side, they each get only half the area that a single view could provide. Much of the analysis of when to use what view combination method comes down to a tradeoff of this cost versus their other benefits. There are two alternatives for the use of screenspace with multiple side-by-side views. One choice is to always show the multiple views side by side, and another is to have a popup view that is only shown in response to a user action.

In some systems the location of the views is arbitrary and left up to the window system or the user, especially when only a few views are used. In other systems view location is explicitly controlled, especially by aligning them into a list or matrix to allow precise comparison between them. Section 8.1.2 has already covered the question of how to order and align these views as part of the larger discussion of region ordering and alignment. In this sense, a view is just one kind of region, a complex one as opposed to a simple one.

Section 8.5 will continue with an extensive discussion of the further design choices involved with coordinating multiple side by side views.

A common alternative to having multiple views side by side is to have a single view that changes over time as the user interacts with it with methods such as filtering, aggregation, and navigation. Chapter 9 will cover these methods in detail; this choice is included here as a reminder that these choices are different solutions to the same problem.

A third alternative to side by side views is to superimpose two views by layering them on top of each other in a single shared frame. Section 8.6 will cover this method in more detail.

8.5 Coordinating Views

A key design choice with multiple side by side views is how to coordinate between them to create **linked views**. [10] The taxonomy classifies view coordination methods according to four design choices: do the views share the same visual encoding, or use different ones? Do the views show the same data, is one a subset of the other, or do they show a disjoint partitioning? Is navigation synchronized between the views? Are they linked by explicit marks between them?

[10] Vocabulary note: Linked views, multiple views, coordinated views, coordinated multiple views, and coupled views are all synonyms for the same fundamental idea.

8.5.1 Encoding Channels Shared

The most common method for linking views together is to have some form of shared visual encoding where a visual channel is used in the same way across the multiple views. Two views could have the same visual encoding, where all channels are shared; these are called **shared-encoding** views in the taxonomy. If any of the visual encoding channels are different, the views are called **multiform**.[11] For example, in a multiform system two views might have different spatial layouts but the same color coding, so there is a shared encoding through the color channel. Or they could be aligned in one spatial direction but not in the other. This form of linking can be done with completely static views.

The true power of linked views comes with interactivity. One of the most common forms of linking is **linked highlighting**, where items that are interactively selected in one view are immediately highlighted in all other views using in the same highlight color color. [12] Linked highlighting is a special case of a shared visual encoding in the color channel. The central benefit of linked highlighting is in seeing how a region that is contiguous in one view is distributed within another.

The rationale behind multiform encoding across views is that a single monolithic view constructed with the methods described in Chapter 8 has strong limits on the number of attributes that can be simultaneously without introducing too much visual clutter. Although simple operations can often be fully supported by a single view of a specific dataset, more complex user tasks often cannot. With multiple views as opposed to a single view, each view does not have to show all of the attributes; they can each show only a subset of the attributes, avoiding the visual clutter of trying to superimpose too many attributes in a single view. Even if two views show exactly the same set of attributes, the visual channels used to encode can differ. The most important channel change is what spatial position encodes; this most salient channel dominates our use of the view, and strongly affects what tasks it best supports. Multiform views can thus exploit the strengths of multiple visual encodings.

Example: EDV

The EDV system features linked highlighting between views [Wills 95]. Figure 8.33 shows a baseball statistics dataset with linked bar charts, scatterplots, and a histogram [Wills 95]. The viewer has selected players with

[11] Vocabulary note: The generic term *multiple views* is often used as a synonym for *multiform views*.

[12] Vocabulary note: Linked highlighting is also called *brushing*.

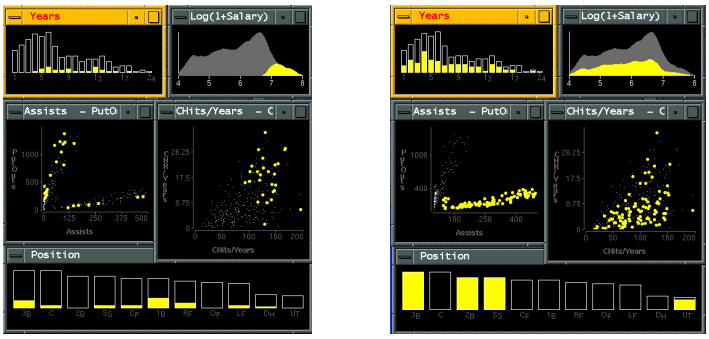


Figure 8.33: Linked highlighting between views shows how regions that are contiguous in one view are distributed within another. (a) Selecting the high salaries in the upper right window shows different distributions in the other views. (b) Selecting the bottom group in the Assists-Putouts window shows that the clump corresponds to specific positions played. From [Wills 95], Figures 4 and 5.

high salaries in the smoothed histogram view on the upper right. The distribution of these players is very different in the other plots. In the `Years` played view bar chart on the upper left, there are no rookie players in this group. The `Assists-PutOuts` scatterplot does not show much correlation with salary. Inspecting the `CHits/Years` plot showing batting ability in terms of career home runs against average career hits shows that the hits per year is more correlated with salary than the home runs per year. The bottom `Position` window shows a loose relationship between salary and the player's position in the field. The `Assists-PutOuts` window shows a clustering into two major groups. Figure 8.33(b) shows the result of selecting the bottom clump. The bottom `Position` window shows that this clump corresponds to specific positions played, whereas these players are fairly evenly distributed in the other windows.

System	**Exploratory Data Visualizer (EDV)**
Data Types	tables
View Comp.	bar charts, scatterplots, and histograms
View Coordination	linked highlighting

8.5.2 Data Shared

A second consideration is how much data is shared between the two views. There are three alternatives: both views could each show all of the data (shared-data); one could show a subset of what is in the other (overview-detail), or they could show different partitions of the dataset into disjoint pieces (small-multiple).

Showing all of the data in each view is common with multiform systems, where the encoding differs. In the taxonomy of methods, these are called **shared-data** views.

The method of showing a subset of the data from one view in another is called **overview-detail**. As the name suggests, one of the views shows information about the entire dataset to provide an overview of everything. One or more additional views shows more detailed information about a subset of the data that is dynamically chosen by the viewer from the larger set depicted in the broader view. A common choice is to have only two views, one for overview and one for detail. When the dataset has multilevel structure at discrete scales, multiple detail views are appropriate to show structure at these different levels, allowing the user to zoom down in to successively smaller subsets of the data.

A common approach is to have two multiple views always available side by side that use same visual encoding to show different viewpoints of the same dataset; these are classified as shared-encoding overview-detail views. When two of these views are shown they often have different sizes, a large one with many pixels versus a small one with few. Sometimes the large window is the main view for exploring the details and the small window is the zoomed-out overview; sometimes the large view devoted to the overview, with a smaller window for details.

For more on changing the viewpoint with navigation, see Section 9.3.

Example: Geographic Birdseye

Interactive online geographic maps are a familiar example of shared-encoding overview-detail views, with a large map exploration view augmented by a small birdseye view providing an orienting overview, as shown in Figure 8.34. A small rectangle within it shows the region viewable within the detail view. The minimum navigational linkage is unidirectional, where position and size of the rectangle in the overview updates as the user pans and zooms within the large detail view. With bidirectionally linked views, the rectangle can also be moved within the small view to update the region shown in the large one.

. . . . . .

Another common approach to overview-detail is to have multiform views,

Figure 8.34: Overview-detail example with geographic maps, where the views have the same encoding and dataset; they differ in viewpoint and size. From [Cockburn et al. 08], Figure 1a.

where the detail view has a different visual encoding than the overview. This detail view that shows additional information about one or more items selected in a central main view is sometimes called a **detail-on-demand** view. This view might be a popup window near the cursor, or in a fixed window in another part of the display.

Example: Multiform Overview+Detail for Microarrays

Figure 8.35 shows an example of a multiform interface designed to support the visual exploration of microarray time-series data by biologists [Craig and Kennedy 03]. Microarrays measure gene expression, which is the activity level of a gene. They are used to compare gene activity across many different situations; examples include different times, different tissue types such as brain vs. bone, exposure to different drugs, samples from different individuals, or samples from known groups such as cancerous or non-cancerous.

The interface focuses on the coordination between the scatterplot view in the center and the graph view in the upper left. The designers carefully analyzed the problem-level user requirements to generate an appropriate data and task abstraction, and concluded that no single view would suffice. The five abstract tasks were finding genes that were on or off across the whole time period, finding genes whose values rose or fell over a specified time window, finding genes with similar time-series patterns, relating all these sets to known functional groups of genes, and exporting the results for use within other tools. The data abstraction identified five key parameters: the raw data types of time and microarray value, and three derived parameters based on value change, percentage of max value, and fold change (a log-scale change measure frequently used in microarray data analysis).

The graph view shows time series data plotted with globally superim-

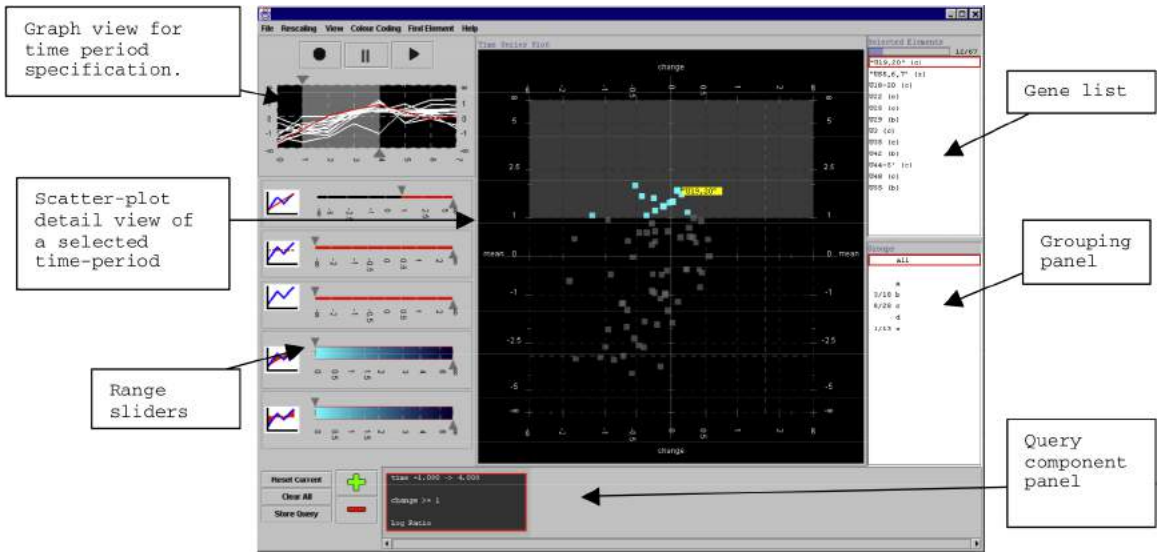


Figure 8.35: Multiform interface for microarray exploration features a central scatterplot linked with the graph view in the upper left. Courtesy of Jessie Kennedy. From [Craig and Kennedy 03], Figure 3.

For more on superimposed line charts, see Section 8.6.1.

posed line charts. Each line mark represents a gene, with the horizontal axis showing time and the vertical axis showing value. The user interacts with this overview to select a time period of interest to show in the scatterplot detail view by changing the position or width of the time slider. The time-series graph view does not support visual queries about value change or fold change, which are derived values computed within the time window chosen. In the scatterplot view, the horizontal axis can be set to either of these derived variables. In the scatterplot, each gene is represented by a point mark. This view also encodes the functional groups with color coding, and dynamically shows the label for the gene under the cursor.

The list view on the right shows the gene names for all genes within the active time window, ordered alphabetically. Although a text list might appear to be a trivial visualization when considered as a standalone view, these kind of simpler views can play a useful role in a multiple-view system. This particular list view provides a textual overview and also support both browsing and search. While interaction via hovering over an item is useful for discovering the identify of a mark in a specific place, it would a very frustrating way to get an overview of all labels because the user would have to click in many places, and try to remember all of the previous labels. Glancing at the list view provides a simple overview of the names, and also

allows the user to quickly select an item with a known name.

System	**Multiform Overview+Detail for Microarrays**
Data Types	multidimensional table: 1 categorical key attrib (gene), 1 ordered key attrib (time), 1 quantitative value attrib (microarray measurement of gene activity at time)
Derived Data	3 quantitative value attribs: (value change, percentage of max value, fold change)
View Comp.	multiple views: superimposed line charts, scatterplots, and lists
Multiple Views	side by side multiform views, with linked highlighting and overview+detail filtering of time range. superimposed line charts

The third alternative for data sharing between views is to show a **different partition** of the dataset in each. Shared-encoding, different-partition views are often called **small multiples**. The shared visual encoding means that the views have a common reference frame so that comparison of spatial position between them is directly meaningful. Small multiples are often aligned into a list or matrix to support comparison with the highest precision. Using small multiples is in some sense the inverse of multiform approach, since the encoding is identical but the data differs.

For more on partitioning data, see Section 8.1.3.3.

For more on aligning regions, see Section 8.1.2.

The weakness of small multiples, as with all side-by-side view combinations, is the screen real estate required to show all of these views simultaneously. The operational limit with current displays of around one million pixels is a few dozen views with several hundred elements in each view.

The strength of the small multiples method is in making different partitions of the dataset simultaneously visible side by side, allowing the user to glance quickly between them with minimal interaction cost and memory load. Small multiples are often used as an alternative to animations, where all frames are visible simultaneously rather than shown one by one. As discussed in Section 7.2, animation imposes a massive memory load when the amount of change between each frame is complex and distributed spatially between many points in the scene.

Example: Cerebral

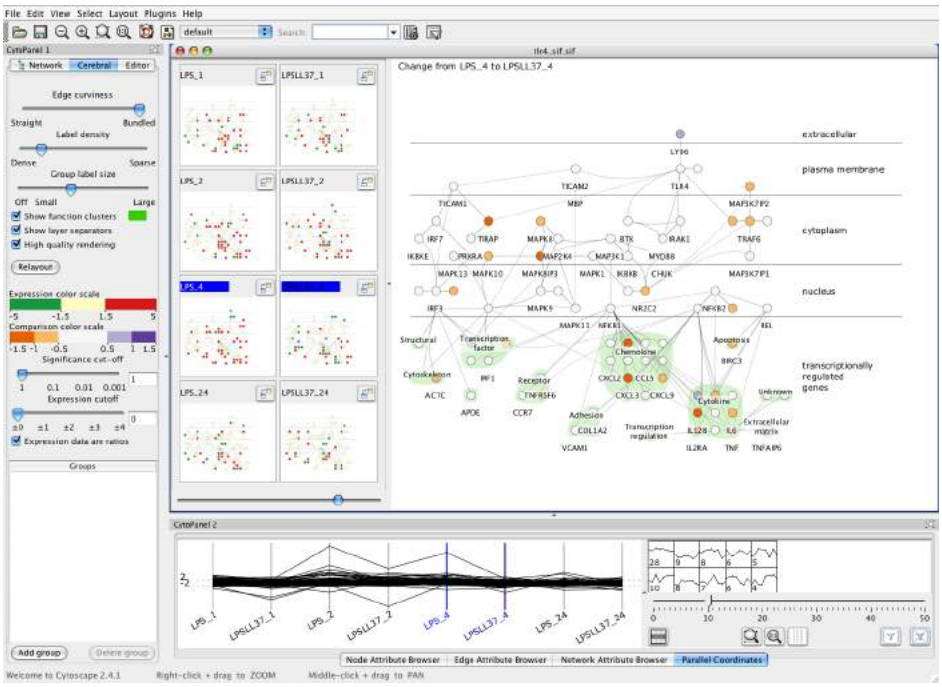


Figure 8.36: Cerebral uses small-multiple views to show the same base graph of gene interactions colored according to microarray measurements made at different times. The coloring in the main view uses the derived variable of the difference in values between the two chosen views. From [Barsky et al. 08], Figure 2.

Figure 8.36 shows an example of small-multiple views in the Cerebral system [Barsky et al. 08]. The dataset is also from the bioinformatics domain, a multidimensional table with the two keys of genes and experimental condition and the value attribute of microarray measurements of gene activity for the condition. The large view on the upper right is a node-link network diagram where nodes are genes and links are the known interactions between genes, shown with connection marks. The layout also encodes an ordered attribute for each node, the location within the cell where the interaction occurs, with vertical spatial position. Containment marks show the groups of co-regulated genes. The small-multiple views to the left of the large window show a partitioning of the dataset by condition. The views are aligned to a matrix and are reorderable within it.

In each small-multiple network view the nodes are colored with a diverging red-green colormap showing the quantitative attribute of gene activity

for that view's condition. (In this domain there is a strong convention for the meaning of red and green that arose from raw images created by the optical microarray sensors that record fluorescence at specific wavelengths, so the diverging colormap uses luminance contrast to ensure distinguishability between red and green for all viewers.) In the large network view, the color coding for the nodes is a diverging orange-blue colormap based on the derived variable of difference in values between the two selected small multiples, whose titlebars are highlighted in blue.

Cerebral is also multiform; the view at the bottom uses parallel coordinates for the visual encoding, along with a control panel for data clustering.

System	**Cerebral**
Data Types	multidimensional table: 1 categorical key attrib (gene), 1 categorical key attrib (condition), 1 quantitative value attrib (gene activity at condition). network: nodes (genes), links (known interaction between genes), 1 ordered attrib on nodes: location within cell of interaction,
Derived Data	1 quantitative value attrib (difference between measurements for two partitions)
View Comp.	views: node-link network using connection marks, vertical spatial position expressing interaction location, containment marks for co-regulated gene groups, diverging colormap; small-multiple network views partitioned on condition, aligned in matrix; parallel coordinates
Multiple Views	side-by-side small-multiple and multiform views, linked highlighting

8.5.3 Navigation Synchronized

With **linked navigation**, movement in one view triggers movement in the others. For example, linked navigation is common with map views that have a smaller birdseye overview window in addition to a larger detail view, where interaction in the small window changes the viewpoint in the large one. Navigation is covered further in Section 9.3.

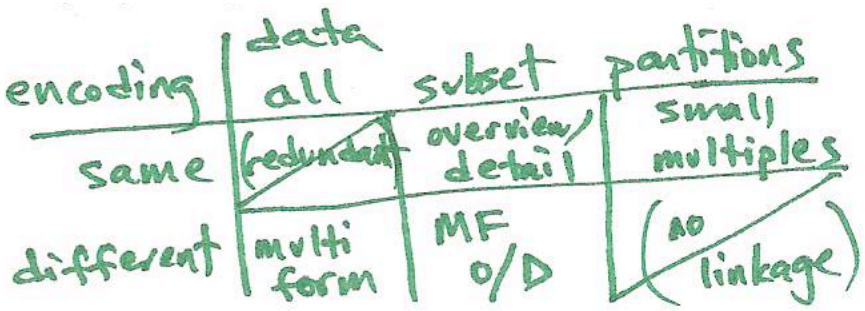


Figure 8.37: Summary of view coordination methods for sharing visual encoding and data.

8.5.4 Linking Views With Marks

A final choice with linking views is to explicitly show the links between items in different views with connection marks, rather than the more implicit use of shared visual encoding channels.

Example: Semantic Substrates

TODO

. . . . . .

8.5.5 Combinations

Figure **??** shows a summary of the view coordination methods for sharing visual encoding and data. The encoding could be the same or different; the data could be the same, a subset, or a partition. Two of the six possibilities are not useful. When everything is shared, with both the data and encoding identical, the two views would be redundant. When nothing is shared, with different data in each view and no shared channels in the visual encoding, there is no linkage between the views. Otherwise, the choices of sharing for encoding and data are independent. For example, the overview-detail method of creating subsets of the data can be used with either multiform or shared-encoding encoding method.

Complex systems will use these methods in combination, so in this book these terms are used to mean that at least one pair of views differs along that particular axis. For example, *multiform* means that at least one pair of views differs, not necessarily that every single view has a different encoding from every other one. Thus, a system might use both multiform

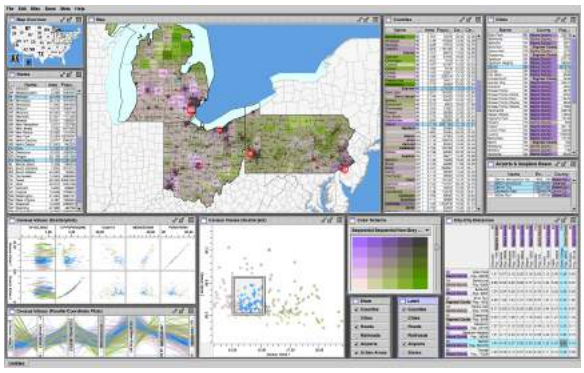


Figure 8.38: The Improvise toolkit [Weaver 04] was used to create this census visualization that has many forms of coordination between views. It has many multiform views, some of which use small multiples, and some of which provide additional detail information. Courtesy of Chris Weaver.

and shared-encoding view coordination.

Example: Improvise

Figure 8.38 shows a visualization of census data that uses many views. In addition to geographic information, the demographic information for each county includes population, density, genders, median age, percent change since 1990, and proportions of major ethnic groups. The system is multiform with geographic, scatterplot, parallel coordinate, tabular, and matrix views. These multiform views all share the same bivariate sequential-sequantial color encoding, documented with a legend in the bottom middle. A set of small-multiple views appears in the lower left in the form of a scatterplot matrix, where each scatterplot shows a different pair of attributes. All of the views are linked by highlighting: the blue selected items are close together in some views and spread out in others. A set of small-multiple reorderable list views result from partitioning the data by attribute. The list views allow direct sorting by and selection within an attribute of interest. The map in the upper left view is a small overview, with linked navigation to the large geographic detail view in the top middle.

System	**Improvise**
Data Types	geographic spatial and multidimensional table (census data): 1 key attrib (county), many quantitative attribs (demographic information)
View Comp.	views: scatterplot matrix, parallel coordinates, choropleth map with size-coded city points, birdseye map overview, scatterplot, reorderable text lists, text matrix. bivariate sequential-sequential colormap
Multiple Views	side-by-side, small-multiple, multiform, overview-detail views; linked highlighting

8.6 Superimposing Layers

Section 8.4 introduced the idea of superimposing views on top of each other as an alternative to showing them side by side, or changing a single view. This section expands on the design choices for this method in more detail.

Many visualization systems control visual clutter through visual layering. A **visual layer** is simply a set of objects spread out over a region, typically the entire display window, where the set of objects in each layer is a visually distinguishable group that can be told apart from objects in other layers at a perceptual level.

All of the visual channels that can be used for grouping are suitable for creating visual layers. One good way to make distinguishable layers is to ensure that each layer uses a different and nonoverlapping range of the visual channels active in the encoding. A common choice is to create two visual layers, a foreground versus a background. With careful design, several layers can be created. The term layer usually implies multiple objects spread through the region spatially intermixed with objects that are not part of that visual layer. However, a single highlighted object could be considered as constituting a very simple visual layer.

With the method of **superimposing layers**, multiple simple drawings are combined on top of each other into a single shared frame. All of the drawings have the same horizontal and vertical extent, and are blended

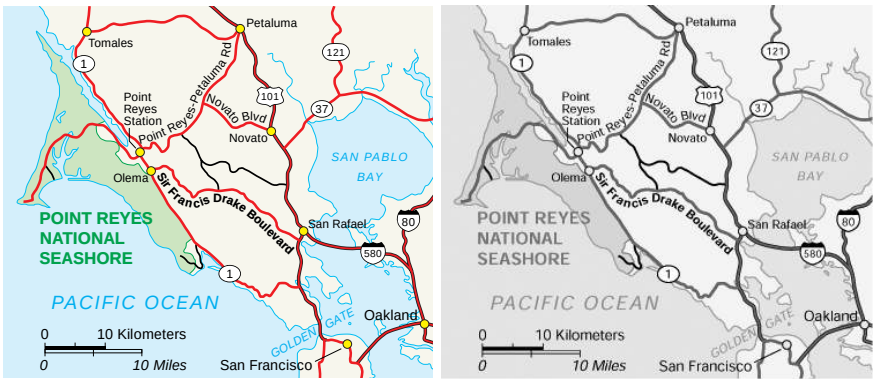


Figure 8.39: Static visual layering in maps. (a) The map layers are created by differences in the hue, saturation, luminance, and size channels on both area and line marks. (b) The greyscale view shows that each layer uses a different range in the luminance channel, providing luminance contrast. Courtesy of Maureen Stone, from [Stone 10].

together as if the single drawings are completely transparent where no marks exist. [13]

Superimposed layering can be done with a small set of **static layers** that do not change, or with **dynamic layers** that change in response to user interaction.

8.6.1 Static Layers

Static visual layering techniques are designed so that all of the layers are displayed simultaneously but the user can choose which to focus on with the selective direction of visual attention. Mapmakers usually design maps in exactly this way.

Example: Cartographic Layering

Figure 8.39 shows an example that lets the viewer easily shift attention between layers. Area marks form a background layer, with three different unsaturated colors distinguishing water, parks, and other land. Line marks form a foreground layer for the road network, with main roads encoded by wide lines in a fully saturated red color and small roads with thinner black

[13] Vocabulary note: In graphics terminology, combining them is an image-space compositing operation, where the drawings have an identical coordinate system.

lines. This layering works well because of the luminance contrast between the elements on different layers [Stone 10].

Technique	**Cartographic Layering**
Data Types	geographic spatial: regions and roads
Derived Data	1 quantitative value attrib: difference between measurements for two partitions
View Comp.	area marks for regions (water, parks, other land), line marks for roads, categorical colormap.
Multiple Views	superimposed static layers. distinguished with color saturation, color luminance, and size channels.

Example: Superimposed Line Charts

For instance, a common statistical graphic is to superimpose several lines representing different data items to create a combined chart, as in Figure 8.40(a). The alignment of the simple constituent drawings is straightforward: they are all superimposed directly on top of each other so that they share the same frame. This simple superimposition works well because the only mark is a thin line that is mostly disjoint with the other marks, so the amount of occlusion is small. Figure 8.40(b) shows that the view is still usable with even a few dozen items superimposed. However, this approach cannot scale to hundreds of lines, as shown in Figure 8.40(c).

Technique	**Superimposed Line Charts**
Data Types	multidimensional table: 1 ordered key attrib (time), 1 categorical key attrib (machine), 1 quantitative value attrib (CPU utilization)
View Comp.	views: line charts. partitioned by machine attrib
Multiple Views	superimposed static layers, distinguished with color
Scalability	ordered key attrib: hundreds categorical key attrib: dozen

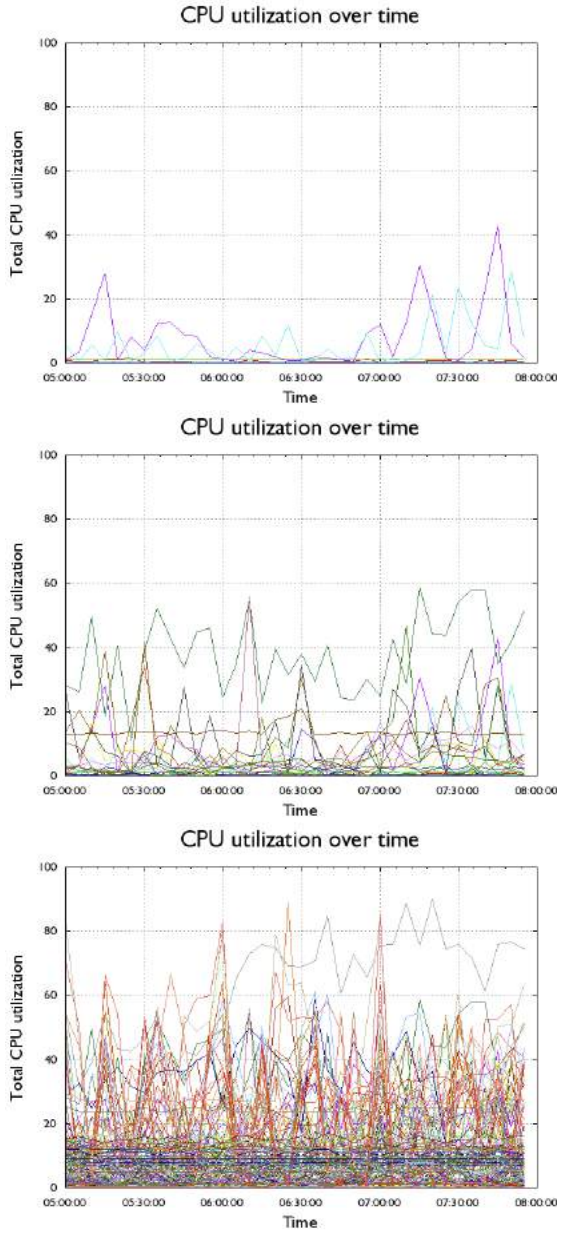


Figure 8.40: Multiple line charts can be superimposed within the same global frame. (a) A small number of items is easily readable. (b) Up to a few dozen lines can still be understood. (c) This technique does not scale to hundreds of items.

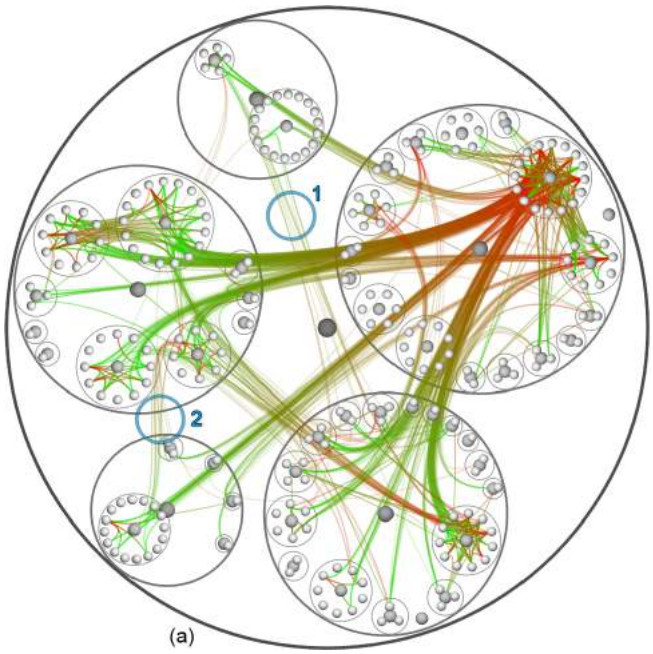


Figure 8.41: The hierarchical edge bundles technique shows a compound graph in three layers: the tree structure in back with containment circle marks, the red-green graph edges with connection marks in a middle layer, and the graph nodes in a front layer. From [Holten 06], Figure 13.

Example: Hierarchical Edge Bundles

A more complex example of static global superimposition is a technique for the combination of a network and a tree, as shown in Figure 8.41. A **compound graph** is the combination of a network and tree; that is, in addition to a base network with links that are pairwise relations between the network nodes, there is also a hierarchical grouping of the nodes. of hierarchical edges between those nodes. In other words, it is a combination of a network and a tree on top of it where the nodes in the network are the leaves of the tree. Thus, the interior nodes of the tree encompass multiple network nodes. The software engineering example in Figure 8.41 shows the call graph, namely which functions call what other functions, in conjunction with the hierarchical structure of where all of the functions are defined within the source code tree.

The **hierarchical edge bundles** technique [Holten 06] uses three visual layers. The tree structure is shown with containment marks, namely the grey enclosing circles on the back layer. The middle layer shows network edges with connection marks that are translucent red-green curves, and these are bundled together to minimize occlusion. The front layer has the nodes that are both the items in the network and the leaves of the tree shown as opaque filled grey circles. The technique does not rely on a specific spatial layout for the tree; it can be applied to many different tree layouts.

Layering is often carried out with modern graphics hardware to manage rendering with planes oriented in the screen plane that are blended together in the correct order from back to front, as if they were at slightly different 3D depths. This approach is one of the many ways to exploit modern graphics hardware to create complex drawings that have the spirit of 2D drawings, rather than true 3D scenes with full perspective.

Technique	**Hierarchical Edge Bundles**
Data Types	compound graph: network, hierarchy whose leaves are nodes in network
View Comp.	views: back layer shows hierarchy with containment marks colored grey, middle layer shows network links colored red-green, front layer shows nodes colored grey
Multiple Views	superimposed static layers, distinguished with color

8.6.2 Dynamic Layers

With interactive systems, the system can dynamically change the visual representation to make some chosen layer more salient than the others in response to the user selecting it as the focus. In these systems the number of possible layers can be huge, since they are constructed on the fly rather than chosen from a very small set of possibilities that must be simultaneously visually distinguishable.

Example: Cerebral

The Cerebral system, shown previously in Figure 8.36, also uses the method of dynamic layering. Figure 8.42 shows the dynamic creation of

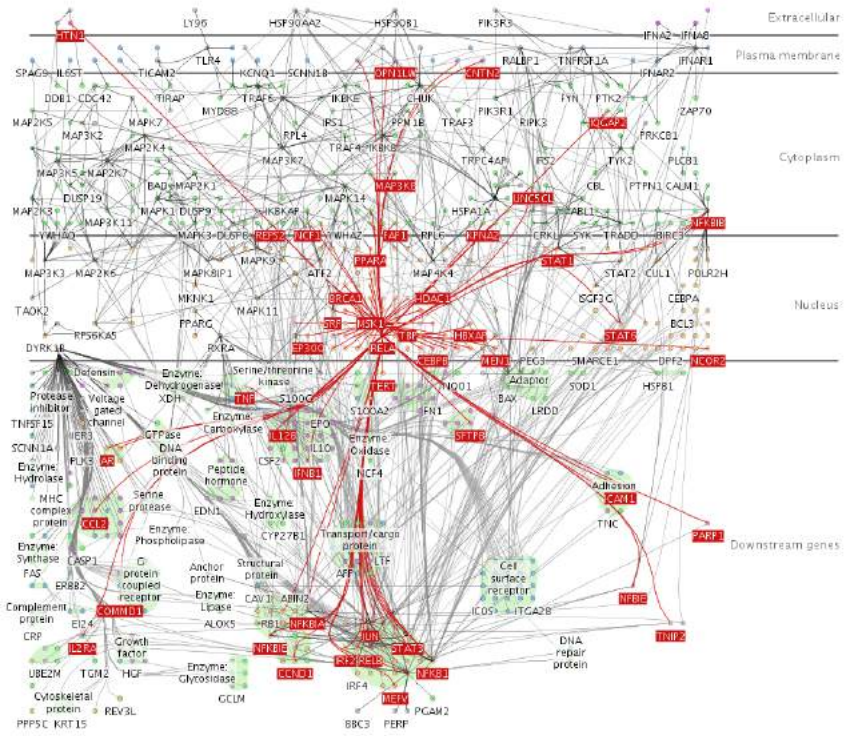


Figure 8.42: Cerebral dynamically creates a foreground visual layer of all nodes one topological hop away in the network from the node underneath the cursor. From [Barsky et al. 07], Figure 1.

a foreground layer that updates constantly as the user moves the cursor. When the cursor is directly over a node, the foreground layer shows its 1-hop neighborhood: all of the nodes in the network that are a single topological hop away from it, plus the links to them from the target node. The 1-hop neighborhood is visually emphasized with a distinctive fully saturated red to create a foreground layer that is visually distinct from the background layer, that has only low-saturation colors. The marks in the foreground layer also have larger linewidth.

8.7 History and Further Reading

Michael Friendly has extensively documented the rich history of visual representations of data [Friendly 08], with particular attention to statistical graphics and thematic cartography. William Playfair, who lived from 1759 to 1823, invented many of the fundamental statistical graphics that are still extensively used today: the time-series line chart, the bar chart, the pie chart, and the circle chart. Edmond Halley presented isolines in 1686 and contour plots in 1701. Thematic cartography, where statistical data is overlaid on geographic maps, blossomed in the 19th century. Choropleth maps, where shading is used to show a variable of interest, were introduced, as were dot maps and proportional symbol maps.

Cleveland has an excellent and extensive discussion of the use of many traditionaal statistical charts including bar charts, line charts, dotplots, and scatterplots [Cleveland 93b].

Slingsby, Dykes, and Wood have a thorough discussion of the use of partitioning and subdivision for attribute hierarchies [Slingsby et al. 09].

Parallel coordinates were independently proposed by both Inselberg [Inselberg and Dimsdale 90] and Wegman [Wegman 90].

Radial displays were surveyed by Draper, Livnat, and Riesenfeld [Draper et al. 09] and characterized through empirical user studies by Diel, Beck, and Burch [Diehl et al. 10].

The design of segmented colormaps has been extensively discussed in the cartographic literature. Brewer offers very readable color use guidelines [Brewer 99] derived from that community, in conjunction with the very useful ColorBrewer tool at `http:///www.colorbrewer2.org`. Rogowitz et al. offer useful guidelines on quantitative colormap creation and the problems with rainbow colormaps [Bergman et al. 95, Rogowitz and Treinish 96, Rogowitz and Treinish 98].

Wilkinson and Friendly [Wilkinson and Friendly 09] discuss the history of heatmaps, cluster heatmaps, and matrix reordering.

The connection and containment channels are heavily used in network and tree visualization. Hermann et al. surveyed early work in that area [Herman et al. 00], and more recent developments are covered by Landesberger et al. [von Landesberger et al. 10]. Treemaps were first proposed by Johnson and Shneiderman [Johnson and Shneiderman 91]. Kong, Heer, and Agrawala provide perceptual guidelines for creating treemaps and identified the data densities at which length-encoded bar charts become less effective than area-encoded treemaps [Kong et al. 10]. Schulz et al. analyzed the design space of non-nodelink approaches to tree drawing [Schulz et al. 11]. McGuffin and Robert analyzed the space efficiency of a wide variety of 2D graphical representations of trees [McGuffin and Robert 10], and provide design guidelines for their use.

Force-directed placement has been heavily studied; the book chapter by Brandes is a good algorithmically oriented overview [Brandes 01]. The GEM algorithm of Frick et al. is a good example of a sophisticated placement algorithm with built-in termination condition [Frick et al. 95]. Many multi-level layouts have been proposed, including sfdp [Hu 05], FM^3 [Hachul and Jünger 04], and TopoLayout [Archambault et al. 07]. Melancon discussed questions of edge density in node-link graph layouts [Melançon 06].

Zacks and Tversky analyze when to use bar charts vs. line charts [Zacks and Tversky 99]. The aspect ratio control technique of *banking to 45°* was first proposed by Cleveland [Cleveland et al. 88, Cleveland 93b], and extended to the an automatic multiscale framework by Heer and Agrawala [Heer and Agrawala 06].

Dense displays have extensively explored for many datatypes by Keim et al. [Keim 00]. An early dense display for text and source code was the SeeSoft system from Eick et al. [Eick et al. 92].

Ware has an extensive discussion of the complex and subtle issues in glyph design, including micro glyph texture fields in Chapter 5 and 6, and macro glyphs in Chapters 6 and 7 [Ware 04]. Ward has a good overview of glyph design in general [Ward 08] and the issues with glyph placement in particular [Ward 02]. Klippel et al. discuss empirical experiments on visual channel use in glyph design [Klippel et al. 09].

Linked highlighting was first proposed by Becker and Cleveland [Becker and Cleveland 87], who called it *brushing*.

Baldonado, Woodruff, and Kuchinsky provide an early set of concise guidelines on designing multiple-view systems [Baldonado et al. 00]. Roberts surveys the methods used when working with multiple views [Roberts 07]. Weaver created the Improvise toolkit that supports many forms of linking between views [Weaver 04], and has explored in depth the implications of designing coordinated multiple view systems [Weaver 10]. Wills has a detailed overview chapter about linked data views [Wills 08].

Javed and Elmqvist discuss the design space of composite visualization [Javed and Elmqvist 12].

Many previous authors have proposed ways to categorize visualization methods, and my framework was influenced by many of them. Wilkinson's grammar of graphics [Wilkinson 05], Keim's early KDD tutorial on visual techniques [Keim 97], Michael McGuffin's taxonomy of multidimensional multivariate visualization (personal communication) , the work of Wickham and collaborators on generalized pair plots [Emerson et al. 12] and product plots [Wickham and Hofmann 11], Heer and Shneiderman's taxonomy [Heer and Sheiderman 12].

9

Reducting Items and Attributes

While the previous chapter focused on how to create and combine views, this chapter focuses on the problem of how to reduce the amount of data that is shown at once within a view. Many datasets are so large that trying to show everything simultaneously would lead to incomprehensible clutter. Figure **??** shows the taxonomy of methods for reducing what is shown. Filtering and aggregation are the two major methods of reduction. These methods can be applied to both items and attributes; the word **element** will be used to refer to either items or attributes when methods that apply both are discussed. Filtering simply eliminates elements, whereas aggregation creates a single new element that stands in for multiple others that it replaces. Navigation can also be considered a method for reduction that is tied to spatial position, since it changes the set of elements that are visible. Zooming in to see fewer items in more detail can be done geometrically, matching the semantics of real-world motion. With semantic zooming, the way to draw items adapts on the fly based on the number of available pixels, so appearance can change dramatically rather than simply shrinking or enlarging. Finally, the focus+context methods reduce what is drawn in a way that combines filtering and aggregation.

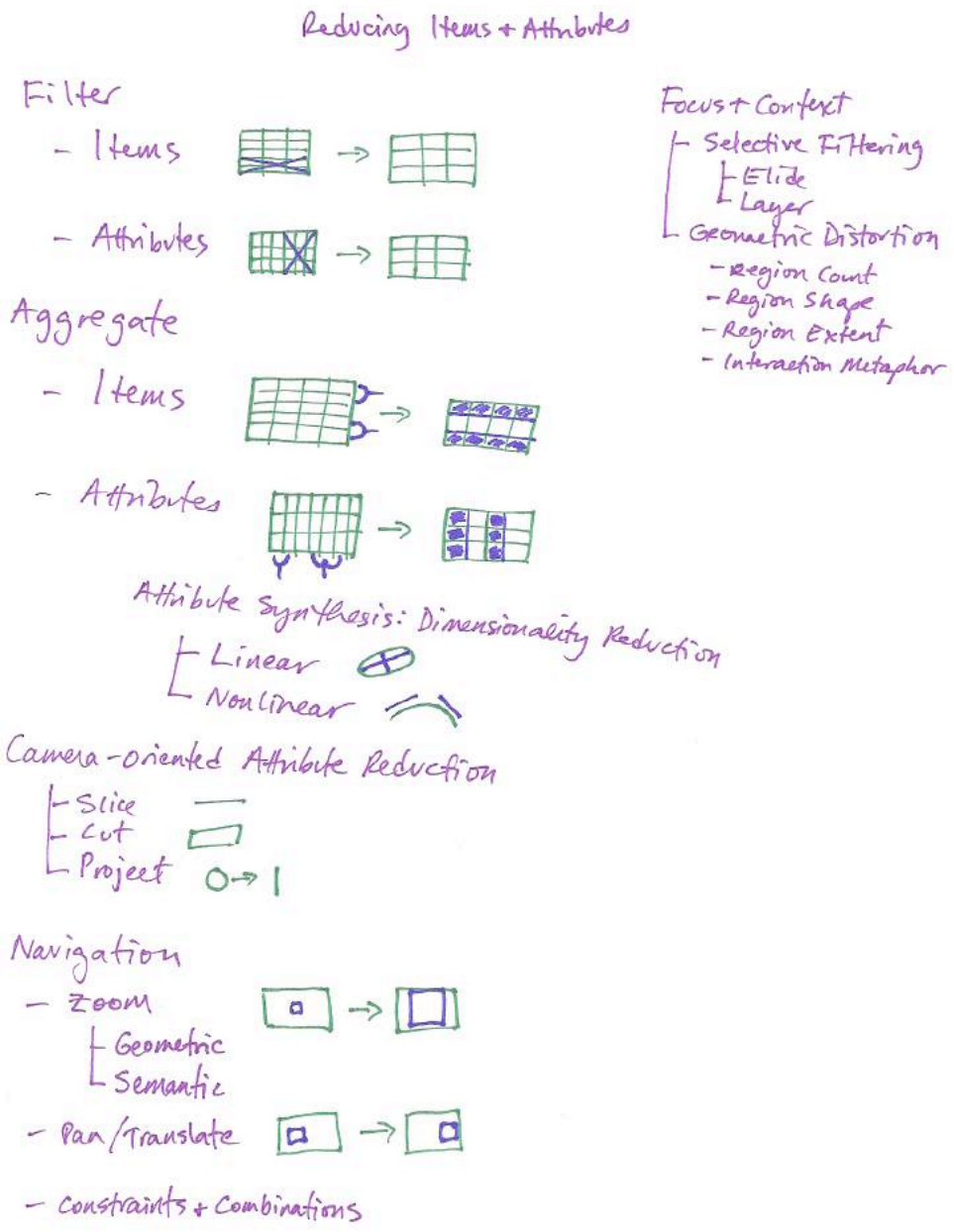


Figure 9.1: Framework of methods for reducing what to show.

9.1 Filtering

The most obvious method of item reduction is **filtering**, where some elements are simply eliminated. Filtering can be applied to both items and attributes. A straightforward approach to filtering is to allow the user to select one or more ranges of interest in one or more of the elements. The range might mean what to show, or what to leave out.

The idea of filtering is very straightforward; the complexity comes from designing a visualization system where filtering can be used to effectively explore a dataset. Consider the simple case of filtering the set of items according to their values for a single quantitative attribute. The goal is to select a range within it in terms of minimum and maximum numeric values, and eliminate the items whose values for that attribute fall outside of the range. From the programmer's point of view, a very simple way to support this functionality would be to simply have the user enter two numbers, a minimum and maximum value. From the user's point of view, this approach is very difficult to use: how on earth do they know what numbers to type? After they type, how do they know whether that choice is correct? When the goal is to support exploration of potentially unknown datasets, you cannot assume that the users already know these kinds of answers.

In an interactive visualization context, filtering is often accomplished through **dynamic queries** where there is a tightly coupled loop between visual encoding and interaction, so that the user can immediately see the results of the intervention. In this method, a display showing a visual encoding of the dataset is used in conjunction with controls that support direct interaction, so that the display updates immediately when the user changes a setting. Often these controls are standard graphical user interface widgets such as sliders, buttons, comboboxes, and text fields. Many extensions of off-the-shelf widgets have also been proposed to better support the needs of interactive visualization.

9.1.1 Filtering Items

In **item filtering**, the goal is to eliminate items based on their values with respect to specific attributes. Fewer items are shown, but the number of attributes shown does not change.

Example: FilmFinder

Figure 9.2 shows the FilmFinder system [Ahlberg and Shneiderman 94] for exploring a movie database. The dataset is a table with nine value

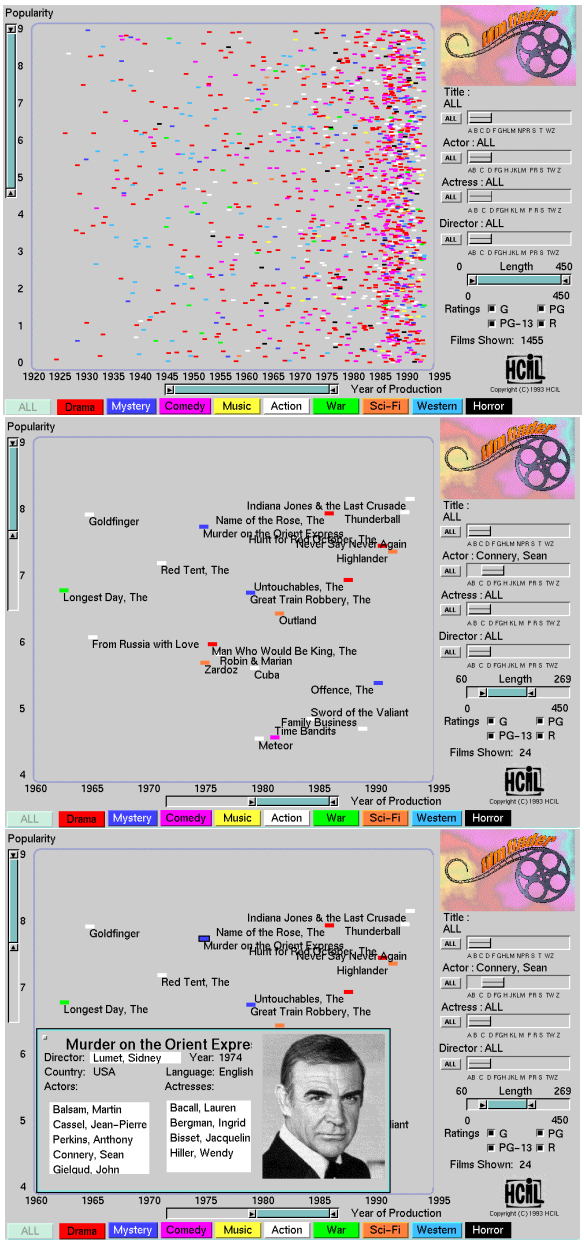


Figure 9.2: FilmFinder features tightly coupled interactive filtering, where the result of moving sliders and pressing buttons is immediately reflected in the visual encoding. (a) Exploration begins with an overview of all movies in the dataset. (b) Moving the actor slider to select Sean Connery filters out most of the other movies, leaving enough room to draw labels. (c) Clicking on the mark representing a movie brings up a detail view. From [Ahlberg and Shneiderman 94], Color Plates 1, 2, 3.

attributes: genre, year made, title, actors, actresses, directors, rating, popularity, and length. The visual encoding features an interactive scatterplot where the items are movies color coded by genre, with scatterplot axes of year made versus movie popularity. The interaction design features filtering, with immediate update of the visual display to filter out or add back items as sliders are moved and buttons are pressed. The visual encoding adapts to the number of items to display; the marks representing movies are automatically enlarged and labelled when enough of the dataset has been filtered away that there is enough room to do so. The system uses multiform overview-detail views, where clicking on any mark brings up a popup detail view with more information about that movie.

FilmFinder is specific example of the general dynamic queries approach, where browsing using tightly coupled visual encoding and interaction is an alternative to searching by running queries against a database. All of the items in a database are shown at the start of a session to provide an overview, and direct manipulation of interface widgets replaces reformulating new queries. This approach is well suited to browsing, as opposed to a search where the user must formulate an explicit query that might return far too many items, or none at all.

Figure 9.2 shows the use of two augmented slider types, a **dual slider** for movie length that allows the user to select both a minimum and maximum value, and several **alpha sliders** that are tuned for selection with text strings rather than numbers.

System	**FilmFinder**
Data Types	table 9 value attribs
View Comp.	scatterplot; detail view with text/images
Multiple Views	multiform, overview-detail
Reduction	item filtering

Standard widgets for filtering controls can be augmented by concisely visually encoding information about the dataset, but in the part of the screen normally thought of as the control panel rather than a separate display area. The idea is to do so while using no or minimal additional screen real estate, in order to create displays that have high information density. These augmented widgets are called **scented widgets** [Willett et al. 07], alluding to the idea of **information scent**: cues that help a searcher decide whether there is value in drilling down further into a particular information source, versus looking elsewhere [Pirolli 07]. Figure 9.3 shows several examples. One way to add information is by inserting a concise statistical

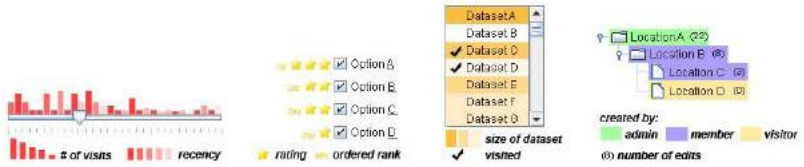


Figure 9.3: The scented widget technique adds visual encoding information directly to standard graphical widgets to make filtering possible with high information density displays. From [Willett et al. 07], Figure 2.

graphic, such as a bar or line chart. Another way is by inserting icons or text labels. A third way is to treat some part of the existing widget as a mark, and encode more information into that region using visual channels such as hue, saturation, and opacity.

The Improvise system shown in Figure 8.38 is another example of the use of filtering. The checkbox list view in the lower middle part of the screen is a simple filter controlling whether various geographic features are shown. The multiform microarray explorer in Figure 8.35 also supports several kinds of filtering, with simple range sliders and a more complex view that allows range selection on top of line charts of time-series data. The selected range in the line chart view acts as a filter for the scatterplot view.

9.1.2 Attribute Filtering

Attributes can also be filtered. With **attribute filtering**, the goal is to eliminate attributes rather than items; that is, to show the same number of items, but fewer attributes for each item.

Item filtering and attribute filtering can be combined, with the result of showing both fewer items and fewer attributes.

Example: DOSFA

Figure 9.4 shows an example of the Dimensional Ordering, Spacing, and Filtering Approach (DOSFA) technique [Yang et al. 03a]. As the name suggests, the technique features attribute filtering.[1] Figure 9.4 shows DOSFA on a dataset of 215 attributes representing word counts and 298 points representing documents in a collection of medical abstracts. DOSFA can be

[1] Vocabulary note: Many techniques for attribute filtering and aggregation use the alternative term *dimension* rather than *attribute* in their names.

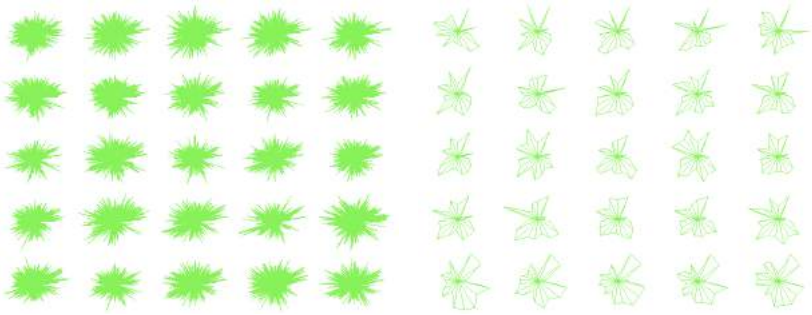


Figure 9.4: The DOSFA technique shown on star glyphs with medical records dataset of 215 dimensions and 298 points. (a) The full dataset is so dense that patterns cannot be seen. (b) After ordering on similarity and filtering on both similarity and importance, the star glyphs show structure. From [Yang et al. 03a], Figures 3a and 3d.

used with many visual encoding approaches; this figure shows it in use with star plots. In Figure 9.4(a) the plot axes are so densely packed that little structure can be seen. Figure 9.4(b) shows the plots after the dimensions are ordered by similarity and filtered by both similarity and importance thresholds. The filtered display does show clear visual patterns.

For more on star plots, see Section 8.1.6.3.

System	**DOSFA**
Data Types	table many value attribs
View Comp.	star plots
Multiple Views	small multiples with matrix alignment
Reduction	attribute filtering

Attribute filtering is often used in conjunction with attribute ordering.[2] If attributes can be ordered according to a derived attribute that measures the similarity between them, then all of the high-scoring ones or low-scoring ones can be easily filtered out interactively. A **similarity measure** for an attribute creates a quantitative or ordered value for each attribute based on all of the data item values for that attribute.[3] One approach is to calculate the **variance** of an attribute: to what extent the values within that attribute are similar or different from each other. There are many

[2]Vocabulary note: A synonym for attribute ordering is dimensional ordering.
[3]Vocabulary note: A synonym for similarity measure is *similarity metric*.

ways to calculate a similarity measure between attributes; some focus on global similarity, and others search for partial matches [Ankerst et al. 98].

9.2 Aggregation

The other major reduction method is **aggregation**, so that a group of elements is represented by a new derived element that stands in for the entire group. Elements are merged together with aggregation, as opposed to eliminated completely with filtering.

A simple example of aggregation is taking the average. For example, one visual mark representing many data items could be encoded with the average value of all items for the attribute of interest. Aggregation typically involves the use of a derived attribute, in this case the average. The four other standard aggregation operators are minimum, maximum, count, and sum; these will be familiar to people who have used database query languages such as SQL.

The challenge of aggregation is to avoid eliminating the interesting signals in the dataset in the process of summarization. Simple averaging is very likely to have this very effect, so it is not an adequate solution to the problem. In the cartographic literature, the problem of creating maps at different scales while retaining the important distinguishing characteristics has been extensively studied, under the name of *cartographic generalization* [Slocum et al. 08].

As with filtering, aggregation can be used for both items and attributes.

9.2.1 Item Aggregation

A simple use of aggregation is to create static views; it can also be combined with interaction for dynamically changing views.

Example: Histograms

Histograms are a way to show the distribution of items within an original attribute. Figure **??**(b) shows a histogram of the distribution of weights for all of the cats in a neighborhood, binned into 5-pound blocks. The range of the original attribute is partitioned into bins, and the number of items that fall into each bin is computed and saved as a derived ordered attribute. The visual encoding of a histogram is very similar to bar charts, with a line mark that uses spatial position in one direction and the bins distributed along an axis in the other direction. One difference is that histograms are normally shown without space between the bars to visually

Figure 9.5: A histogram aggregates an arbitrary number of items into a concise representation of their distribution.

imply continuity, whereas bar charts conversely have spaces between the bars to imply discretization. Despite their visual similarity, histograms are very different than bar charts. They do not show the original table directly; rather, they are an example of an aggregation method that shows a derived table that is more concise than the original dataset. The number of bins in the histogram can be chosen independently of the number of items in the dataset. The choice of bin size is crucial and tricky: a histogram can look quite different depending on the discretization chosen.

Technique	**histogram**
Data Types	table: 1 quantitative value attribute
Derived Data	derived table: 1 derived ordered key attrib: bin, 1 derived quantitative value attrib: item count per bin
Visual Encoding	rectilinear layout, line mark: aligned position: express derived value attrib. position: key attrib

Example: Continuous Scatterplots

Another example of aggregation is continuous scatterplots, where the problem of occlusion on scatterplots is solved by plotting an aggregate value at each pixel rather than drawing every single item as an individual point. Occlusion can be a major readability problem with scatterplots, because many dots could be overplotted on the same location. Size coding exacerbates the problem, as does the use of text labels. Continuous scatterplots use color coding at each pixel to indicate the density of overplotting, often

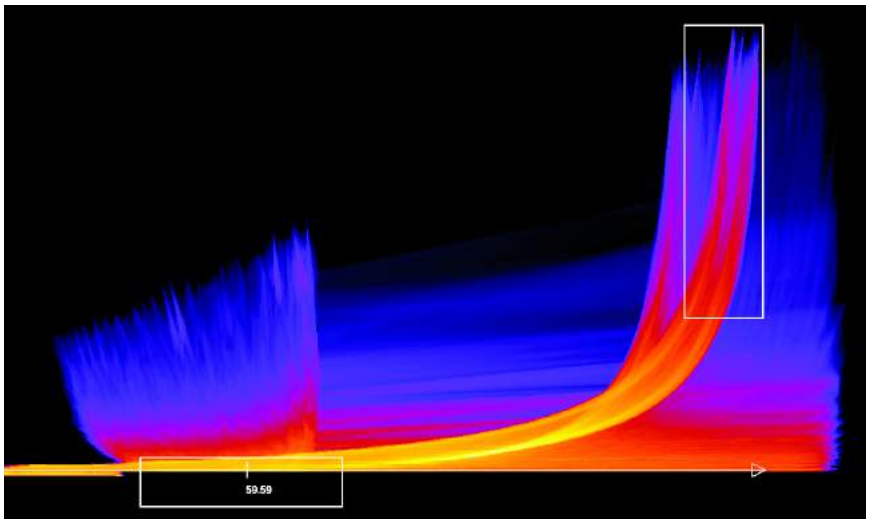


Figure 9.6: Continuous scatterplots use color to show the density at each location, solving the problem of occlusion from overplotting and allowing scalability to large datasets. From [Bachthaler and Weiskopf 08], Figure 9.

in conjunction with transparency. Conceptually, this approach uses a derived attribute, overplot density, which can be calculated after the layout is computed. Practically, many hardware acceleration techniques sidestep the need to do this calculation explicitly.

Figure 9.6 shows a continuous scatterplot of an tornado air flow dataset, with the magnitude of the velocity on the horizontal and the z-direction velocity on the vertical. The density is shown with a log-scale sequential colormap with monotonically increasing luminance. It starts with dark blues at the low end, continues with reds in the middle, and has yellows and whites at the high end.

Scatterplots began as a technique for discrete, categorical data. They have been generalized to a mathematical framework of density functions for continuous data, giving rise to continuous scatterplots in the 2D case and and continuous histograms in the 1D case [Bachthaler and Weiskopf 08]. Continuous scatterplots use a dense, spacefilling 2D matrix alignment, where each pixel is given a different color. Although the technique of continuous scatterplots has a similar name to the technique of scatterplots, analysis via the taxonomy of methods shows that the approach is in fact very different.

Technique	**continuous scatterplot**
Data Types	table: 2 quantitative value attributes
Derived Data	derived table: 2 ordered key attribs: (x,y) pixel locations, 1 quantitative attrib: overplot density
Visual Encoding	dense spacefilling 2D matrix alignment, sequential categorical hue + ordered luminance colormap
Reduction	aggregation

Example: Boxplot Charts

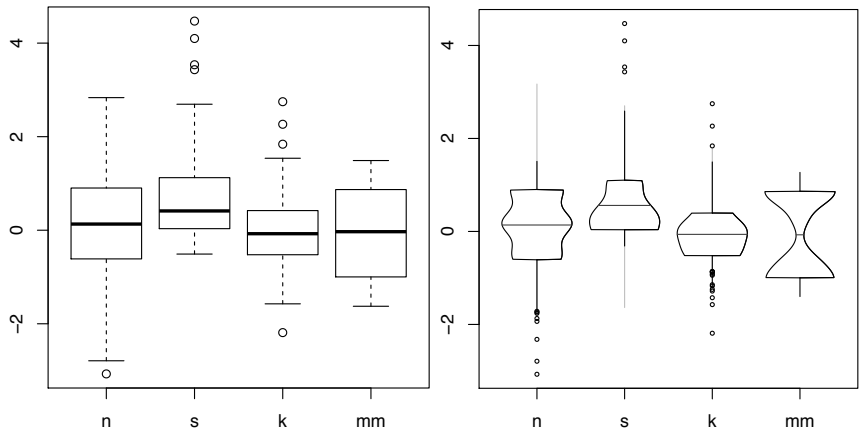


Figure 9.7: Boxplots present summary statistics for the distribution of a quantitative attribute, using five derived values. These plots illustrate four kinds of distributions: normal (n), skewed (s), peaked (p), and multimodal (mm). a) Standard box plots. b) Vase plots, which use horizontal spatial position to show density directly. From [Wickham and Stryjewski 12], Figure 5.

The **boxplot** is a visually concise way to visually show an aggregate statistical summary of all the values that occur within the distribution of a single quantitative attribute. It uses five derived variables carefully chosen

to provide information about the attribute's distribution: the median (50% point), the lower and upper quartiles (25% and 75% points), and the upper and lower fences (chosen values near the extremes, beyond which points should be counted as outliers). Figure 9.7a shows the visual encoding of these five numbers using a simple glyph that relies on vertical spatial position. The eponymous box stretches between the lower and upper quartiles, and has a horizontal line at the median. The **whiskers** are vertical lines that extend from the core box to the fences marked with horizontal lines. [4] Outliers beyond the range of the chosen fence cutoff are shown explicitly as discrete dots, just as in scatterplots or dotplots.

A boxplot is similar in spirit to the bar in a bar chart in that only a single spatial axis is used to visually encode data, but boxplots show five numbers through the use of a glyph rather than the single number encoded by the linear mark in a bar chart. A boxplot chart features multiple boxplots within a single shared frame to contrast different attribute distributions, just as bar charts show multiple bars along the second axis. In Figure 9.7, the quantitative value attribute is mapped to the vertical axis and the categorical key attribute to the horizontal one.

The boxplot can be considered as an item reduction technique that provides an aggregate view of a distribution through the use of derived data. Boxplots are highly scalable in terms of aggregating the target quantitative attribute from what could be an arbitrarily large set of values down to five numbers; for example, it could easily handle from thousands to millions of values within that attribute. The spatial encoding of these five numbers along the central axis requires only a moderate amount of screen space, since we have high visual acuity with spatial position. Each boxplot requires only a very small amount of screen space along the secondary axis, leading to a high level of scalability in terms of the number of categorical values that can be accommodated in a boxplot chart; roughly hundreds.

Boxplots directly show the **spread**, namely the degree of dispersion, with the extent of the box. They show the **skew** of the distribution compared to a normal distribution with the peak at the center by the asymmetry between the top and bottom sections of the box. Standard boxplots are designed to handle **unimodal** data, where there is only value that occurs the most frequently. There are many variants of boxplots that augment the basic visual encoding with more information. Figure 9.7b shows a variable-width variant called the **vase plot** that uses an additional spatial dimension within the glyph by altering the width of the central box according to the density, allowing a visual check if the distribution is instead **multimodal**, with multiple peaks. The variable-width variants require more screen space along the secondary axis than the simpler version, in an example of

[4] Vocabulary note: Boxplots are also known as box-and-whisker diagrams.

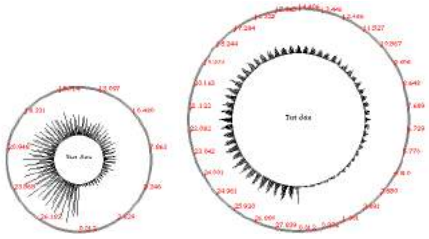
the classic cost/benefit tradeoff where conveying more information requires more room.

Technique	**boxplot chart**
Data Types	table: many quantitative value attributes
Derived Data	derived table: key attrib: names of original attribs quantitative attrib: 5 numbers representing distribution for each orig attrib
Visual Encoding	glyph to express value stats with spatial position, spatial position: derived key attrib
Reduction	aggregation
Abstract Tasks	characterize distribution find outliers, extremes, averages detect skew
Scalability	items: unlimited levels: dozens for key attrib

Many of the interesting uses of aggregation in visualization involve dynamically changing sets: the mapping between individual items and the aggregated visual mark changes on the fly. The simple case is to allow the user to explicitly request aggregation and deaggregation of item sets. More sophisticated approaches do these operations automatically as a result of higher-level interaction and navigation, usually based on spatial proximity.

Example: SolarPlot

Figure 9.8 shows the example of SolarPlot, a radial histogram with an interactively controllable aggregation level [Chuah 98]. The user directly manipulates the size of the base circle that is the radial axis of the chart. This change of radius indirectly changes the number of available histogram bins, and thus the aggregation level. Like all histograms, the SolarPlot aggregation operator is `count`: the height of the bar represents the number of items in the set. The dataset shown is ticket sales over time, starting from the base of the circle and progressing counterclockwise to cover 30 years in total. The small circle in Figure 9.8a is heavily aggregated. It does show an increase in ticket sales over the years. The larger circle in Figure 9.8b is less aggregated, and seasonal patterns within each year can be distinguished.



(a) (b)

Figure 9.8: The SolarPlot circular histogram provides indirect control of aggregation level by changing the circle size. a) The small circle shows the increase in ticket sales over time. b) Enlarging the circle shows seasonal patterns in addition to the gradual increase. From [Chuah 98], Figures 1 and 2.

Technique	**SolarPlot**
Data Types	table: 1 quantitative attribute
Derived Data	derived table: 1 derived ordered key attrib: bin, 1 derived quantitative value attrib: item count per bin number of bins interactively controlled
Visual Encoding	radial layout; line marks; line length: express derived value attrib, angle: key attrib
Reduction	item aggregation
Scalability	original items: unlimited derived bins: proportional to screen space allocated

The general method of **hierarchical aggregation** is to construct the derived data of a hierarchical clustering of items in the original dataset, and allow the user to interactively control the level of detail to show based on this hierarchy. There are many specific examples of techniques that use variants of this method.

The cluster-calendar system in Figure 7.4 is one example of hierarchical aggregation. Another is hierarchical parallel coordinates.

Example: Hierarchical parallel coordinates

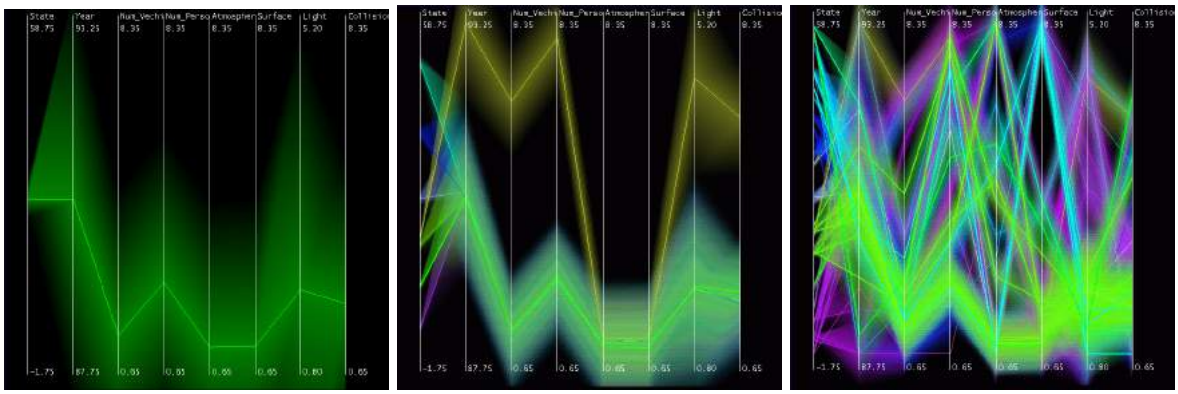


Figure 9.9: Hierarchical parallel coordinates show high-dimensional data using spatial position at multiple levels of detail. From [Fua et al. 99], Figure 4.

The technique of **hierarchical parallel coordinates** [Fua et al. 99] uses interactively controlled aggregation as a method to increase the scalability of the basic parallel coordianates visual encoding to hundreds of thousands of items. The dataset is transformed by computing derived data: a hierarchical clustering of the items. Several statistics about each cluster are computed, including the number of points it contains; the mean, minimum, and maximum values; and the depth in the hierarchy. A cluster is represented by a band of varying width and opacity, where the mean is in the middle and width at each axis depends on the minimum and maximum item values for that attribute within the cluster. Thus, in the limit, a cluster of a single item is shown by a single line, just as with the original technique. The cluster bands are colored according to their proximity in the cluster hierarchy, so that clusters far away from each other have very different colors.

The level of detail displayed at a global level for the entire dataset can be interactively controlled by the user using a single slider. The parameter controlled by that slider is again a derived variable that varies the aggregate level of detail shown in a smooth and continuous way. Figure 9.9 shows a dataset with 8 attributes and 230,000 items at different levels of detail. On the left is the highest-level overview showing the single top-level cluster, with very broad bands of green. In the middle is a mid-level view showing several clusters, where the extents of the tan cluster is clearly distinguishable from the now-smaller green one. On the right is a more detailed view

with dozens of clusters that have tighter bands; the proximity-based coloring mitigates the effect of occlusion.

Technique	**hierarchical parallel coordinates**
Data Types	table
Derived Data	cluster hierarchy atop original table of items
Visual Encoding	color channel to show proximity in cluster hierarchy
Interaction	change global level of detail
Scalability	items: 10K - 100K clusters: dozens

9.2.2 Attribute Aggregation

Just as attributes can also be filtered, attributes can also be aggregated. A new attribute is synthesized to take the place of many original attributes. A simple approach to aggregating attributes is to group them by some kind of similarity measure, and then calculate an average across that similar set.

In the general method typically called **dimensionality reduction**, the goal is to preserve the meaningful structure of a dataset while using fewer attributes to represent the items.[5] The rationale that a small set of new synthetic attributes might be able to faithfully represent most of the structure or variance in the dataset is the assumption that there is hidden structure and significant redundancy in the original dataset, because the underlying latent variables could not be measured directly.

For more on the task of finding latent variables, see Section **??**.

The two main methods for dimensionality reduction are linear and nonlinear mappings.

9.2.2.1 Linear Mappings

The most common approach to dimensionality reduction is **Principal Component Analysis** (PCA), a technique first proposed over one hundred years ago. This technique is a linear mapping method, where the new synthetic dimensions are a weighted combination of the original ones. With PCA, the synthetic dimensions are created in order of variance: the first new dimensions captures as much of the variance in the original dataset as possible, with each successive dimension expressing less of the variance. For some datasets this approach yields excellent results, as shown in Figure 9.10. However, some datasets are not easily expressed as linear combinations of dimensions, for example when the underlying

[5] Vocabulary note: Attribute aggregation, attribute synthesis, and dimensionality reduction are all synonyms. The latter name is common in the literature. I use attribute aggregation as a name to show where this method fits into the taxonomy of the book; it is not a typical usage by other authors.

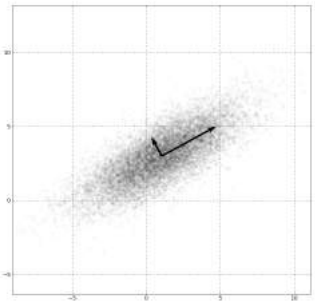


Figure 9.10: PCA is the most common dimensionality reduction technique. The two synthetic dimensions shown here as arrows express all of the variance in this Gaussian blob, with the first dimension capturing the largest amount. Image from http://en.wikipedia.org/wiki/File:GaussianScatterPCA.png

structures are curves rather than straight lines. In these cases, nonlinear dimensionsionality reduction is a more appropriate method.

9.2.2.2 Nonlinear Mappings With nonlinear dimensionality reduction, the new dimensions cannot be expressed in terms of a straightforward combination of the original ones. Many nonlinear approaches have been proposed; **Multidimensional Scaling** (MDS) is the name for one family of approaches, where the goal is to minimize the differences in distances between points in the high-dimensional space vs. the new lower-dimensional space. Figure **??** shows the results of the Glimmer MDS technique used to view a collection of documents in a single scatterplot.

9.3 Navigation

The term **navigation** refers to changing the point of view from which things are drawn. The underlying metaphor is a virtual camera located in a particular spot and aimed in a particular direction. When that camera **viewpoint** is changed, the set of items visible in the camera **frame** also changes.

Navigation can be broken down into three components. **Zooming** moves the camera closer to or farther from the plane. Zooming the camera in close will show fewer items, which appear to be larger. Zooming out so that the camera is far away will show more items, and they will appear

smaller. **Panning** the camera moves it parallel to the plane of the image, either up and down or from side to side. In 3D navigation, the general term **translating** is more commonly used for any motion that changes camera position. [6] **Rotating** spins the camera around its own axis. Rotation is rare in two-dimensional navigation, but is much more important with 3D motion.

Since changing the viewpoint changes the visible set of items, the outcome of navigation could be considered as some combination of filtering and aggregation. Zooming in or panning with a zoomed-in camera filters based on the spatial layout; zooming out can act as aggregation that creates an overview.

9.3.1 Zooming

9.3.1.1 Geometric Zooming. **Geometric zooming** is intuitive because it corresponds almost exactly with our real-world experience of walking closer to objects, or grabbing objects with our hands and moving them closer to our eyes. In the 2D case, the metaphor is moving a piece of paper closer to or further from our eyes, while keeping it parallel to our face.

TODO: need more here, incl example

9.3.1.2 Semantic Zooming. With geometric zooming, the fundamental appearance of objects is fixed and moving the viewpoint simply changes the size at which they are drawn in image space. An alternative, nongeometric approach to motion does not correspond to simply moving a virtual camera. With **semantic zooming**, the representation of the object adapts to the number of pixels available in the image-space region occupied by the object. The visual appearance of an object can change subtly or dramatically at different scales.

For instance, an abstract visual representation of a text file might always be a rectangular box, but its contents would change considerably. When the box was small, it would contain only a short text label, with the name of the file. A medium-sized box could contain the full document title. A large box could accommodate a multi-line summary of the file contents. If the representation did not change, the multiple lines of text would simply be illegible when the box was small.

[6] Vocabulary note: In this book, pan and zoom are used to mean translating the camera parallel to and perpendicular to the image plane; this loose sense matches the usage in the information visualization literature. In cinematography, these motions are called trucking and dollying. Trucking, where the camera is translated parallel to the image plane, is distinguished from panning, where the camera stays in the same location and turns to point in a different direction. Dollying, where the camera is translated closer to the scene, is distinguished from zooming by changing the focal length of the camera lens.

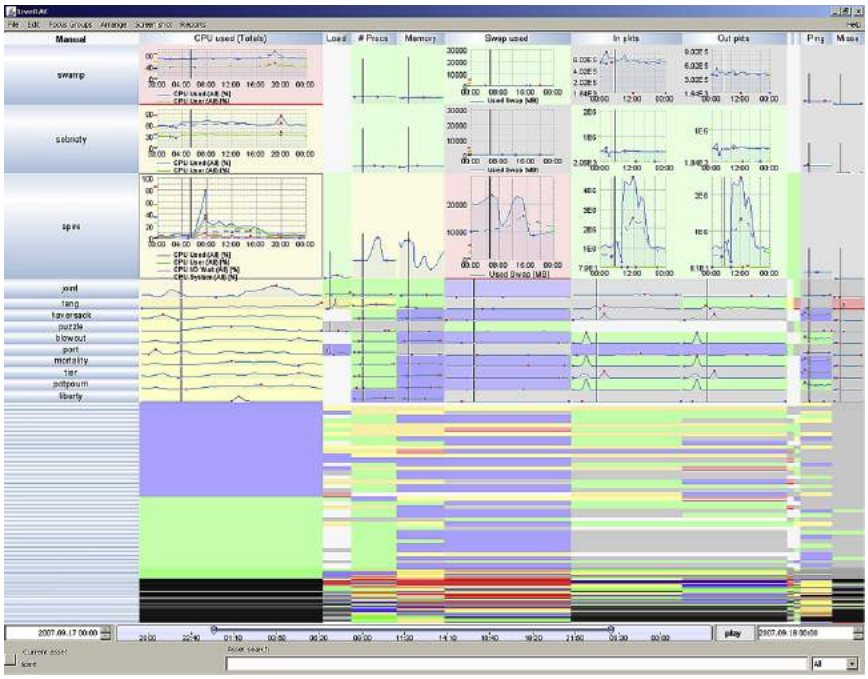


Figure 9.11: LiveRAC uses semantic zooming within a stretchable grid of time-series line charts. From [McLachlan et al. 08], Figure 2b.

Figure 9.11 shows an example from the LiveRAC system for analyzing large collections of time-series data, in this case resource usage for large-scale system administration [McLachlan et al. 08]. Line charts in a very large grid use semantic zooming, automatically adapting to the amount of space available as rows and columns are stretched and squished. When the available box is tiny, then only a single categorical variable is shown with color coding. Slightly larger boxes also use **sparklines**, very concise line charts with dots marking the minimum and maximum values for that time period. As the box size passes thresholds, axes are drawn and multiple line charts are superimposed. The stretch-and-squish navigation metaphor is an example of a focus+context method, discussed further in Section 9.5.

Figure 9.12 shows a more subtle example of semantic zooming from the Constellation system [Munzner et al. 99]. What changes during the zoom is the relative allocation of space for the first word in the definition versus the rest of the words. When the camera is zoomed out, the first word gets the lion's share of the vertical space; when zoomed in, the words are all in the same font. This navigation technique is also an example of constrained

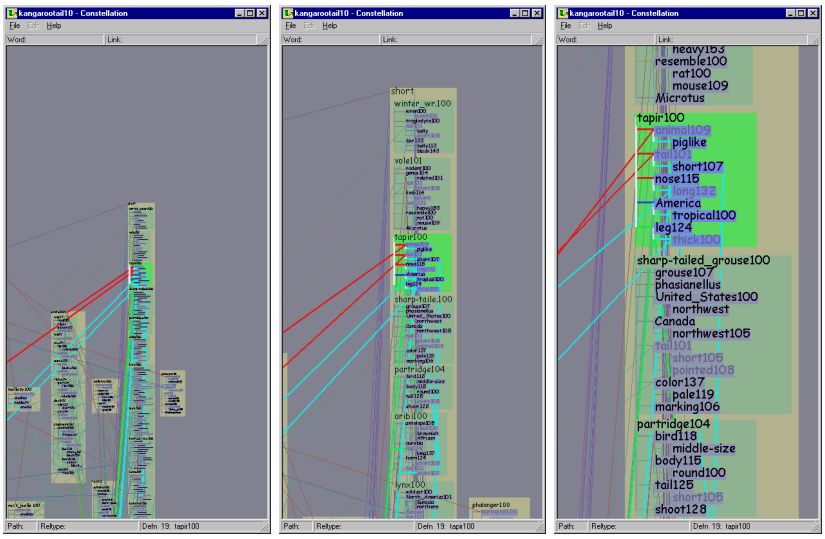


Figure 9.12: Constellation uses a more subtle form of semantic zooming, where the space allocated for the first word vs. the rest of the definition changes according to the zoom level. From [Munzner 00], Figure 5.19.

navigation, as discussed in Section 9.3.3.2.

9.3.2 Panning

TODO

9.3.3 Combined and Constrained Navigation

9.3.3.1 Combined Navigation Despite the intuitive nature of panning and zooming, finding an efficient way to navigate from one virtual viewpoint to another can still be a complex problem. To analyze this problem, it is useful to distinguish between considering location in **world space**, where distances between objects do not change when the camera is moved, and **image space**, where location is measured on the image itself. [7] In image space, distances between objects vary dramatically based on camera viewpoint. Zooming out means that a fixed distance in image space corresponds to a much larger distance in world space than when the camera is zoomed in.

[7] Vocabulary note: *Screen space* is a synonym for *image space* used in the computer graphics literature; in this book it is often used to mean the number of pixels available or used on a display.

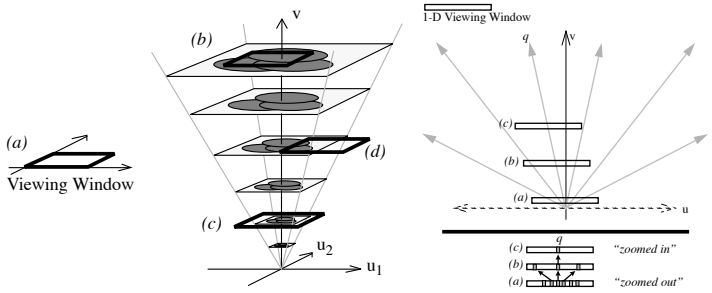


Figure 9.13: Space-scale diagrams support reasoning about navigation trajectories. The vertical axis represents zooming. (a) 2D camera. (b) 1D camera. From [Furnas and Bederson 95], Figures 2 and 5.

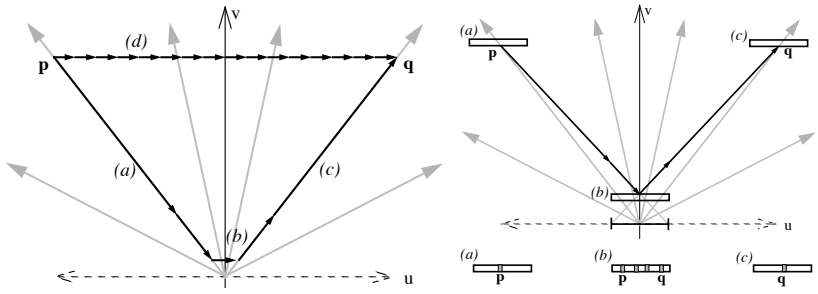


Figure 9.14: Space-scale diagrams about the pan-zoom problem. (a) Panning when zoomed in is slower than when zoomed out, for a given step size of optical flow. (b) One efficient trajectory relies mostly on zooming and minimizes panning. From [Furnas and Bederson 95], Figures 8 and 10.

Space-scale diagrams are a useful aid to thinking about navigation and trajectories [Furnas and Bederson 95]. Figure 9.13(a) introduces the idea. The bold rectangle represents the image frame seen through a 2D camera, the vertical v axis corresponds zooming in, and moving horizontally in the u_1, v_1 plane corresponds to panning. Close to the origin, when the camera is zoomed out, the entire world fits within the frame. Moving up the vertical axis, where the camera zooms in, results in a frame that only shows part of the world. Figure 9.13(b) shows with the simplified 1D version of the diagram, where there is still a vertical v zoom axis but only one horizontal u axis. An image frame is now a line segment rather than a rectangle, and a visible object is a point on that segment. The rays from the original correspond to the visibility of an object within the world: when a ray intersects the frame, it is visible.

The zoom level of the camera controls the speed at which it can pan, in order to have the **optical flow** – how quickly objects move through the frame across a series of images – remain roughly constant throughout the navigation. Consider the case of horizontal panning, where the goal is to have the object's position within the frame move by no more than 10% of the image width on each frame of an animated transition, so that the action is easy to follow. If the camera pans so quickly that no objects from one frame are still visible in the next, the motion would be very difficult to understand. Figure 9.14(a) illustrates the fact that panning with a zoomed-in camera is much slower than with a zoomed-out camera, for a given speed of optical flow.

Efficient navigation thus typically involves the combination of panning and zooming, rather than simply panning. One simple strategy to move from an old viewpoint to a new one while maintaining the viewer's sense of orientation is to maximize zooming and minimize panning: zoom out far enough that all objects from both viewpoints are visible within the frame, and then zoom in to center the new object within the frame. This trajectory is a combination of pure zooming and a minimal amount of panning. Figure 9.14(b) illustrates it.

Example: OrthoZoom

TODO .....

9.3.3.2 Unconstrained vs. Constrained Navigation. With **unconstrained navigation**, the camera can move anywhere at all. It is very easy to implement for programmers who have learned the framework of standard computer graphics. It does have the conceptual appeal of allowing users access to a clearly defined mathematical category of transformations. However, users often have difficulty in figuring out how to achieve a desired viewpoint with completely free navigation, especially in three dimensions. Navigating with the six degrees of freedom available in 3D space is not a skill that most people have acquired; pilots of airplanes and helicopters are the exception rather than the rule. To make matters worse, when interacting with virtual objects drawn with computer graphics it is easy to inadvertently move the viewpoint inside what are intended to be solid objects. The physics of reality prevent interpenetration between our heads and objects like walls, floors, or basketballs. With computer graphics, collision detection is a computationally intensive process that is separate from drawing and does not come for free. Another problem is ending up with the camera pointed at nothing at all, so that the entire frame is empty and it is not obvious where to go in order to see something interesting. This

problem is especially common in 3D settings, but can still be a problem in 2D settings with unconstrainee panning and zooming. A button to reset the view is of course a good idea, but is far from a complete solution.

In contrast, **constrained navigation** methods have some kind of limit on the possible motion of the camera. One of the simplest approaches in a 2D setting is to limit the zoom range, so that the camera cannot zoom out much further than the distance where the entire drawn scene is visible, or zoom in much further than the size of the smallest object.

Many approaches allow the user to easily select some item of interest, and then the system automatically calculates a smooth camera trajectory for an animated transition from the current viewpoint to a useful new place where that particular item is easy to see, to maintain context. Figure 9.12 shows an example where the final viewpoint is be zoomed in enough that labels are clearly visible, zoomed out far enough that the full extent of the object fits in the frame, and panned so that the object is within in the frame. In this case, a click anywhere within the green box enclosing a set of words is interpreted as a request to frame the entire complex object in the view, not a request to simply zoom into that exact point in space. In 3D, a useful additional constraint is that final orientation looks straight down at the object along a vector perpendicular to it, with the up vector of the camera consistent with the vertical axis of the object.

Constrained navigation is particularly powerful when combined with linked navigation between multiple views. For example, a tabular or list view could be sorted by many possible attributes, including simple alphabetical ordering. These views support search search well, where the name of an interesting item is known in advance. Clicking on that object in the list view could trigger navigation to ensure that the object of interest is framed within another, more elaborate view designed for browsing local neighborhoods where the items are laid out with a very different encoding for spatial position.

Many systems support both types of navigation. For example, constrained navigation may be designed to provide shortcuts for common use, with unconstrained navigation as backup for the uncommon cases where the user needs completely general controls.

9.4 Camera-Oriented Attribute Reduction

One set of methods for reducing the number of attributes is based on geometric intuitions about manipulating a virtual camera. Given this foundation, the term *dimensions* rather than *attributes* is used in this section. Although these methods are inspired by ways to manipulate a virtual camera in a 3D scene, they also generalize to higher-dimensional spaces. These

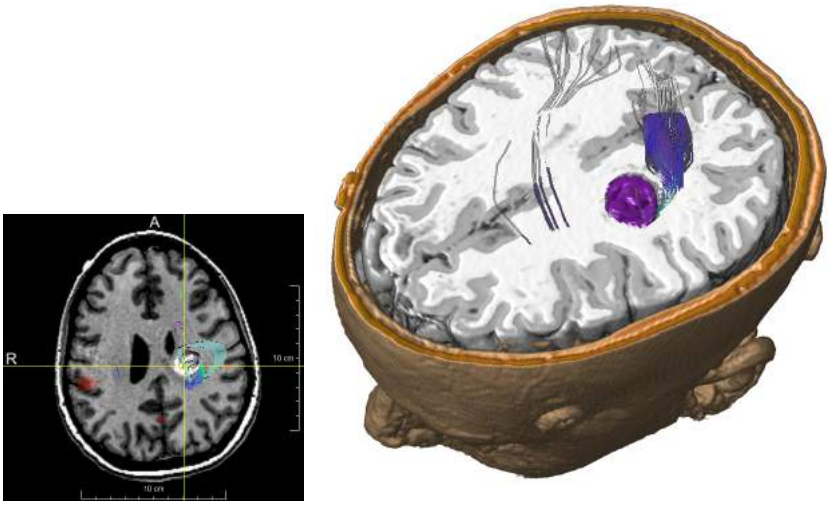


Figure 9.15: Slicing and cutting are camera-inspired methods for data reduction based on dimensions. (a) Axis-aligned slice. (b) Axis-aligned cut. From [Rieder et al. 08], Figures 7c, Teaser.

approaches are typically used for spatial field datasets, but can sometimes be usefully applied to abstract data.

9.4.1 Slicing and Cutting

With **slicing**, a single value is chosen from the dimension to eliminate, and only the items matching that value for the dimension are extracted to include in the lower-dimensional slice. Slicing is a particularly intuitive metaphor when reducing spatial data from 3D to 2D.

Figure 9.15(a) shows a classic example of inspecting a 3D scan of a human head by interactively changing the slice location to different heights along the skull. That is, the value chosen is a specific height in the third dimension, and the information extracted is the 2D position and color for each point matching that height. Here the slicing plane is aligned to one the major axes of the human body, which are also the original dimensions used to gather the data. One benefit of axis-aligned slices is that they may correspond to views that are familiar to the viewer: for example, medical students learn anatomy in terms of cross-sections along the standard coronal, sagittal, and horizontal planes.

Closely related to slicing is **cutting**, where all items with values above or below some chosen value for the particular dimension are shown. Fig-

ure 9.15(b) shows an example, again with medical image data of a human brain.

It is also possible to slice along a plane at an arbitrary orientation that does not match the original axes, exploiting the same intuition. Mathematically, the operation is more complex, because finding the values to extract may require a significant computation rather than a simple look-up.[8]

Slicing is not restricted to a change from 3D to 2D. The same idea holds in the more general case, where the slicing plane could a **hyperplane**; that is, the higher-dimensional version of a plane. In the case of reducing to just one dimension, the hyperplane would simply be a line. Slicing can be used to eliminate multiple dimensions at once, for example by reducing from 6D to 1D.

Example: HyperSlice

TODO .....

9.4.2 Projection

With **projection**, all items are shown, but without the information for specific dimensions chosen to exclude. For instance, the shadow of an object on a wall or floor is a projection from 3D to 2D. There are many types of projection; some retain partial information about the removed dimensions, while others eliminate that completely.

A very simple form of projection is **orthographic projection**: for each item the values for excluded dimensions are simply dropped, and no information about the eliminated dimensions is retained. In the case of orthographic projection from 3D to 2D, all information about depth is lost.[9] Projections are often used via multiple views, where there can be a view for each possible combination of dimensions. For instance, standard architectural blueprints show the three possible views for axis-aligned 2D views of a 3D XYZ scene: a floor plan for the XY dimensions, a side view for YZ, and a front view for XZ.

A more complex yet more familiar form of projection is the standard **perspective projection**, which also projects from 3D to 2D. This transformation happens automatically in our eyes, and is mathematically modelled in the perspective transformation used in the computer graphics virtual camera. While a great deal of information about depth is lost, some is

[8]Vocabulary note: Using the vocabulary of signal processing, care must taken to minimize sampling and interpolation artifacts.

[9]Vocabulary note: The term *dimensional filtering*, common when working with abstract data, is essentially a synonym for *orthographic projection*, more commonly used when working with spatial data.

retained through the perspective foreshortening effect where distant objects are closer together on the image plane than nearby objects would be.

Many **map projections** from the surface of the earth to 2D maps have been proposed by cartographers, including the well-known Mercator projection. Because these projections go from a curved to a flat space, some distortion of either angles or areas is inevitable. Although these might seem to be projections from 3D to 2D, more precisely they are projections from a 2D curved space to a 2D flat space. Thus, they do not cleanly fit the intuition of projection as used for dimensionality reduction, since there is no change of dimensionality.

9.5 Focus+Context

The **focus+context** method is a way to integrate detailed information about a selected focus set with contextual information providing overview information in a single view. In other words, focus and context information is **embedded** together, rather than shown in two **separate** views as with multiple-view methods. It uses both low-level item reduction methods, aggregation and filtering, simultaneously. A very large family of specific techniques that use some form of focus+context embedding have been proposed.[10]

The goal of focus+context embedded techniques is to mitigate the potential for disorientation that comes with purely geometric zooming. With realistic camera motion, only a small part of world space is visible in the image when the camera is zoomed in to see details for a small region, as discussed in Section 9.3.1.1. With geometric navigation and a single view, as in temporally multiplexed overviewing, the only way to maintain orientation is to internally remember one's own navigation history. Focus+context approaches attempt to support orientation by providing contextual information intended to act as recognizeable landmarks, using external memory to reduce internal cognitive load.

Focus+context approaches are thus an example of non-literal navigation, in the same spirit as semantic zooming. The shared idea with all of them is that a subset of the dataset items are interactively chosen by the user to be the focus, and are drawn in detail. The visual encoding also includes information about some or all of the rest of the dataset shown for context, integrated into and embedded within the same view that shows the

[10]Vocabulary note: Many names are essentially synonyms for or special cases of focus+context: nonlinear magnification, fisheye views, generalized fisheye views, hyperbolic views, distortion-oriented presentations, distortion viewing, detail in context, rubber sheet navigation, stretch and squish navigation, pliable surfaces, polyfocal projections, bifocal displays.

focus items. Some techniques achieve this selective presentation without the use of geometric distortion, but many use carefully chosen distortion to combine magnified focus regions and minified context regions into a unified view.

This embedding method cannot be fully understood when considered purely from the visual encoding point of view *or* purely from the interaction point of view; it is fundamentally a synthesis of both. The key idea of focus+context is that the focus set changes dynamically as the user interacts with the system, and thus the visual representation also changes dynamically. Many of the techniques involve indirect control, where the focus set is inferred via the combination of the user's navigation choices and the inherent structure of the dataset.

9.5.1 Selective Filtering

One general framework for reasoning about focus and context set selection is with a **degree of interest (DOI)** function: $DOI = I(x) - D(x, y)$. In this equation, I is an interest function; D is the distance, either semantic or spatial; x is the location of the data element; y is the current focus [Furnas 86]. There could be only one focus point, or multiple independent foci. These functions typically exploit knowledge about dataset structure, especially hierarchical relationships. For example, if a few subsections of a document were selected to be the foci, then a good context would be their enclosing sections. The DOI function can be thought of as a continuous function does not explicitly distinguish between focus items, context items, and those to ignore or elide; those interpretations are made by algorithms that use the function, often based on threshold values.

9.5.1.1 Elision and Aggregation

One approach to selective filtering is to elide items from the view whose DOI functions are below some threshold; **elision** is the act of omitting items. A very straightforward approach is to simply hide all of these items, but most techniques try to provide some sort of visual feedback about the elided information either directly or indirectly.

The DOITrees Revisited system shown in Figure 9.16 uses multiple foci to show an elided version of a 600,000 node tree. The shaded triangles provide an aggregate representation showing the size of the elided subtrees. The context in which to show them is computed using tree traversal from the many focus nodes up towards their common ancestors and the tree root. In this case, distance is computed topologically based on hops through the tree, rather than geometrically through Euclidean space. The focus nodes can be chosen explicitly by clicking, or indirectly through searching.

9.5.1.2 Layering

Another approach to integrating focus and context is the use of layering. The Toolglass and Magic Lenses system shown in Fig-

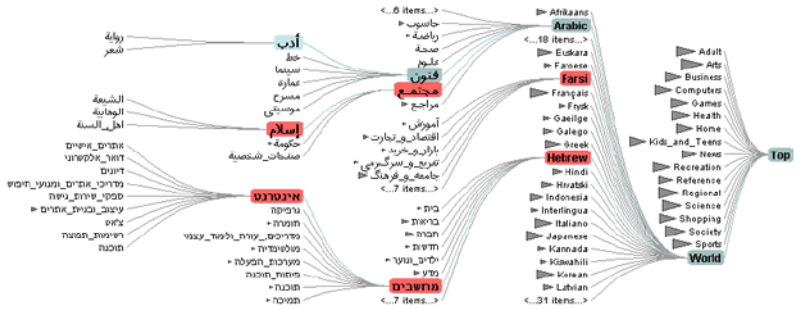


Figure 9.16: DOITrees Revisited uses selective filtering, elision, and aggregate representations to show multiple focus nodes within context in a 600,000 node tree. From [Heer and Card 04], Figure 1.

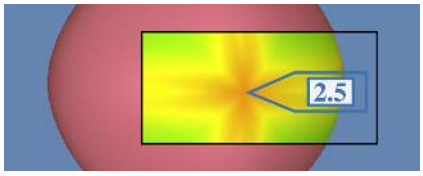


Figure 9.17: Toolglass and Magic Lenses provides focus and context through layers: the see-through lens color-codes the sphere with Gaussian curvature information. From [Bier et al. 93], Figure 12.

ure 9.17 uses a see-through lens to show color-coded Gaussian curvature in a foreground layer, atop the background layer consisting of the rest of the scene. This specific use of dynamic layering is related to the more general use of visual layering within a single view discussed in Section 8.6. In this case, the layer has a clearly defined local extent rather than stretching globally across the entire image. Within the lens details are shown, and the unchanged remainder of the image provides context. The lens layer occludes the region beneath it.

9.5.2 Geometric Distortion

In addition to using selective filtering, many focus+context techniques solve the problem of integrating focus and context into a single view by geometric warping of the contextual regions to make room for the details in the focus regions.

There are several major choices of methods that underlie all geometric distortion techniques. One choice is the number of focus regions: is there

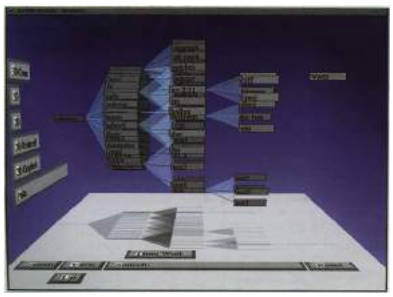


Figure 9.18: The Cone Tree system explicitly uses 3D perspective for focus+context. .

only a single region of focus, or does the technique allow multiple foci? Another choice is the shape of the focus: is it a radial, rectangular, or a completely arbitrary shape? A third choice is the extent of the focus: is it global across the entire image, or constrained to just a local region? A fourth choice is the interaction metaphor. One possibility is mathematical projection. Another is moveable lenses, evocative of the real-world use of a magnifying glass lens. A third is stretching and squishing a rubber sheet. A fourth is working with vector fields.

The next five sections discuss several example distortion techniques to illustrate these method choices: 3D perspective, fisheye lenses, hyperbolic geometry, stretch and squish navigation, and magnification fields.

9.5.2.1 3D Perspective. One approach to distortion uses 3D perspective distortion, where there is a single focus of global extent, and the interaction metaphor is projection. The perspective distortion from standard 3D viewing is a very intuitive and familiar effect. It was used with the explicit intention of providing a distortion-based focus+context user experience in many early visualization systems, such as the influential Cone Tree system shown in Figure 9.18 [Robertson et al. 91].

Although many people found it compelling and expressed a strong preference for it on their first encounter, this approach lost popularity as the tradeoffs between the costs and benefits of 3D spatial layout for abstract information became more understood, as discussed in Section 7.1.

9.5.2.2 Fisheye Lenses. Fisheye distortion uses a single focus with local extent and radial shape, usually used with a lens interaction metaphor. The fisheye lens metaphor provides the same radial distortion effect as the physical optical lenses used with cameras and for door peepholes. The interaction with fisheye distortion is often implemented as a lens-style effect, where the user can interactively move a focus point around in the scene.

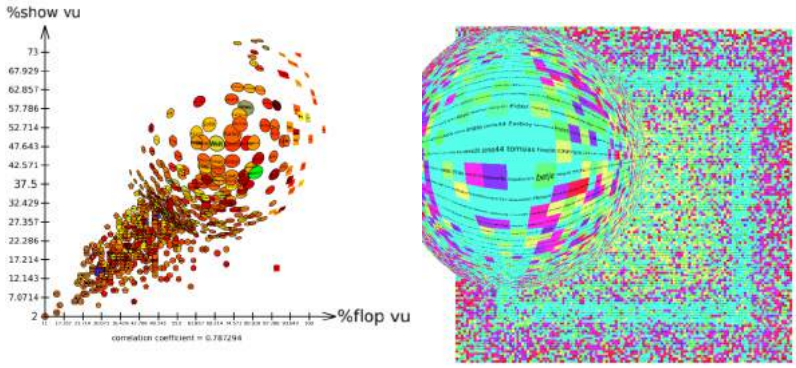


Figure 9.19: Focus+context with interactive fisheye lenses, with poker player dataset. (a) Scatterplot showing correlation between two strategies. (b) Pixel-oriented view showing correlation between a specific complex strategy and the player's winning rate, encoded by color. Courtesy of David Auber.

Figure 9.19 shows two examples of a fisheye lens used a online poker player dataset. The scatterplot shows the percentage of time that a player goes to showdown (playing until people have to show all of their cards) vs. the flop (playing until the point where three cards are placed face-up on the board). In the pixel-oriented view, blocks representing players are color-coded according their winning rate, and a space-filling curve is used to lay out these blocks in order of a specific derived attribute; in this case, a particular betting strategy. In the parts of the scene under the fisheye lens, the labels are large enough to read; that focus region remains embedded within the surrounding context, showing the global pattern within the rest of the dataset. The mathematics of fisheye distortion is straightforward; modern graphics hardware supports high performance fisheye lenses using vertex shaders [Lambert et al. 10].

9.5.2.3 Hyperbolic Geometry. Hyperbolic distortion uses a single global focus, usually radial in shape, with a projection interaction metaphor. Hyperbolic geometry provides an elegant mathematical formalism for creating global fisheye-style effects in a very different way than a local lens that warps the surface of an image. Instead, it uses projections that map from an infinite non-Euclidean space into a finite 2D circle or 3D sphere of Euclidean space.

Non-Euclidean geometry can easily accommodate structures such as trees that grow by an exponential factor. In Euclidean geometry, han-

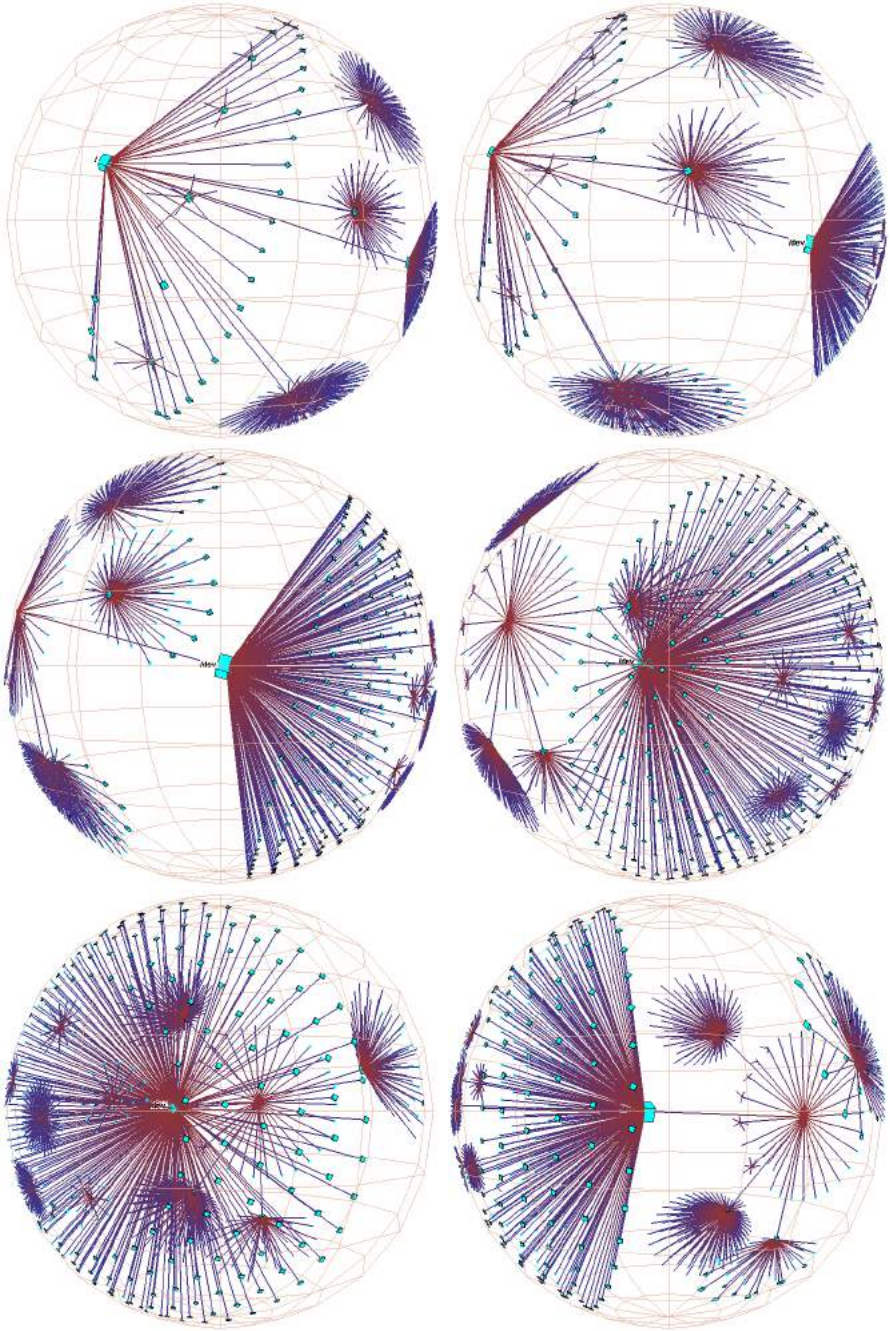


Figure 9.20: Navigation through a filesystem tree with 3D hyperbolic geometry, using hyperbolic translation to change the focus point and then standard rotation to spin the structure around. From [Munzner 98], Figure 3.

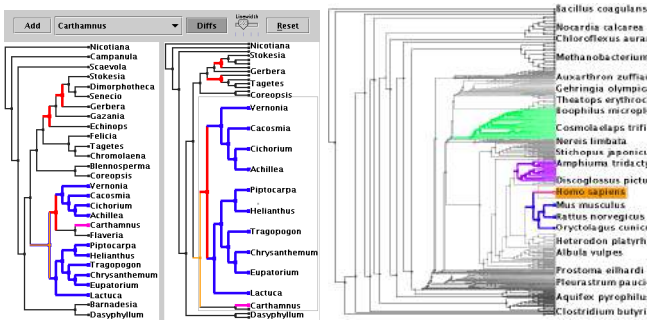


Figure 9.21: Stretch and squish navigation with multiple rectangular foci for exploring phylogenetic trees. (a) Stretching a single region when comparing two small trees. (b) Stretching multiple regions within a large tree. From [Munzner et al. 03], Figures 5 and 1.

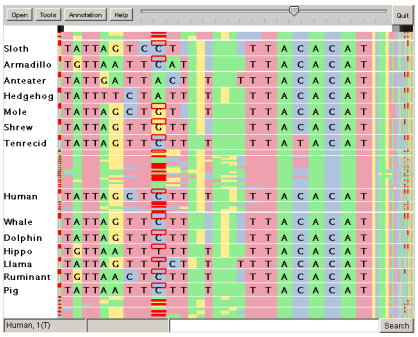


Figure 9.22: PRISequenceJuxtaposer features the guaranteed visibility of marks representing items with a high importance value, via a rendering algorithm with custom subpixel aggregation. From [Slack et al. 06], Figure 3.

dling exponential growth is a problem because the room available to lay out objects expands by a polynomial factor as structure sizes increase; for example, the area of a circle grows by a square factor of its radius. Figure 9.20 shows an example, where hyperbolic translation corresponds to changing the focus point of the projection, and then standard Euclidean rotation halfway around the sphere shows more of the filesystem structure.

9.5.2.4 Stretch and Squish Navigation. Stretch and squish distortion uses multiple rectangular foci of global extent, and the rubber sheet interaction metaphor. In this metaphor, the entire scene is considered to be drawn on

a rubber sheet where stretching one region squishes the rest, as shown in Figures 9.21, 9.11, and 9.22. The borders of the sheet stay fixed so that all items are within the viewport. The user can choose to separately stretch the rectangular focal regions in the horizontal and vertical directions.

These figures also illustrate the **guaranteed visibility** technique that ensures that important objects are always visible within the scene, even if they are very small. Guaranteed visibility is an example of aggregation that operates at the sub-pixel level and takes the importance attribute of each item into account. Standard graphics systems use assumptions that work well when drawing realistic scenes but not necessarily true for abstract visualizations. In reality, distant objects are not visually salient, so it is a reasonable optimization to simply not draw items that are sufficiently far away. If the viewpoint is moved closer to these objects, they will become larger on the image plane, and will be drawn. However, in abstract scenes the distance from the camera is a poor stand-in for the **importance** value for an object; often an original or derived attribute is used instead of or in combination with geometric distance. The example in Figure 9.22 is a collection of gene sequences that are over 16,000 nucleotides in width displayed in a frame of less than 700 pixels wide [Slack et al. 06]. The red marks that indicate differences between gene sequences stay visible at all times because they are given a high importance value, even in very squished regions where hundreds or thousands of items may fall within a single pixel. Figure 9.11 also illustrates this technique: the value used for the box color coding also indicates importance, so the boxes representing alerts that are colored red are always visible.

9.5.2.5 Magnification Fields. Magnification fields is a general computational framework featuring multiple foci of arbitrary magnification levels and shapes, whose scope can be constrained to affect only local regions. The underlying mathematical framework support calculations of the implicit magnification field required to achieve a desired transformation effect, as shown in Figure 9.23. The framework supports many possible interaction metaphors including lenses and stretchable surfaces. It can also expose the magnification fields directly to the user, for example data-driven magnification trails of moving objects.

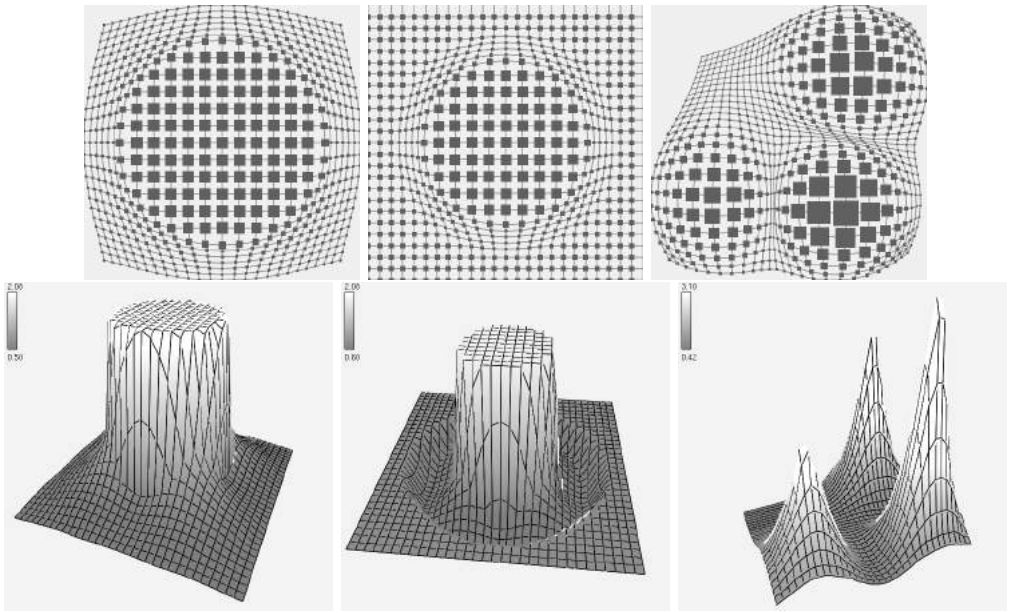


Figure 9.23: General frameworks calculate the magnification and minification fields needed to achieve desired transformations in the image. **Top row:** Desired transformations. **Bottom row:** Calculated magnification fields. From [Keahey 98], Figure 3.

9.6 History and Further Reading

Ahlberg and Shneiderman popularized filtering with tightly coupled views with dynamic queries, and extending standard widgets to better support these queries [Ahlberg and Shneiderman 94]. Willett et al. proposed scented widgets [Willett et al. 07], alluding to the idea of information scent proposed by Pirolli and others as part of the theory of information foraging [Pirolli 07].

Mackinlay, Card, and Robertson discuss constrained navigation in 3D [Mackinlay et al. 90]. Furnas and Bederson [Furnas and Bederson 95] introduced space-scale diagrams for visual reasoning about multiscale navigation. Van Wijk and Nuij present a framework for calculating smooth and efficient panning and zooming trajectories [van Wijk and Nuij 03]. Bederson and Hollan explored the possibilities of semantic zooming with the Pad++ system [Bederson and Hollan 94].

Hornbæk and Hertzum present a synthesis review of the many ways that overviews are used in information visualization [Hornbæk and Hertzum 11].

The definition in this chapter is a simplification of theirs, which distinguishes between the action of gaining an overview of an information space, and a specific view designed to support this action.

Tukey originally proposed the boxplot, popularized through his influential book on Exploratory Data Analysis [Tukey 77]. Wickham and Stryjewski discuss the many variants of boxplots that have been proposed in the past forty years [Wickham and Stryjewski 12].

Fua, Ward, and Rudensteiner proposed hierarchical parallel coordinates [Fua et al. 99].

Elmqvist and Fekete discuss hierarchical aggregation [Elmqvist and Fekete 10].

Furnas introduced the fundamental idea of generalized fisheye views [Furnas 82, Furnas 86]. His followup paper twenty years later questioned the overwhelming emphasis on geometric distortion in the work of many others [Furnas 06]. Other early proposals for focus+context interfaces were the Bifocal Display from Spence and Apperly [Spence and Apperley 82] and the polyfocal cartography of Kadmon and Shlomi [Kadmon and Shlomi 78]. Leung and Apperley provided an early taxonomy of distortion-based interfaces [Leung and Apperley 94], introducing the unifying vocabulary of magnification and transformation functions.

Influential 3D focus+context interfaces from Xerox PARC included the Perspective Wall [Mackinlay et al. 91] and Cone Trees [Robertson et al. 91]. Two general frameworks for focus+context magnification and minification are Carpendale's elastic presentation spaces [Carpendale et al. 95, Carpendale et al. 96] and Keahey's nonlinear magnification fields [Keahey and Robertson 97]. Munzner et al. proposed the guaranteed visibility technique and presented algorithms for scalable stretch and squish navigation in the TreeJuxtaposer system [Munzner et al. 03].

Cockburn et al. discuss many of the techniques that use the three major overviewing methods of temporal multiplexing through zooming, multiple views, and focus+context, in the context of the empirical evidence on their strengths and weaknesses [Cockburn et al. 08]. Lam and Munzner also provide an extensive discussion of the tradeoffs between these methods as part of their design guidelines for when and how to use them [Lam and Munzner 10].

DOSFA [Yang et al. 03a] is only one of many approaches to attribute reduction from Ward and others [Peng et al. 04, Yang et al. 04, Yang et al. 03b]. Johnansson and Johansson also propose a pipeline for attribute reduction [Johansson and Johansson 09]. Ankerst et al. provide an extensive exploration of similarity metrics for dimensional aggregation [Ankerst et al. 98].

Pearson proposed principal component analysis in 1901 [Pearson 01]. The foundational ideas behind multidimensional scaling were first proposed

by Young and Householder [Young and Householder 38] in the 1930s, then further developed by Torgerson [Torgerson 52] in the 1950s. Chalmers proposed an influential MDS system that used a stochastic force simulation approach [Chalmers 96]. Ingram et al. build on these ideas in the Glimmer system for MDS that exploits the parallelism of graphics hardware; that paper also discusses the history and variants of MDS in detail [Ingram et al. 09].

10

Methods: Slogans and Guidelines

10.1 Overview First, Detail on Demand, Zoom and Filter

Ben Shneiderman proposed the highly influential "visualization mantra" **Overview First, Detail on Demand, Zoom and Filter** in 1996 [Shneiderman 96].

TODO

10.2 Search, Show Context, Expand on Demand

TODO

10.3 Distortion Costs and Benefits

The development and evaluation of overviewing techniques has been a very active thread of research in information visualization. Nevertheless, the costs and benefits of the three major approaches to overviewing – temporal multiplexing vs. separate views vs. embedded focus+context – are still not fully understood.

What has gradually become clear is that distortion-based embedding in particular has measurable costs, in addition to whatever benefits it may provide. One cost is the problem that distance or length judgements are severely impaired, so distortion is a poor match with any tasks that require such comparisons. Thus, one of the most successful use cases for geometric distortion is with exploration of node-link graphs. Understanding the topological structure of the graph is conjectured to be robust to distortion when that structure is shown using lines to connect between nodes, or containment to show nested metanode structure using boxes, because precise

angle and length judgements are not necessary.

One potential cost is that users may not be aware of the distortion, and thus misunderstand the underlying object structure. This risk is highest when the user is exploring an unfamiliar or sparse structure, and many techniques incorporate explicit indications of distortion to lessen this risk. Hyperbolic views typically show the enclosing circle or sphere, magnification fields often show a superimposed grid or shading to imply the height of the stretched surface.

Even when users do understand the nature of the distortion, another cost is the internal overhead of maintaining **object constancy**: understanding that when an item seen in two different frames represents the same object, just seen from a different viewpoint. Understanding the underlying shape of a complex structure could require mentally subtracting the effect of the transformation in order to recognize the relationship between the components of an image before and after the transformation. Although in most cases we do this calculation almost effortlessly for standard 3D perspective distortion, the cost of mentally tracking general distortions increases as the amount of distortion increases [Lam et al. 06]. Some empirical evidence shows that constrained and predictable distortion is better tolerated than more drastic distortion [Lam and Munzner 10].

The originator of the generalized fisheye view approach has expressed surprise about the enthusiasm with which others have embraced distortion, and suggests that the question *what* is being shown in terms of selective filtering is more central than that of *how* is is shown with any specific distortion technique [Furnas 06]. For example, the fisheye metaphor is not limited to a geometric lens used after spatial layout; it can be used directly on structured data, such as a hierarchical document where some sections are collapsed while others are left expanded.

Figure 10.1 illustrates four different approaches on the same node-link graph: fisheye lens, an ordinary magnifying lens, and a neighborhood highlighting technique using only layering, and a combination of that layering with the Bring and Go interaction technique [Lambert et al. 10]. We will discuss these examples in detail to shed some light on the costs and benefits of distortion versus occlusion versus other interaction.

The local fisheye distortion has a small circle region of very high magnification at the center of the lens surrounded by a larger intermediate region that continuously varies from medium magnification to very high compression, returning to low compression in the outer periphery. Although fisheye lenses were developed with the goal of reducing the viewer's disorientation, unfortunately they can be quite disorienting. The continuous magnification change introduces some amount of cognitive load to untangle the underlying shape from the imposed distortion. Distortion is less problematic with familiar shapes, like geographic maps of known places, because people can

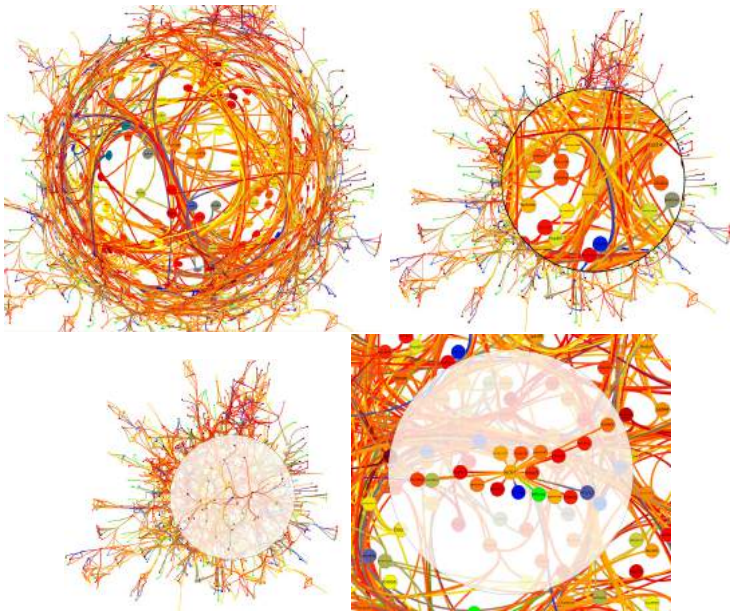


Figure 10.1: Four approaches to graph exploration. (a) Fisheye lens. (b) Magnifying lens. (c) Neighborhood highlighting with layering. (d) Neighborhood highlighting with both layering and Bring and Go interaction. From [Lambert et al. 10], Figures 2a, 2b, 3b, 4b.

interpret the distortion as a change to a known baseline. With unfamiliar structures, it can be difficult to reconstruct the basic shape by mentally removing the distortion effects. While these techniques are designed to be used in an interactive setting, where the user quickly move the lens around and compare the undistorted to the distorted states, there is still some cognitive load.

In contrast, the local magnifying lens has just two discrete levels: a highly magnified circle of medium size, and the low-compression periphery of medium size. There is a discontinuous jump between these two levels, where the lens occludes the immediately surrounding region. In this particular example, it is not clear that the benefit of avoiding occlusion is worth the cost of interpreting the continuous magnification change.

The last two approaches show a specific region of interest, in this case a local topological neighborhood of nodes reachable within one or two hops from a chosen target node. The neighborhood highlighting lens does not distort spatial position at all; it uses layering by reducing the opacity of items not in that neighborhood, automatically calculating a lens diameter

to accommodate the entire neighborhood. While this approach would not help for tasks where magnification is critical, such as reading node labels, it does a good job of supporting path following.

A fisheye lens can be interpreted as a temporary warping that affects the location of all objects within the active region. In contrast, the Bring and Go interaction technique [Moscovich et al. 09] is selective, temporarily changing the location of only specific objects of interest: that is, bringing the one-hop neighbors close to the target node. The layout is designed to simplify the configuration as much as possible while still preserving direction and relative distance information, in hopes of minimizing potential disorientation.

10.4 Networks: Connection vs. Containment vs. Matrix Views

TODO

10.5 Displaying Dimensionally Reduced Data

TODO

10.6 Further Reading

Furnas, who proposed several early focus+context techniques including generalized fisheye views [Furnas 82, Furnas 86], wrote a followup paper twenty years later that questioned the overwhelming emphasis on geometric distortion in the work of many others [Furnas 06].

Cockburn et al. discuss many of the techniques that use the three major overviewing methods of temporal multiplexing through zooming, multiple views, and focus+context, in the context of the empirical evidence on their strengths and weaknesses [Cockburn et al. 08]. Lam and Munzner also provide an extensive discussion of the tradeoffs between these methods as part of their design guidelines for when and how to use them [Lam and Munzner 10].

Part V

Analysis

Part V presents a gallery of specific techniques and systems analyzed according to the methods framework presented in Part IV. Each example is presented as a solution to a specific combination of data and task abstractions, and broken down according to the view construction and data reduction methods used to support visual encoding and interaction goals. The discussion of strengths and weaknesses of the techniques incorporates knowledge from evaluation when it is available.

The gallery is organized by data abstraction rather than visual representation in keeping with the needs of a designer following the nested model presented in Part I. When going from the abstraction layer to the technique layer, what the designer knows is the data and task abstractions and what the designer must choose or create is an appropriate technique. That choice gives rise the visual representation. It is organized by data abstraction rather than task abstraction because the taxonomy of visualization data types is significantly more mature than the taxonomy of tasks. Creating and gaining consensus within the field for a good taxonomy of tasks at multiple levels is still a very open research problem.

11

Analysis

This analysis chapter walks through many examples of visualization techniques and systems, analyzing them according to several factors: what data types are handled, what kind of derived data is computed for the data abstraction, what methods from the framework of Part IV are used, what visual encoding is used, what abstract tasks are addressed, and the scalability of the technique. The scalability analysis is based on a standard screen size of roughly one thousand by one thousand pixels, for simplicity in discussions of the available screen space. These examples continue the analysis gradually introduced in Part IV; now that all methods have been introduced, each example has a complete analysis.

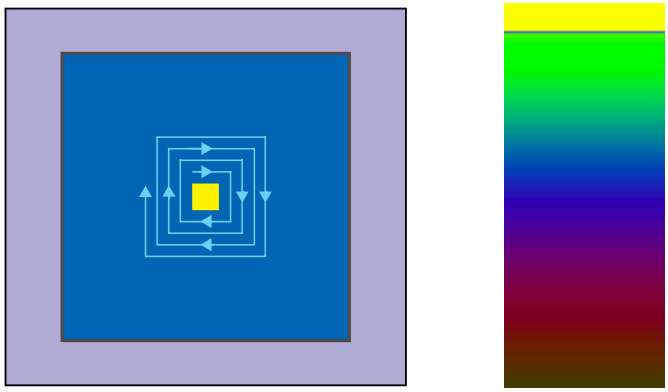
The chapter begins with sections addressing the major data types: tables, networks and trees, spatial data, and text. It continues with a section for systems that address combinations of multiple data types, which is the common case for real-world problems.

11.1 Tabular Data

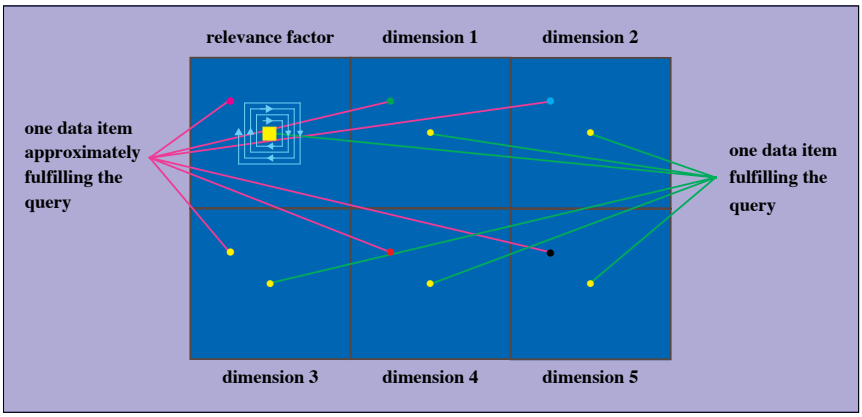
11.1.1 VisDB

The VisDB system for database visualization [Keim and Kriegel 94] treats an entire database as a very large table of data. The system shows the combination of that table and a specific query that matches some subset of the items in it. VisDB computes a set of derived attributes that measure the relevance of the query with respect to the original attributes and items. Each item is given a relevance score with respect to each original attribute. An overall relevance score is computed that combines these individual scores, conceptually creating an additional derived attribute column in the table.

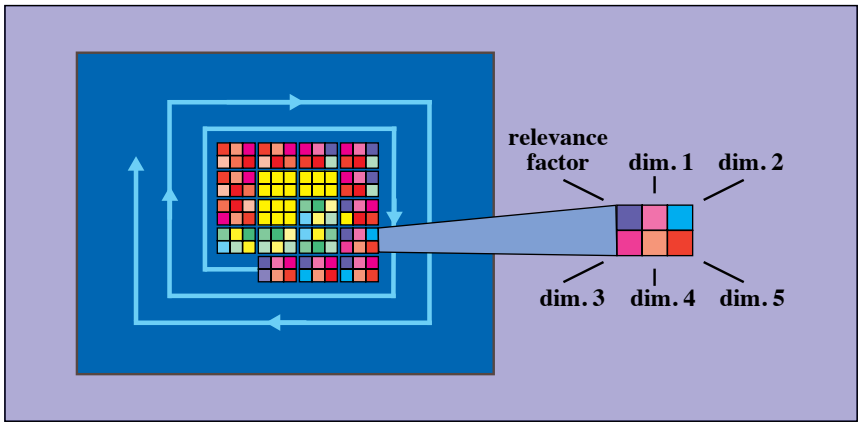
VisDB supports two major dense layout alternatives. Both share the same basic spatial layout and colormap. The spatial ordering within a view is not a standard rectilinear or radial layout; it follows a spiral pattern emanating from the center, as shown in Figure 11.1a. The sequential colormap,



(a) (b)

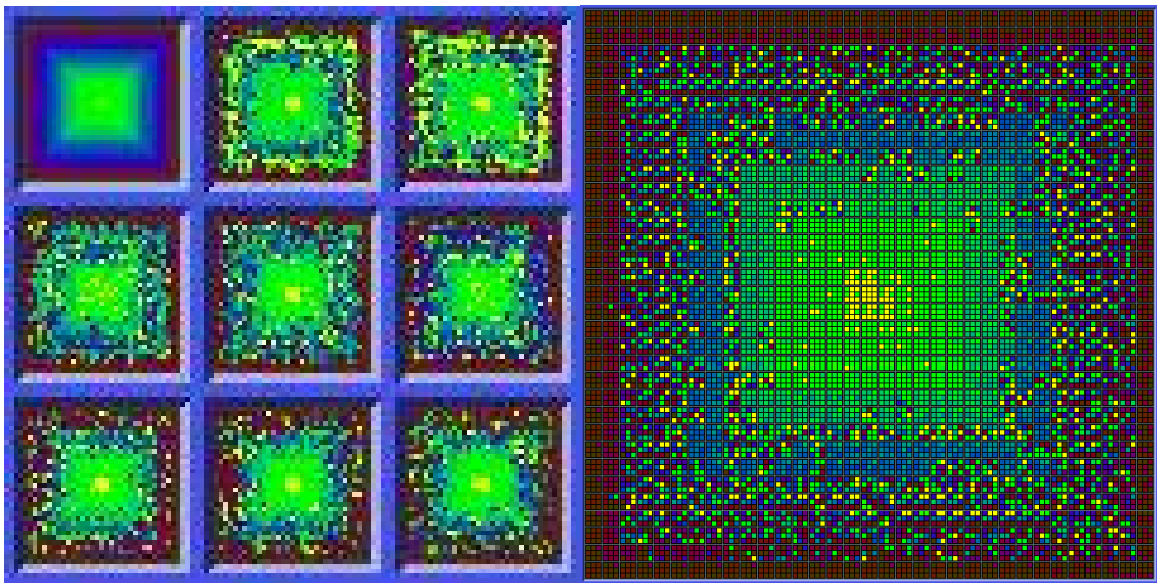


(c)



(d)

Figure 11.1: VisDB layouts schematically, for a dataset with 5 attributes. a) Each attribute is shown in a separate small-multiple view. b) Each item is shown with a glyph with per-attribute sections in a single combined view. c) These two views are ordered internally in a spiral emanating from the center. d) The sequential colormap uses multiple hues with monotonically increasing luminance. From [Keim and Kriegel 94], Figures 4, 2, 1, 5.



(a) (b)

Figure 11.2: VisDB layout examples with a dataset of 8 attributes and 1K items. a) One small multiple view for each attribute. b) Single combined view with multi-attribute glyph. c) Small-multiple views have four sectors each. From [Keim and Kriegel 94], Figure 6.

shown in Figure 11.1a, uses multiple hues with ordered with monotonically increasing luminance. At the bottom is dark red, the mid-range has purple and blue, it continues with cyan and then even brighter green, and there is a very bright yellow at the high end to emphasize the top of the range.

One possibility is to create one small-multiple view for each attribute, so that each view shows an attribute-based grouping of all the items for that attribute. Figure 11.1c illustrates the technique schematically, and Figure 11.2a shows an example with a dataset of 1000 items. The items are ordered by the derived overall relevance attribute, which is the upper left view; spatial position and color provide redundant information in this view. In the other views, the items are placed in the same order, but colored according to relevance score for that view's attribute, creating a different visual pattern of color that allows global comparison across attributes.

The alternative layout technique is a single-view layout that groups together all of the attributes for each item into a glyph, as in Figures 11.1d and 11.2b. This layout supports comparison across items, rather than

across attributes.

Both of these layouts are dense and spacefilling. In both, when the number of items is greater than the available pixels for the view, items are filtered according to the relevance values. The total table size can be many times the number of items visible in a single screen, up to several million. The small-multiples method scales to roughly 10-12 attributes, and around a million total items across all views given the limit of one item per pixel; each view can thus show around 100K items. In contrast, the glyph-based method scales to about an order of magnitude fewer visible items, around 100K. The elements within the glyph need to be boxes larger than a single pixel in order to be salient, and glyphs need borders around them in order to be distinguished from neighboring ones.

System	**VisDB**
Data Types	table (database) + query returning table subset (database query)
Derived Data	k+1 quantitative attributes per original item: query relevance for the k original attributes plus overall relevance
View Comp.	dense, spacefilling, spatial position channel, color channel, glyphs
Reduction	filtering
Multiple Views	small multiples
Abstract Tasks	find distribution, correlation, outliers/hotspots, clusters/groups, correspondences/similarities
Domain Tasks	query refinement
Scalability	attributes: 1 dozen total items: several million visible items (using multiples, in total): 1 million visible items (using glyphs): 100K

11.1.2 Hierarchical Clustering Explorer

The Hierarchical Clustering Explorer (HCE) supports exploration of hierarchically clustered multidimensional tables. It was originally designed for the genomics domain where the table reprents microarray measurements of gene expression. The original data is a multidimensional table with two independent attributes, genes and experimental conditions, and a single quantitative dependent attribute, measurements of the activity of each gene under each experimental condition.

HCE uses multiple types of derived data. First, a cluster hierarchy of the items is created based on a similarity measure between items. The

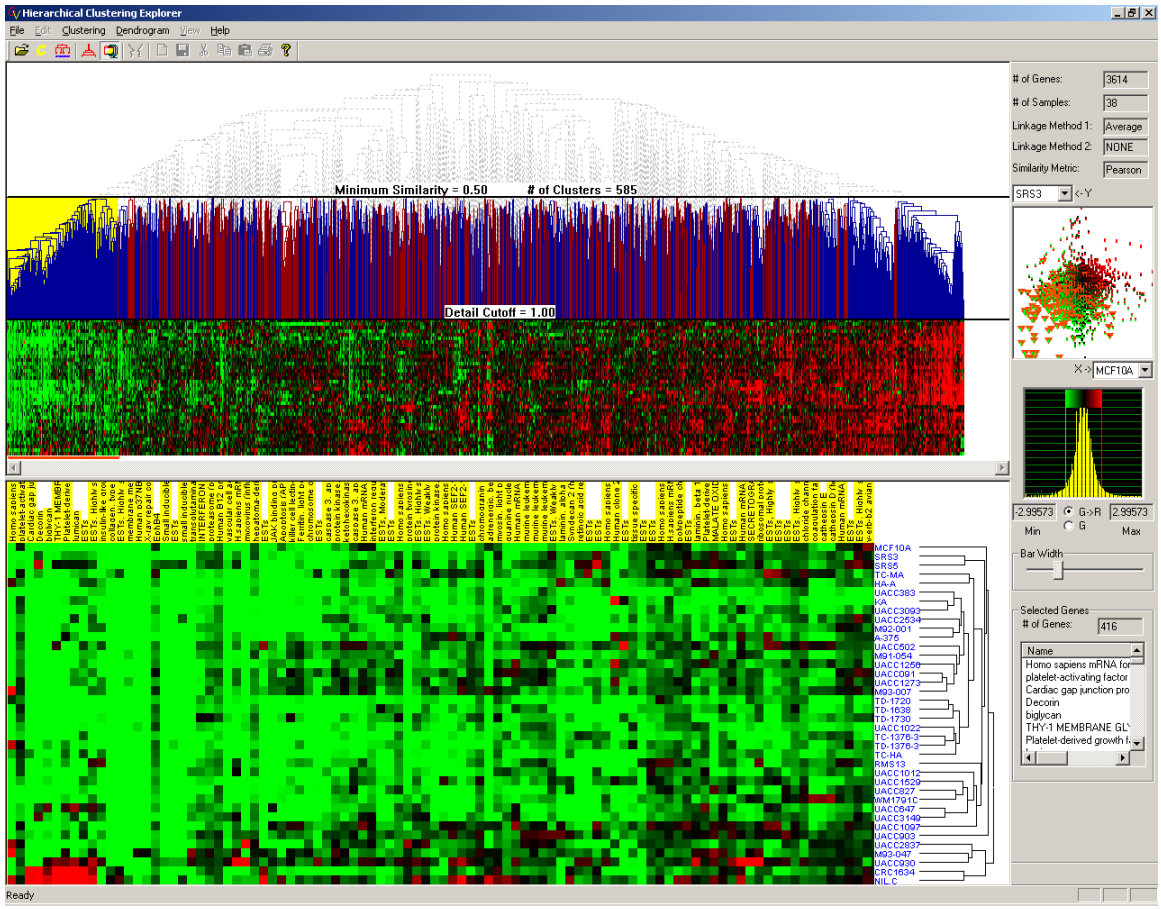


Figure 11.3: Hierachical Clustering Explorer uses interactive aggregation and filtering for the scalable display of a multidimensional table using multiple overview+detail views of a cluster heatmap. From [Seo and Shneiderman 02], Figure 2.

clustering is computed outside the tool itself, and is expected as input to the system.

Figure 11.3 shows HCE, a multiform overview-detail multiple-view system with linked highlighting. The primary view in HCE is a cluster heatmap. There are also scatterplot views for showing a pairwise combination of any two attributes, and histogram views to show the distribution of a single attribute.

For more on cluster heatmaps, see Section 8.1.3.2.

The scalability target of HCE is between 100 and 20,000 gene attributes

and between 2 and 80 experimental condition attributes. This target is reached with the combination of several methods: multiple views with overview and detail, and interactively controlled aggregation, filtering, and navigation. Figure 11.3 shows the two cluster heatmap views in HCE. The overview at the top uses an aggregated representation where an entire dataset of 3614 genes is shown in less then 1500 pixels, by replacing individual leaves with the average values of adjacent leaves. The density level of the overview can be interactively changed by the user, for a tradeoff between an aggregate view where some detail is lost but the entire display is visible at once, and a more zoomed-in view where only some of the columns are visible simultaneously and navigation by horizontal panning is required to see the rest. The horizontal line through the dendrogram labelled Minimum Similarity is an interactive filtering control. Dragging it down vertically dynamically filters out the columns in the heatmap that correspond to the parts of the dendrogram above the bar, and partitions the heatmap into pieces that correspond to the number of clusters just below the bar.

The detail view at the bottom shows a heatmap of the cluster selected in the top overview. It also shows the second dendrogram for hierarchical clustering of the rows on the side; this dendrogram is not shown above in order to to maximize the number of columns that can fit within in the overview. The background of the selected cluster is highlighted in yellow in the overview, and the correspondence between the views is emphasized by coloring the column labels along the top of the detail view yellow as well, for linked highlighting.

Figure 11.4 shows a later version of HCE, incorporating the rank-by-feature technique that combines the methods of reordering and aggregation to guide exploration and achieve scalability. In this technique, the data abstraction is augmented with new derived attributes: for several choices or ordering criteria, each original attribute and pairwise combination of attributes are ordered by that criterion. The resulting rankings are used to reorder the attributes within a list, or the pairs of attributes within a table. These reorderings are shown in a compact aggregate view for an overview display that has linked detail view to show a particular selection.

Figure 11.6 shows a close-up view of the interface for ordering attributes, with four linked views that also use the overview/detail method. On the left is a control panel for selecting the ordering criterion. Many possibilities exist; in their prototype, the authors chose the normality of the distribution, its uniformity, the number of outliers, the number of unique values, and the size of the biggest gap. The next view, Score Overview, is a very compact dense view, with a vertical strip of color-coded boxes that have labels to their right. Each box corresponds to one attribute, and shows the quantitative derived attribute that was computed according to the chosen

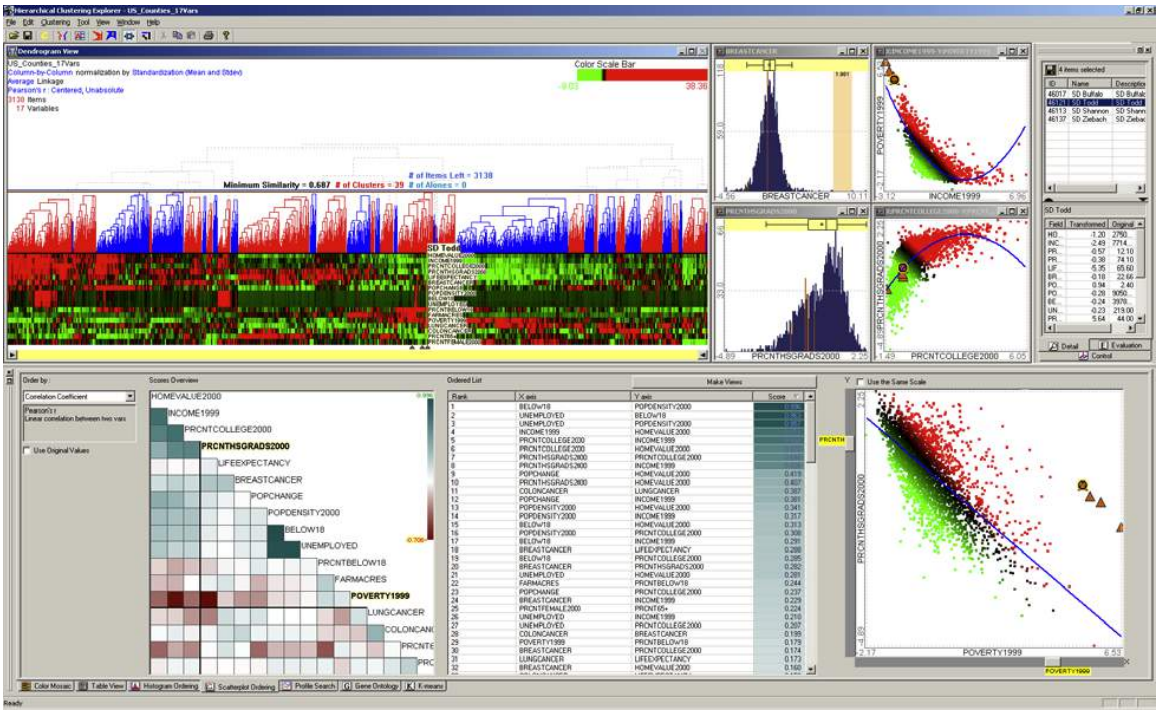


Figure 11.4: A later version of HCE uses the methods of interactive reordering, aggregation, and linked views to guide users in exploring the dataset systematically. From [Seo and Shneiderman 05], Figure 1.

criterion with a continuous diverging colormap, which is shown on the right side of the view as a legend. This overview is linked to the main heatmap view by shared spatial position of the boxes, which have the same vertical ordering as in the heatmap. This dense overview is designed to show all attributes at once without the need to scroll.

The middle Ordered List view is a less compact list view that also has one line per attribute, but each line is taller in order to fit a readable font, so panning navigation is provided by a scrollbar. Each line is wide enough to show a 5-number summary of the attribute's distribute and the numeric value of the ordering criterion superimposed in the box. The key aspect of this view is that the vertical ordered is linked to the criterion chosen in the control panel on the left. Finally, the rightmost detail view shows an attribute chosen by selection in one of the other views. This view visually encodes the entire distribution of the attribute as a histogram on the bottom with the vertical spatial position channel, and as a boxplot

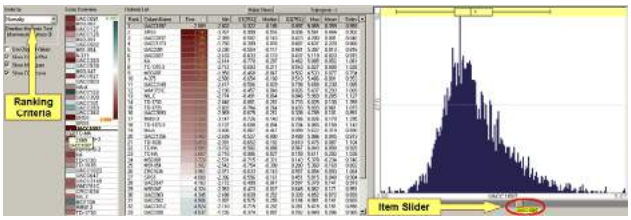


Figure 11.5: The HCE rank-by-feature interface for ordering attributes, with a combined histogram and boxplot for the detail view. From [Seo and Shneiderman 05], Figure 2.

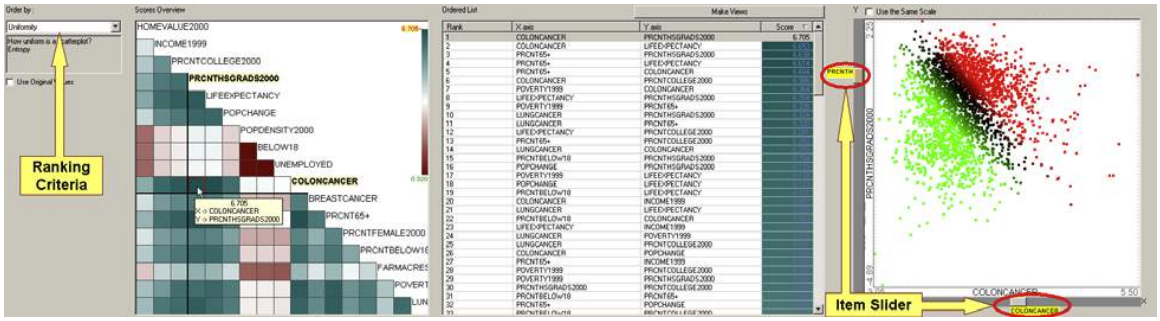


Figure 11.6: The HCE rank-by-feature interface for ordering pairwise combinations of attributes, with a scatterplot for the detail view. From [Seo and Shneiderman 05], Figure 3.

along the top to show the summary statistics using a glyph with horizontal spatial position. The attribute shown in the detail view can also be changed by dragging the slider at the bottom.

Figure 11.6 shows a close-up view of the interface for ordering pairwise combinations of attributes. In this case, the overview is the lower triangle of a matrix, and the detail view is a scatterplot. Just as with list ordering, there are many possible choices of matrix reordering criteria. The authors provide correlation coefficient, least-square errors, quadracity, number of outliers, uniformity of scatterplots, and number of items in a specifiable region of interest. The Ordered List view is typically requires several screenfulls to scroll through, since it includes every pairwise combination of the attributes; it only shows the names of both attributes and the derived criterion value itself, and not any other numerical values. Two item sliders allow fast changes of the attribute pair shown in the detail view.

System	**Hierarchical Clustering Explorer (HCE)**
Data Types	multidimensional table: 2 categorical key attributes (genes, conditions); 1 quantitative value attribute (gene activity level in condition)
Derived Data	hierarchical clustering of table rows regression lines for each pairwise attribute combination (scatterplot) quantitative derived attribute for each attribute and pairwise attribute combination, for each ranking criterion for each attribute, 5 quantitative values summarizing distribution (for boxplot and ordered list views)
View Coord.	views: cluster heatmap, scatterplots, histograms, boxplots, rank-by-feature overviews: continuous diverging colormaps, on list or lower triangular matrix
Interaction	heatmap overview: changeable levels of aggregation, filtering; navigation through panning; selection for detail views rank-by-feature overviews: selection for detail views rank-by-feature mid-level views: interactive reordering, navigation with scrolling
Reduction	filtering, aggregation, navigation (scrolling)
Multiple Views	multiform with linked highlighting and shared spatial position overview/detail with selection in overview populating detail view
Abstract Tasks	find relationships between attributes (correlation) find clusters, gaps, outliers, trends of items
Domain Tasks	gene expression measurement exploration demographics exploration
Scalability	genes (independent attribute): 20K conditions (independent attribute): 80 gene activity in condition (quantitative dependent attribute): 20K*80 = 1.6M

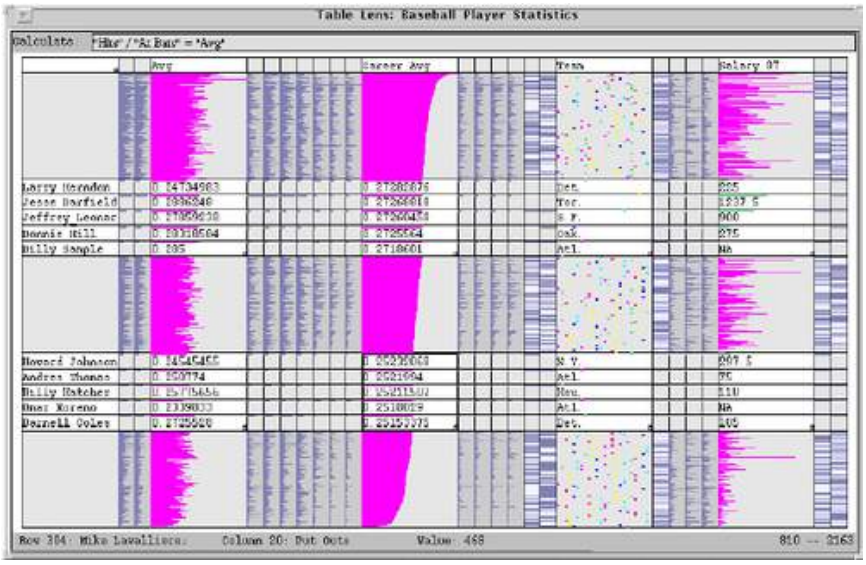


Figure 11.7: The TableLens system features interactive resorting of all item rows when a column of attributes is clicked [Rao and Card 94].

11.1.3 Table Lens

TODO

11.2 Networks and Trees

11.2.1 PivotGraph

Connection, containment, and matrix views of networks are different visual encodings of the same data abstraction; they both depict the link structure of a network. In contrast, the PivotGraph technique [Wattenberg 06] visually encodes a different abstraction: a new network derived from the original one by aggregating groups of nodes and links into a **roll-up** according to categorical attribute values on the nodes. The user can also **select** attributes of interest that filter the derived network. Roll-ups can be made for up to two attributes at once; for two dimensions nodes are laid out on a grid, and for one dimension they are laid out on a line. Node positions in the grid are computed to minimize link-crossing clutter, and the links between them are drawn as curves. The user interactively explores the graph through roll-up and selection to see visual encodings that

directly summarize the high-level relationships between the attribute-based groups, and can drill down to see more details for any node or link on demand. When the user chooses a different roll-up, an animated transition smoothly interpolates between the two layouts.

Figure 11.8 shows an example of a simple node-link drawing side by side with a PivotGraph roll-up. The network has two categorical attributes: gender, shown with `M` for male and `F` for female, and company division, shown with `1` or `2`. In the node-link view, node shape represents gender with squares for `M` and circles for `F`, and greyscale value represents company division with black for `1` and grey for `2`. The full original network is shown, emphasizing its topological structure. In the PivotGraph view, the derived network has four nodes that represent the four possible groups of the combination of these two attributes: males in division 1 in the upper left, females in division 2 in the upper right, males in division 2 on the lower left, and females in division 2 on the lower right. The size of each of these aggregate groups is encoded with node size. Similarly, a single link in this derived graph represents the entire group of edges in the original graph that linked items in these groups, and the size of that group is encoded with line width. We can quickly see the magnitude of these groups, for example that all possible gender/division pairs are connected except for men and women in division 2. In Figure 11.8c, a more complex network is rolled up by two categorical attributes that have five values for each, with a third quantitative attribute encoded by a divering red-green color scale.

The PivotGraph technique is highly scalable because it summarizes an arbitrarily large number of nodes and links in the original network, easily handling from thousands to millions. The visual complexity of the derived network layout depends only on the number of attribute values for the two attributes chosen for roll-up.

PivotGraph is very well suited for the task of comparisons across attributes at the aggregate level, a task that is difficult with both node-link and matrix views. Conversely, it is poorly suited for understanding topological features of networks; PivotGraph is thus suitable as one of several linked multiform views of networks, so that the complementary strenths of each view can be exploited.

Table 11.1 summarizes the analysis of PivotGraph.

11.2.2 Constellation

TODO

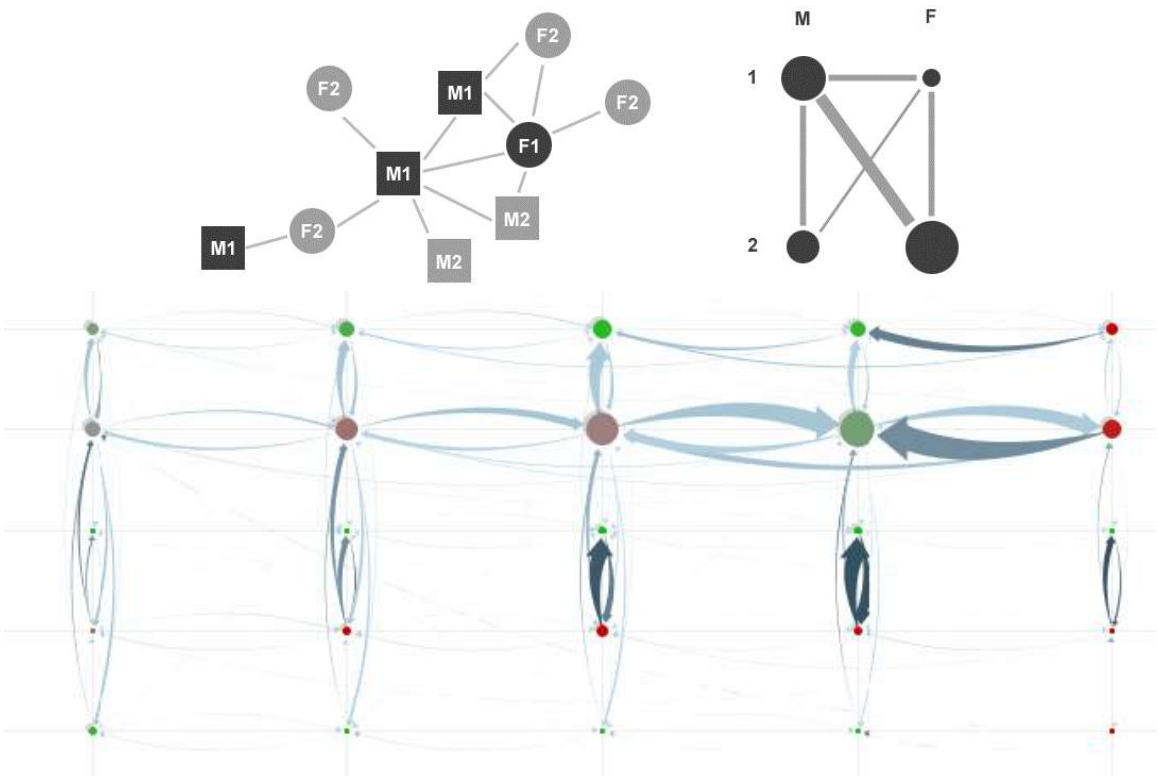


Figure 11.8: PivotGraph. a) Node-link view of small network with two attributes on nodes: gender (`M/F`) is encoded by node shape, and company division (`1/2`) is encoded by greyscale value. b) The schematic PivotGraph roll-up of the same simple network where size of nodes and links of the derived graph shows the number of items in these aggregated groups. c) PivotGraph representation of more complex network rolled up by two categorical attributes with five values of each. From [Wattenberg 06], Figures 4 and 1.

11.2.3 InterRing

The InterRing technique [Yang et al. 02] for tree exploration uses a space-filling radial layout for visually encoding the hierarchy. The interaction is built around a multi-focus focus+context distortion approach to change the amount of room allocated to different parts of the hierarchy. Figure 11.9 shows an example of distortion used to emphasize interactively indicated regions of interest. The structure-based coloring redundantly emphasizes

Technique	**PivotGraph**
Data Types	network
Derived Data	derived network of aggregate nodes/links based on grouping by chosen attributes
View Comp.	connection, size
Reduction	aggregation, filtering
Abstract Tasks	cross-attribute comparison of node groups
Scalability	nodes/edges (original): unlimited attributes: 2 roll-up, several filter per layout values per attribute: several, up to 1 dozen

Table 11.1: Analysis of PivotGraph.

the hierarchical information encoded spatially. While it could be eliminated in a single-view context if an unrelated attribute should be color coded instead, it is intended to integrate with multiple-view usage when seeing a color coding of the hierarchical structure is desirable in the other linked views with different spatial layouts.

The scalability of InterRing is moderate; it handles hundreds of nodes easily, and can scale to several thousand nodes if given a full screen for its layout. The main constraint is screen space: as a spacefilling method, the intent is is that there is enough room to encode each leaf in the tree with at least one pixel. The number of edges in a tree is the same as the number of nodes, so there is no need to separately analyze that aspect. Another useful factor to consider with trees is the maximum **tree depth** supported by the encoding; that is, the number of levels between the root and the furthest-away leaves. InterRing can handle up to several dozen levels, whereas hundreds would be overwhelming.

In addition to being a viable single-view approach when tree exploration is the only task, InterRing is designed to work well with other views where a hierarchy view should support selection, navigation, and rollup/drilldown operations. It also supports directly editing the hierarchy itself. In contrast, many tree browsing systems do not support modification of the hierarchy.

Table **??** summarizes the analysis of parallel coordinates and their hierarchical variant.

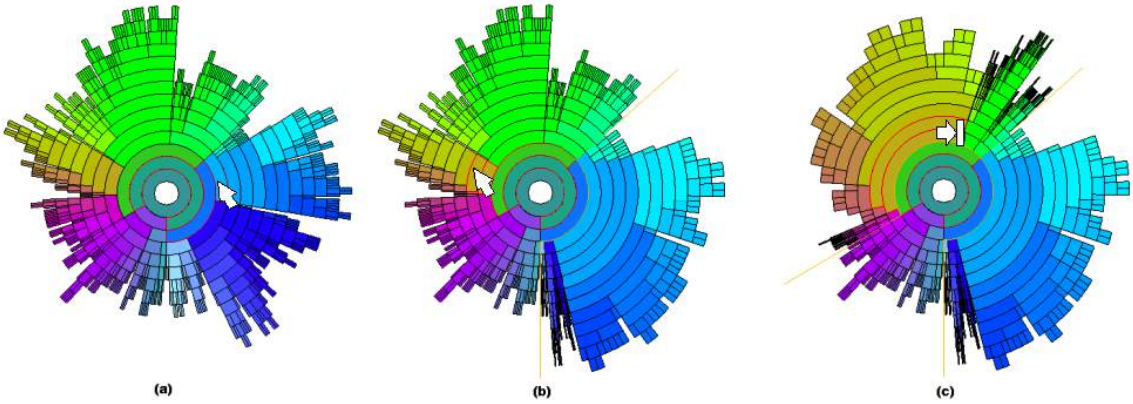


Figure 11.9: The InterRing hierarchy visualization technique uses a space-filling radial visual encoding and distortion-based focus+context interaction. a) The hierarchy before distortion. b) The blue selected subtree is enlarged. c) A second tan region is enlarged. From [Yang et al. 02], Figure 4.

Technique	**InterRing**
Data Types	tree
View Comp.	color, radial, spacefilling
Reduction	focus+context distortion
Abstract Tasks	selection, rollup/drilldown, hierarchy editing
Scalability	nodes: thousands levels in tree: dozens

Table 11.2: Analysis of InterRing.

11.3 Spatial Data

TODO

11.4 Text/Logs

11.4.1 Rivet PView

TODO

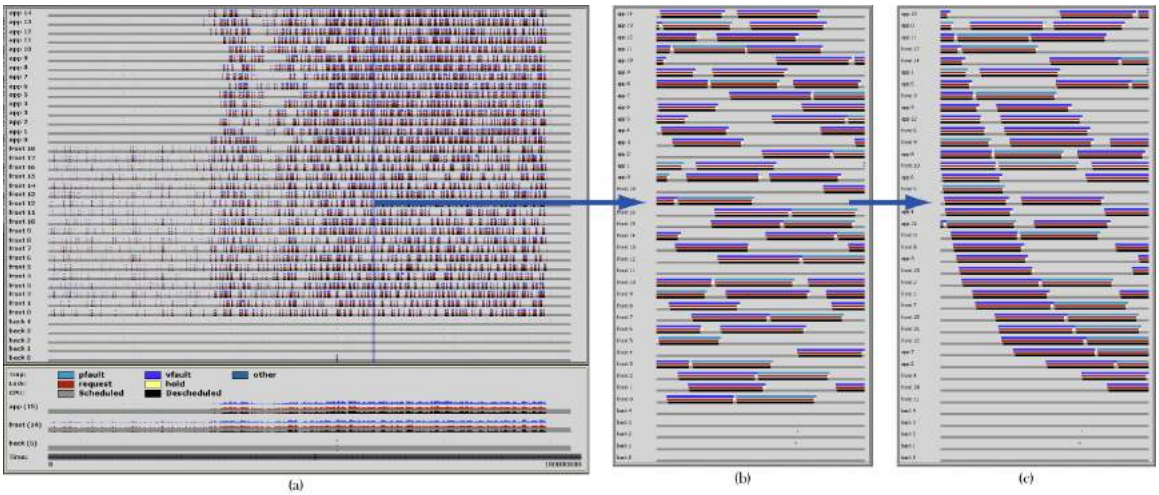


Figure 11.10: The Rivet PView system for performance analysis of parallel systems shows a clear pattern when the rows are reordered according to the acquisition time for a particular lock. From [Bosch et al. 00], Figure 7.

11.5 Complex Combinations

TODO

11.6 History and Further Reading

Keim and Krieger created the VisDB system [Keim and Kriegel 94].

Seo and Shneiderman designed several versions of the Hierarchical Clustering Explorer system [Seo and Shneiderman 02,Seo and Shneiderman 05]. Wilkinson and Friendly cover the rich history of the cluster heatmap [Wilkinson and Friendly 09].

Henry et al. extensively explore the use of matrix views in combination with node-link views for graphs, particularly for social network analysis [Henry and Fekete 06, Henry et al. 07]. van Ham also discusses the scalability and interaction possibilities of matrix views in the software engineering domain [van Ham 03]. Ghoniem, Fekete, and Castagliola empirically characterized the uses of matrix versus node-link views for a broad set of abstract tasks [Ghoniem et al. 05].

Wattenberg proposed the PivotGraph technique [Wattenberg 06].

Yang, Ward, and Rudensteiner proposed the InterRing technique [Yang et al. 02].

Bibliography

[Agrawala and Stolte 02] Maneesh Agrawala and Chris Stolte. "Rendering Effective Route Maps: Improving Usability Through Generalization." In *Proc. SIGGRAPH*, pp. 241–249, 2002.

[Ahlberg and Shneiderman 94] Chris Ahlberg and Ben Shneiderman. "Visual information seeking: Tight coupling of dynamic query filters with starfield displays." In *Proc. ACM Conf. on Human Factors in Computing Systems (CHI)*, pp. 313–317, 1994.

[Aigner et al. 11] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. *Visualization of Time-Oriented Data*. Springer, 2011.

[Amar and Stasko 04] Robert Amar and John Stasko. "A Knowledge Task-Based Framework for the Design and Evaluation of Information Visualizations." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 143–149, 2004.

[Amar et al. 05] Robert Amar, James Eagan, and John Stasko. "Low-Level Components of Analytic Activity in Information Visualization." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 111–117, 2005.

[Andre and Wickens 95] Anthony D. Andre and Christopher D. Wickens. "When Users Want What's not Best for Them." *Ergonomics in Design: The Quarterly of Human Factors Applications* 3:4 (1995), 10–14.

[Andrews 08] Keith Andrews. "Evaluation Comes In Many Guises." AVI Workshop on BEyond time and errors (BELIV) Position Paper, 2008. www.dis.uniroma1.it/~beliv08/pospap/andrews.pdf.

[Ankerst et al. 98] Michael Ankerst, Stefan Berchtold, and Daniel A. Keim. "Similarity clustering of dimensions for an enhanced visualization of multidimensional data." In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 52–60, 1998.

[Anscombe 73] F.J. Anscombe. "Graphs in Statistical Analysis." *American Statistician* 27 (1973), 17–21.

[Archambault et al. 07] Dan Archambault, Tamara Munzner, and David Auber. "TopoLayout: Multilevel Graph Layout by Topological Features." *IEEE Trans. on Visualization and Computer Graphics* 13:2 (2007), 305–317.

[Archambault et al. 08] Dan Archambault, Tamara Munzner, and David Auber. "GrouseFlocks: Steerable Exploration of Graph Hierarchy Space." *IEEE Trans. on Visualization and Computer Graphics* 14:4 (2008), 900–913.

[Auber et al. 12] David Auber, Daniel Archambault, Romain Bourquiand Antoine Lambert, Morgan Mathiaut, Patrick Mary, Maylis Delest, Jonathan Dubois, and Guy Melançon. "The Tulip 3 Framework: A Scalable Software Library for Information Visualization Applications Based on Relational Data." Technical report, INRIA Research Report 7860, 2012.

[Auber 02] David Auber. "Using Strahler numbers for real time visual exploration of huge graphs." In *International Conference on Computer Vision and Graphics*, pp. 56–69, 2002.

[Bachthaler and Weiskopf 08] Sven Bachthaler and Daniel Weiskopf. "Continuous Scatterplots." *IEEE Trans. Visualization and Computer Graphics (Proc. Vis 08)* 14:6 (2008), 1428–1435.

[Baldonado et al. 00] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. "Guidelines for Using Multiple Views in Information Visualizations." In *Proc. ACM Advanced Visual Interfaces (AVI)*, pp. 110–119, 2000.

[Barsky et al. 07] Aaron Barsky, Jennifer L. Gardy, Robert E. Hancock, and Tamara Munzner. "Cerebral: a Cytoscape plugin for layout of and interaction with biological networks using subcellular localization annotation." *Bioinformatics* 23:8 (2007), 1040–1042.

[Barsky et al. 08] Aaron Barsky, Tamara Munzner, Jennifer Gardy, and Robert Kincaid. "Cerebral: Visualizing Multiple Experimental Conditions on a Graph with Biological Context." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2008)* 14:6 (2008), 1253–1260.

[Becker and Cleveland 87] Richard A. Becker and William S. Cleveland. "Brushing Scatterplots." *Technometrics* 29 (1987), 127–142.

[Becker et al. 96] Ronald A. Becker, William S. Cleveland, and Ming-Jen Shyu. "The Visual Design and Control of Trellis Display." *Journal of Computational and Statistical Graphics* 5:2 (1996), 123–155.

[Bederson and Hollan 94] Ben Bederson and James D Hollan. "Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics." In *Proc. User Interface Software and Technology (UIST)*, pp. 17–26, 1994.

[Bergman et al. 95] Lawrence D. Bergman, Bernice E. Rogowitz, and Lloyd A. Treinish. "A Rule-based Tool for Assisting Colormap Selection." In *Proc. IEEE Visualization (Vis)*, pp. 118–125, 1995.

[Bertin 67] Jacques Bertin. *Sémiologie Graphique: Les diagrammes – Les réseaux – Les cartes.* reissued by Editions de l'Ecole des Hautes Etudes en Sciences, 1999, 1967.

[Bier et al. 93] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. "Toolglass and Magic Lenses: The See-Through Interface." *Computer Graphics (Proc. SIGGRAPH 93)*, pp. 73–80.

[Booshehrian et al. 11] Maryam Booshehrian, Torsten Möller, Randall M. Peterman, and Tamara Munzner. "Vismon: Facilitating Risk Assessment and Decision Making In Fisheries Management." Technical report, School of Computing Science, Simon Fraser University, Technical Report TR 2011-04, 2011.

[Bosch et al. 00] Robert Bosch, Chris Stolte, Gordon Stoll, Mendel Rosenblum, and Pat Hanrahan. "Performance Analysis and Visualization of Parallel Systems Using SimOS and Rivet: A Case Study." In *Proc. Symp. High-Performance Computer Architecture (HPCA)*, pp. 360–371, 2000.

[Bostock et al. 11] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. "D3: Data-Driven Documents." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2011)* 17:12 (2011), 2301–2309.

[Brandes 01] Ulrik Brandes. "Chapter 4, Drawing on Physical Analogies." In *Drawing Graphs: Methods and Models*, LNCS Tutorial, 2025, edited by M. Kaufmann and D. Wagner, LNCS Tutorial, 2025, pp. 71–86. Springer-Verlag, 2001.

[Brewer 99] Cynthia A. Brewer. "Color Use Guidelines for Data Representation." In *Proceedings of the Section on Statistical Graphics*, pp. 55–60. American Statistical Association, 1999.

[Buchheim et al. 02] Christoph Buchheim, Michael Jünger, and Sebastien Leipert. "Improving Walker's Algorithm to Run in Linear Time." In *Proc. Graph Drawing (GD'02)*, 2528, 2528, pp. 344–353. Springer LNCS, 2002.

[Card et al. 91] Stuart Card, George Robertson, and Jock Mackinlay. "The Information Visualizer, An Information Workspace." In *Proc. ACM Conf. on Human Factors in Computing Systems (CHI)*, pp. 181–186, 1991.

[Card et al. 99] Stuart K. Card, Jock Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, 1999.

[Carpendale et al. 95] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. "Three-Dimensional Pliable Surfaces: For Effective Presentation of Visual Information." In *Proc. ACM Symp. User Interface Software and Technology (UIST)*, pp. 217–226, 1995.

[Carpendale et al. 96] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. "Distortion Viewing Techniques for 3D Data." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 46–53, 1996.

[Carpendale 08] Sheelagh Carpendale. "Evaluating Information Visualizations." In *Information Visualization: Human-Centered Issues and Perspectives*, 4950, edited by Andreas Kerren, John T. Stasko, Jean-Daniel Fekete, and Chris North, 4950, pp. 19–45. Springer LNCS, 2008.

[Chalmers 96] M. Chalmers. "A Linear Iteration Time Layout Algorithm for Visualising High Dimensional Data." In *Proc. IEEE Visualization*, pp. 127–132, 1996.

[Chi and Riedl 98] Ed H. Chi and John T. Riedl. "An Operator Interaction Framework for Visualization Systems." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 63–70, 1998.

[Chuah 98] Mei C. Chuah. "Dynamic Aggregation with Circular Visual Designs." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 35–43, 1998.

[Cleveland and McGill 84] William S. Cleveland and Robert McGill. "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods." *Journal of the American Statistical Association* 79:387 (1984), 531–554.

[Cleveland et al. 88] William S. Cleveland, Marylyn E. McGill, and Robert McGill. "The Shape Parameter of a Two-Variable Graph." *Journal of the American Statistical Association* 83:402 (1988), 289–300.

[Cleveland 93a] William S. Cleveland. "A Model for Studying Display Methods of Statistical Graphics (with Discussion)." *Journal of Computational and Statistical Graphics* 2:4 (1993), 323–364.

[Cleveland 93b] William S. Cleveland. *Visualizing Data*. Hobart Press, 1993.

[Cockburn and McKenzie 00] Andy Cockburn and Bruce McKenzie. "An Evaluation of Cone Trees." In *People and Computers XIV: Usability or Else. British Computer Society Conf. on Human Computer Interaction*, pp. 425–436. Springer Verlag, 2000.

[Cockburn and McKenzie 01] Andy Cockburn and Bruce McKenzie. "3D or Not 3D? Evaluating the Effect of the Third Dimension in a Document Management System." In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 434–441, 2001.

[Cockburn et al. 08] Andy Cockburn, Amy Karlson, and Benjamin B. Bederson. "A Review of Overview+Detail, Zooming, and Focus+Context Interfaces." *ACM Computing Surveys* 41:1 (2008), 1–31.

[Cormen et al. 90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[Craig and Kennedy 03] Paul Craig and Jessie Kennedy. "Coordinated Graph and Scatter-Plot Views for the Visual Exploration of Microarray Time-Series Data." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 173–180, 2003.

[Csikszentmihalyi 91] Mihaly Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. Harper, 1991.

[Davidson et al. 01] George S. Davidson, Brian N. Wylie, and Kevin W. Boyack. "Cluster Stability and the Use of Noise in Interpretation of Clustering." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 23–30, 2001.

[Diehl et al. 10] Stephan Diehl, Fabian Beck, and Micheal Burch. "Uncovering Strengths and Weaknesses of Radial Visualizations - an Empirical Approach." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis)* 16:6 (2010), 935–942.

[Dow et al. 05] Steven Dow, Blair MacIntyre, Jaemin Lee, Christopher Oezbek, Jay David Bolter, and Maribeth Gandy. "Wizard of Oz support throughout an iterative design process." *IEEE Pervasive Computing* 4:4 (2005), 18–26.

[Draper et al. 09] Geoffrey M. Draper, Yarden Livnat, and Richard F. Riesenfeld. "A survey of radial methods for information visualization." *IEEE Trans. Visualization and Computer Graphics* 15:5 (2009), 759–776.

[Eagan et al. 01] James Eagan, Mary Jean Harrold, James A. Jones, and John Stasko. "Technical Note: Visually Encoding Program Test Information to Find Faults in Software." In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 33–36, 2001.

[Eick et al. 92] Stephen G. Eick, Joseph L. Steffen, and Eric E Sumner, Jr. "Seesoft-A Tool For Visualizing Line Oriented Software Statistics." *IEEE Trans. Software Eng.* 18:11 (1992), 957–968.

[Elmqvist and Fekete 10] Niklas Elmqvist and Jean-Daniel Fekete. "Hierarchical Aggregation for Information Visualization: Overview, Techniques and Design Guidelines." *IEEE Transactions on Visualization and Computer Graphics* 16:3 (2010), 439–454.

[Emerson et al. 12] John Emerson, Walton Green, Barret Schloerke, Dianne Cook, Heike Hofmann, and Hadley Wickham. "The Generalized Pairs Plot.", 2012. Submitted for publication.

[Few 04] Stephen Few. *Show Me the Numbers: Designing Tables and Graphs to Enlighten.* Analytics Press, 2004.

[Frick et al. 95] A. Frick, A. Ludwig, and H. Mehldau. "A Fast Adaptive Layout Algorithm for Undirected Graphs." In *Proc. Graph Drawing (GD'94)*, LNCS, 894, LNCS, 894, pp. 388–403. Springer, 1995.

[Friendly 08] Michael Friendly. "A Brief History of Data Visualization." In *Handbook of Data Visualization*, *Computational Statistics*, edited by Antony Unwin, Chun houh Chen, and Wolfgang K. Härdle, pp. 15–56. Springer-Verlag, 2008.

[Fua et al. 99] Ying-Huey Fua, Matthew O. Ward, and Elke A. Rundensteiner. "Hierarchical Parallel Coordinates for Exploration of Large Datasets." In *Proc. IEEE Visualization Conference (Vis '99)*, pp. 43–50, 1999.

[Furnas and Bederson 95] George Furnas and Ben Bederson. "Space-Scale Diagrams: Understanding Multiscale Interfaces." In *Proc. ACM Human Factors in Computing Systems (CHI)*, pp. 234–241, 1995.

[Furnas 82] George W. Furnas. "The FISHEYE view: A new look at structured files." Technical report, Bell Laboratories Technical Memorandum #82-11221-22, 1982.

[Furnas 86] George W. Furnas. "Generalized Fisheye Views." In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 16–23, 1986.

[Furnas 06] George W. Furnas. "A Fisheye Follow-up: Further Reflection on Focus + Context." In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 999–1008, 2006.

[Ghoniem et al. 05] Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. "On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis." *Information Visualization* 4:2 (2005), 114–135.

[Gorey 63] Edward Gorey. *The Gashlycrumb Tinies; or, After the Outing.* Simon and Schuster, 1963.

[Grivet et al. 06] Sébastian Grivet, David Auber, Jean-Philippe Domenger, and Guy Melançon. "Bubble Tree Drawing Algorithm." In *Proc. Computational Imaging and Vision, Computer Vision and Graphics*, pp. 633–641, 2006.

[Grossman et al. 07] Tovi Grossman, Daniel Wigdor, and Ravin Balakrishnan. "Exploring and Reducing the Effects of Orientation on Text Readability in Volumetric Displays." In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 483–492, 2007.

[Hachul and Jünger 04] S. Hachul and M. Jünger. "Drawing Large Graphs with a Potential-Field-Based Multilevel Algorithm." In *Proc. Graph Drawing (GD'04)*, LNCS, 3383, LNCS, 3383, pp. 285–295. Springer-Verlag, 2004.

[Hansen and Johnson 05] Charles C. Hansen and Christopher R. Johnson, editors. *The Visualization Handbook.* Elsevier, 2005.

[Healey 07] Christopher G. Healey. "Perception in Vision." http://www.csc.ncsu.edu/faculty/healey/PP, 2007.

[Heer and Agrawala 06] Jeffrey Heer and Maneesh Agrawala. "Multi-Scale Banking to 45 Degrees." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2006)* 12:5 (2006), 701–708.

[Heer and Card 04] Jeffrey Heer and Stuart K. Card. "DOITrees Revisited: Scalable, Space-Constrained Visualization of Hierarchical Data." In *Proc. Advanced Visual Interfaces (AVI)*, pp. 421–424, 2004.

[Heer and Robertson 07] Jeffrey Heer and George Robertson. "Animated Transitions in Statistical Data Graphics." *IEEE Trans. on Visualization and Computer Graphics (Proc. InfoVis07)* 13:6 (2007), 1240–1247.

[Heer and Sheiderman 12] Jeffrey Heer and Ben Sheiderman. "A taxonomy of tools that support the fluent and flexible use of visualizations." *Communications of the ACM* 55:4 (2012), 45–54.

[Heer et al. 09] Jeffrey Heer, Nicholas Kong, and Maneesh Agrawala. "Sizing the Horizon: The Effects of Chart Size and Layering on the Graphical Perception of Time Series Visualizations." In *Proc. ACM Human Factors in Computing Systems (CHI)*, pp. 1303–1312, 2009.

[Henry and Fekete 06] Nathalie Henry and Jean-Daniel Fekete. "MatrixExplorer: a Dual-Representation System to Explore Social Networks." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 06)* 12:5 (2006), 677–684.

[Henry et al. 07] Nathalie Henry, Jean-Daniel Fekete, and Michael McGuffin. "NodeTrix: a Hybrid Visualization of Social Networks." *IEEE Trans. Computer Graphics and Visualization (Proc. InfoVis 2007)* 13:6 (2007), 1302–1309.

[Henze 98] Chris Henze. "Feature detection in linked derived spaces." In *IEEE Conf. Visualization (Vis)*, pp. 87–94, 1998.

[Herman et al. 00] Ivan Herman, Guy Melançon, and M. Scott Marshall. "Graph Visualisation in Information Visualisation: a Survey." *IEEE Trans. Visualization and Computer Graphics (TVCG)* 6:1 (2000), 24–44.

[Holten 06] Danny Holten. "Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data." *IEEE Transactions on Visualization and Computer Graphics (TVCG; Proc. InfoVis 2006)* 12:5 (2006), 741–748.

[Hornbæk and Hertzum 11] Kaspar Hornbæk and Morten Hertzum. "The Notion of Overview in Information Visualization." *International Journal of Human-Computer Studies* 69:7-8 (2011), 509–525.

[Hu 05] Yifan Hu. "Efficient and high quality force-directed graph drawing." *The Mathematica Journal* 10 (2005), 37–71.

[Ingram et al. 09] Stephen Ingram, Tamara Munzner, and Marc Olano. "Glimmer: Multilevel MDS on the GPU." *IEEE Trans. Visualization and Computer Graphics* 15:2 (2009), 249–261.

[Inselberg and Dimsdale 90] Alfred Inselberg and Bernard Dimsdale. "Parallel Coordinates: A Tool for Visualizing Multi-Dimensional Geometry." In *Proc. IEEE Visualization (Vis)*, 1990.

[Javed and Elmqvist 12] Waqas Javed and Niklas Elmqvist. "Exploring the Design Space of Composite Visualization." In *Proc. Pacific Visualization Symp. (PacificVis)*, pp. 1–9, 2012.

[Johansson and Johansson 09] Sara Johansson and Jimmy Johansson. "Interactive Dimensionality Reduction Through User-defined Combinations of Quality Metrics." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 09)* 15:6 (2009), 993–1000.

[Johnson and Shneiderman 91] Brian Johnson and Ben Shneiderman. "Treemaps: A Space-filling Approach to the Visualization of Hierarchical Information." In *Proceedings of IEEE Visualization '91 Conference*, pp. 284–291, 1991.

[Johnson 10] Jeff Johnson. *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules.* Morgan Kaufmann, 2010.

[Jones et al. 02] James A. Jones, Mary Jean Harrold, and John Stasko. "Visualization of Test Information to Assist Fault Localization." In *Proc. Intl. Conf. on Software Engineering (ICSE)*, pp. 467–477, 2002.

[Kadmon and Shlomi 78] Naftali Kadmon and Eli Shlomi. "A Polyfocal Projection for Statistical Surfaces." *The Cartographic Journal* 15:1 (1978), 36–41.

[Keahey and Robertson 97] T. Alan Keahey and Edward L. Robertson. "Nonlinear magnification fields." In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 51–58, 1997.

[Keahey 98] T. Alan Keahey. "The generalized detail in-context problem." In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 44–51, 1998.

[Keim and Kriegel 94] Daniel A. Keim and Hans-Peter Kriegel. "VisDB: database exploration using multidimensional visualization." *IEEE Computer Graphics and Applications* 14:5 (1994), 40–49.

[Keim 97] Daniel A. Keim. "Visual Techniques for Exploring Databases." Knowledge Discovery and Data Mining (KDD) Tutorial Program, 1997.

[Keim 00] Daniel A. Keim. "Designing Pixel-Oriented Visualization Techniques: Theory and Applications." *IEEE Trans. Visualization and Computer Graphics* 6:1 (2000), 59–78.

[Kindlmann 02] Gordon Kindlmann. "Transfer Functions in Direct Volume Rendering: Design, Interface, Interaction." SIGGRAPH 2002 Course Notes, 2002. http://www.cs.utah.edu/~gk/papers/sig02-TF-notes.pdf.

[Klippel et al. 09] Alexander Klippel, Frank Hardisty, Rui Li, and Chris Weaver. "Color Enhanced Star Plot Glyphs – Can Salient Shape Characteristics be Overcome?" *Cartographica* 44:3 (2009), 217–231.

[Kong et al. 10] Nicholas Kong, Jeffrey Heer, and Maneesh Agrawala. "Perceptual Guidelines for Creating Rectangular Treemaps." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2010)* 16:6 (2010), 990–998.

[Kuniavsky 03] Mike Kuniavsky. *Observing the User Experience: A Practitioner's Guide to User Research*. Morgan Kaufmann, 2003.

[Lam and Munzner 10] Heidi Lam and Tamara Munzner. *A Guide to Visual Multi-Level Interface Design From Synthesis of Empirical Study Evidence*. Synthesis Lectures on Visualization Series, Morgan Claypool, 2010.

[Lam et al. 06] Heidi Lam, Ronald A. Rensink, and Tamara Munzner. "Effects of 2D Geometric Transformations on Visual Memory." In *Proc. Symp. Applied Perception in Graphics and Visualization (APGV)*, pp. 119–126, 2006.

[Lam 08] Heidi Lam. "A Framework of Interaction Costs in Information Visualization." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis '08)* 14:6 (2008), 1149–1156.

[Lambert et al. 10] Antoine Lambert, David Auber, and Guy Melançon. "Living Flows: Enhanced Exploration of Edge-Bundled Graphs Based on GPU-Intensive Edge Rendering." In *Intl. Conf. Information Visualisation (IV)*, pp. 523–530, 2010.

[Larkin and Simon 87] Jill H. Larkin and Herbert A. Simon. "Why a Diagram is (Sometimes) Worth Ten Thousand Words." *Cognitive Science* 11:1 (1987), 65–99.

[Lasseter 87] John Lasseter. "Principles of traditional animation applied to 3D computer animation." *Computer Graphics (Proc. SIGGRAPH 87)* 21:4 (1987), 35–44.

[Leung and Apperley 94] Ying K. Leung and Mark Apperley. "A Review and Taxonomy of Distortion-Oriented Presentation Techniques." *ACM Trans. on Computer-Human Interaction (ToCHI)* 1:2 (1994), 126–160.

[Lopez-Hernandez et al. 10] Roberto Lopez-Hernandez, David Guilmaine, Michael J. McGuffin, and Lee Barford. "A Layer-Oriented Interface for Visualizing Time-Series Data from Oscilloscopes." In *Proc. IEEE Pacific Visualization (PacificVis)*, pp. 41–48, 2010.

[Lynch 60] Kevin Lynch. *The Image of the City*. MIT Press, 1960.

[Mackinlay et al. 90] Jock D. Mackinlay, Stuart K. Card, and George G. Robertson. "Rapid controlled movement through a virtual 3D workspace." *Computer Graphics (Proc. SIGGRAPH 90)*, pp. 171–176.

[Mackinlay et al. 91] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. "The Perspective Wall: Detail and Context Smoothly Integrated." In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 173–179, 1991.

[Mackinlay 86] Jock Mackinlay. "Automating the Design of Graphical Presentations of Relational Information." *ACM Trans. on Graphics (TOG)* 5:2 (1986), 110–141.

[McGrath 94] J.E. McGrath. "Methodology Matters: Doing Research in the Behavioral and Social Sciences." In *Readings in Human-Computer Interaction: Toward the Year 2000*, edited by R.M. Baecker, J.Grudin, and S. Greenberg W. Buxton and. Morgan Kaufmann, 1994.

[McGuffin and Balakrishnan 05] Michael J. McGuffin and Ravin Balakrishnan. "Interactive Visualization of Genealogical Graphs." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 17–24, 2005.

[McGuffin and Robert 10] Michael J. McGuffin and Jean-Marc Robert. "Quantifying the Space-Efficiency of 2D Graphical Representations of Trees." *Information Visualization* 9:2 (2010), 115–140.

[McLachlan et al. 08] Peter McLachlan, Tamara Munzner, Eleftherios Koutsofios, and Stephen North. "LiveRAC - Interactive Visual Exploration of System Management Time-Series Data." In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 1483–1492, 2008.

[Melançon 06] Guy Melançon. "Just how dense are dense graphs in the real world?: a methodological note." In *Proc AVI Workshop BEyond time and errors: novel evaLuation methods for Information Visualization (BELIV 06)*, 2006.

[Misue et al. 95] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. "Layout Adjustment and the Mental Map." *Journal of Visual Languages and Computing* 6 (1995), 183–210.

[Moscovich et al. 09] Tomer Moscovich, Fanny Chevalier, Nathalie Henry, Emmanuel Pietriga, and Jean-Daniel Fekete. "Topology-aware navigation in large networks." In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 2319–2328, 2009.

[Mukherjea et al. 96] Sougata Mukherjea, Kyoji Hirata, and Yoshinori Hara. "Visualizing the Results of Multimedia Web Search Engines." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 64–65, 1996.

[Munzner and Burchard 95] Tamara Munzner and Paul Burchard. "Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space." In *Proc. Virtual Reality Modeling Language Symposium (VRML)*, pp. 33–38. ACM SIGGRAPH, 1995.

[Munzner et al. 99] Tamara Munzner, François Guimbretière, and George Robertson. "Constellation: A Visualization Tool For Linguistic Queries from MindNet." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 132–135, 1999.

[Munzner et al. 03] Tamara Munzner, François Guimbretière, Serdar Tasiran, Li Zhang, and Yunhong Zhou. "TreeJuxtaposer: Scalable Tree Comparison Using Focus+Context With Guaranteed Visibility." *ACM Transactions on Graphics (Proc. SIGGRAPH)* 22:3 (2003), 453–462.

[Munzner 98] Tamara Munzner. "Exploring Large Graphs in 3D Hyperbolic Space." *IEEE Computer Graphics and Applications* 18:4 (1998), 18–23.

[Munzner 00] Tamara Munzner. "Interactive Visualization of Large Graphs and Networks." Ph.D. thesis, Stanford University Department of Computer Science, 2000.

[Munzner 09] Tamara Munzner. "A Nested Model for Visualization Design and Validation." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 09)* 15:6 (2009), 921–928.

[Noack 03] Andreas Noack. "An Energy Model for Visual Graph Clustering." In *Proc. Graph Drawing (GD'03)*, LNCS, 2912, LNCS, 2912, pp. 425–436. Springer-Verlag, 2003.

[Pearson 01] Karl Pearson. "On Lines and Planes of Closest Fit to Systems of Points in Space." *Philosophical Magazine* 2:6 (1901), 559–572.

[Peng et al. 04] Wei Peng, Matthew O. Ward, and Elke A. Rundensteiner. "Clutter Reduction in Multi-Dimensional Data Visualization Using Dimension Reordering." In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 89–96, 2004.

[Phan et al. 05] Doantam Phan, Ling Xiao, Ron Yeh, Pat Hanrahan, and Terry Winograd. "Flow Map Layout." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 219–224, 2005.

[Pirolli 07] Peter Pirolli. *Information Foraging Theory: Adaptive Interaction with Information*. Oxford University Press, 2007.

[Plumlee and Ware 06] M. Plumlee and C. Ware. "Zooming versus multiple window interfaces: Cognitive costs of visual comparisons." *Proc. ACM Trans. on Computer-Human Interaction (ToCHI)* 13:2 (2006), 179–209.

[Pretorius and van Wijk 09] A. Johannes Pretorius and Jarke J. van Wijk. "What does the user want to see? What do the data want to be?" *Information Visualization Journal*. Advance online publication Jun 4, doi:10.1057/ivs.2009.13.

[Rao and Card 94] Ramana Rao and Stuart K. Card. "The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information." In *Proc. ACM Human Factors in Computing Systems (CHI)*, pp. 318–322, 1994.

[Rieder et al. 08] Christian Rieder, Felix Ritter, Matthias Raspe, and Heinz-Otto Peitgen. "Interactive Visualization of Multimodal Volume Data for Neurosurgical Tumor Treatment." *Computer Graphics Forum (Proc. EuroVis 2008)* 27:3 (2008), 1055–1062.

[Roberts 07] Jonathan C. Roberts. "State of the Art: Coordinated & Multiple Views in Exploratory Visualization." In *Proc. Conf. on Coordinated & Multiple Views in Exploratory Visualization (CMV)*, pp. 61–71, 2007.

[Robertson et al. 91] George Robertson, Jock Mackinlay, and Stuart Card. "Cone Trees: Animated 3D visualizations of hierarchical information." In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 189–194, 1991.

[Robertson et al. 98] George Robertson, Mary Czerwinski, Kevin Larson, Daniel C. Robbins, David Thiel, and Maarten Van Dantzich. "Data Mountain: Using Spatial Memory for Document Management." In *Proc. User Interface Software and Technology (UIST)*, pp. 153–162, 1998.

[Robertson et al. 08] George Robertson, Roland Fernandez, Danyel Fisher, Bongshin Lee, and John Stasko. "Effectiveness of Animation in Trend Visualization." *IEEE Trans. on Visualization and Computer Graphics (Proc. InfoVis08)* 14:6 (2008), 1325–1332.

[Rogowitz and Treinish 96] Bernice E. Rogowitz and Lloyd A. Treinish. "How Not to Lie with Visualization." *Computers In Physics* 10:3 (1996), 268–273.

[Rogowitz and Treinish 98] Bernice E. Rogowitz and Lloyd A. Treinish. "Data visualization: the end of the rainbow." *IEEE Spectrum* 35:12 (1998), 52–59. Alternate version published online as *Why Should Engineers and Scientists Be Worried About Color?*, `http://www.research.ibm.com/people/l/lloydt/color/color.HTM`.

[Saraiya et al. 05] Purvi Saraiya, Chris North, and Karen Duca. "An Insight-Based Methodology for Evaluating Bioinformatics Visualizations." *IEEE Trans. Visualization and Computer Graphics (TVCG)* 11:4 (2005), 443–456.

[Schulz et al. 11] Hans-Jörg Schulz, Steffen Hadlak, and Heidrun Schumann. "The Design Space of Implicit Hierarchy Visualization: A Survey." *IEEE Trans. Visualization and Computer Graphics* 17:4 (2011), 393–411.

[Sedlmair et al. 12] Michael Sedlmair, Miriah Meyer, and Tamara Munzner. "Design Study Methodology: Reflections from the Trenches and the Stacks." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2012)*. To appear.

[Seo and Shneiderman 02] Jinwook Seo and Ben Shneiderman. "Interactively Exploring Hierarchical Clustering Results." *IEEE Computer* 35:7 (2002), 80–86.

[Seo and Shneiderman 05] Jinwook Seo and Ben Shneiderman. "A Rank-by-Feature Framework for Interactive Exploration of Multidimensional Data." *Information Visualization* 4:2 (2005), 96–113.

[Sharp et al. 07] Helen Sharp, Yvonne Rogers, and Jenny Preece. *Interaction Design: Beyond Human-Computer Interaction*. Wiley, 2007.

[Shneiderman 96] Ben Shneiderman. "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations." In *Proc. IEEE Visual Languages*, pp. 336–343, 1996.

[Simons 00] Daniel J. Simons. "Current approaches to change blindness." *Visual Cognition* 7:1/2/3 (2000), 1–15.

[Sinha and Meller 07] Amit Sinha and Jaroslaw Meller. "Cinteny: flexible analysis and visualization of synteny and genome rearrangements in multiple organisms." *BMC Bioinformatics* 8:1 (2007), 82.

[Slack et al. 06] James Slack, Kristian Hildebrand, and Tamara Munzner. "PRISAD: Partitioned Rendering Infrastructure for Scalable Accordion Drawing (Extended Version)." *Information Visualization* 5:2 (2006), 137–151.

[Slingsby et al. 09] Adrian Slingsby, Jason Dykes, and Jo Wood. "Configuring Hierarchical Layouts to Address Research Questions." *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis 2009)* 15:6 (2009), 977–984.

[Slocum et al. 08] Terry A. Slocum, Robert B. McMaster, Fritz C. Kessler, and Hugh H. Howard. *Thematic Cartography and Geovisualization*, Third edition. Prentice Hall, 2008.

[Spence and Apperley 82] Robert Spence and Mark Apperley. "Data Base Navigation: An Office Environment for the Professional." *Behaviour and Information Technology* 1:1 (1982), 43–54.

[Spence 07] Robert Spence. *Information Visualization: Design for Interaction*, Second edition. Prentice Hall, 2007.

[St. John et al. 01] Mark St. John, Michael B. Cowen, Harvey S. Smallman, and Heather M. Oonk. "The use of 2-D and 3-D displays for shape understanding versus relative position tasks." *Human Factors* 43:1 (2001), 79–98.

[Stevens 46] S. S. Stevens. "On the Theory of Scales of Measurement." *Science* 103:2684 (1946), 677–680.

[Stevens 57] S. S. Stevens. "On the psychophysical law." *Psychological Review* 64:3 (1957), 153–181.

[Stevens 75] S. S. Stevens. *Psychophysics: Introduction to its Perceptual, Neural, and Social Prospects*. Wiley, 1975.

[Stolte et al. 99] Chris Stolte, Robert Bosch, Pat Hanrahan, and Mendel Rosenblum. "Visualizing Application Behavior on Superscalar Processors." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 10–17, 1999.

[Stolte et al. 08] Chris Stolte, Diane Tang, and Pat Hanrahan. "Polaris: a system for query, analysis, and visualization of multidimensional databases." *Commun. ACM* 51:11 (2008), 75–84.

[Stone 03] Maureen Stone. *A Field Guide to Digital Color*. A K Peters, 2003.

[Stone 05] Maureen Stone. "Representing Colors as Three Numbers." *Computer Graphics and Applications* 25:4 (2005), 78–85.

[Stone 06] Maureen Stone. "Color In Information Display.", 2006. Http://www.stonesc.com/Vis06.

[Stone 10] Maureen Stone. "Get it right in black and white." http://www.stonesc.com/wordpress/2010/03/get-it-right-in-black-and-white/, 2010.

[Telea 07] Alexandru Telea. *Data Visualization: Principles and Practice*. AK Peters, 2007.

[Torgerson 52] W. S. Torgerson. "Multidimensional Scaling: I. Theory and Method." *Psychometrika* 17 (1952), 401–419.

[Tory and Möller 04a] Melanie Tory and Torsten Möller. "Human Factors in Visualization Research." *IEEE Trans. Visualization and Computer Graphics (TVCG)* 10:1 (2004), 72–84.

[Tory and Möller 04b] Melanie Tory and Torsten Möller. "Rethinking Visualization: A High-Level Taxonomy." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 151–158, 2004.

[Tory and Möller 05] Melanie Tory and Torsten Möller. "Evaluating Visualizations: Do Expert Reviews Work?" *IEEE Computer Graphics and Applications* 25:5.

[Treisman and Gormican 88] Anne Treisman and Stephen Gormican. "Feature analysis in early vision: Evidence from search asymmetries." *Psychological Review* 95:1 (1988), 15–48.

[Tufte 83] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 1983.

[Tufte 91] Edward Tufte. *Envisioning Information.* Graphics Press, 1991.

[Tufte 97] Edward R. Tufte. *Visual Explanations.* Graphics Press, 1997.

[Tukey 77] John W. Tukey. *Exploratory Data Analysis.* Addison-Wesley,, 1977.

[Tversky et al. 02] Barbara Tversky, Julie Morrison, and Mireille Betrancourt. "Animation: Can It Facilitate?" *International Journal of Human Computer Studies* 57:4 (2002), 247–262.

[Tversky 92] Barbara Tversky. "Distortions in Cognitive Maps." *Geoforum* 23:2 (1992), 131–138.

[Valiati et al. 06] Eliane Valiati, Marcelo Pimenta, and Carla M.D.S. Freitas. "A Taxonomy of Tasks for Guiding the Evaluation of Multidimensional Visualizations." In *Proc. AVI Workshop on BEyond time and errors: novel evaLuation methods for Information Visualization (BELIV)*. ACM, 2006. Article 15.

[van Ham 03] Frank van Ham. "Using Multilevel Call Matrices in Large Software Projects." In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 227–232, 2003.

[van Wijk and Nuij 03] Jarke J. van Wijk and Wim A. A. Nuij. "Smooth and efficient zooming and panning." In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 15–22, 2003.

[van Wijk and van Selow 99] Jarke J. van Wijk and Edward R. van Selow. "Cluster and Calendar based Visualization of Time Series Data." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 4–9, 1999.

[van Wijk 06] Jarke J. van Wijk. "Bridging the Gaps." *IEEE Computer Graphics & Applications* 26:6 (2006), 6–9.

[von Landesberger et al. 10] Tatiana von Landesberger, Arjan Kuijper, Tobias Schreck, Jörn Kohlhammer, Jarke J. van Wijk, Jean-Daniel Fekete, and Dieter W. Fellner. "Visual Analysis of Large Graphs." *Proc. Eurographics*.

[Wainer and Francolini 80] Howard Wainer and Carl M. Francolini. "An Empirical Inquiry Concerning Human Understanding of Two-Variable Color Maps." *The American Statistician* 34:2 (1980), 81–93.

[Ward et al. 10] Matthew O. Ward, Georges Grinstein, and Daniel Keim. *Interactive Data Visualization: Foundations, Techniques, and Applications*. AK Peters, 2010.

[Ward 02] Matthew O. Ward. "A Taxonomy of Glyph Placement Strategies for Multidimensional Data Visualization." *Information Visualization Journal* 1:3-4 (2002), 194–210.

[Ward 08] Matthew O. Ward. "Multivariate Data Glyphs: Principles and Practice." In *Handbook of Data Visualization*, *Computational Statistics*, edited by Antony Unwin, Chun houh Chen, and Wolfgang K. Härdle, pp. 179–198. Springer-Verlag, 2008.

[Ware et al. 02] Colin Ware, Helen Purchase, Linda Colpys, and Matthew McGill. "Cognitive Measures of Graph Aesthetics." *Information Visualization* 1:2 (2002), 103–110.

[Ware 01] Colin Ware. "Designing with a 2 1/2 D Attitude." *Information Design Journal* 10:3 (2001), 255–262.

[Ware 04] Colin Ware. *Information Visualization: Perception for Design*, Second edition. Morgan Kaufmann/Academic Press, 2004.

[Ware 08] Colin Ware. *Visual Thinking for Design*. Morgan Kaufmann, 2008.

[Wattenberg 06] Martin Wattenberg. "Visual exploration of multivariate graphs." In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)*, pp. 811–819, 2006.

[Weaver 04] Chris Weaver. "Building Highly-Coordinated Visualizations In Improvise." In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 159–166, 2004.

[Weaver 10] Chris Weaver. "Cross-Filtered Views for Multidimensional Visual Analysis." *IEEE Trans. Visualization and Computer Graphics* 16:2 (2010), 192–204.

[Wegman 90] Edward J. Wegman. "Hyperdimensional Data Analysis Using Parallel Coordinates." *Journ. American Statistical Association* 85:411 (1990), 664–675.

[Wickham and Hofmann 11] Hadley Wickham and Heike Hofmann. "Product Plots." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2011)*.

[Wickham and Stryjewski 12] Hadley Wickham and Lisa Stryjewski. "40 years of boxplots." *The American Statistican*.

[Wilkinson and Friendly 09] Leland Wilkinson and Michael Friendly. "The History of the Cluster Heat Map." *The American Statistician* 63:2 (2009), 179–184.

[Wilkinson and Wills 08] Leland Wilkinson and Graham Wills. "Scagnostics Distributions." *Journal of Computational and Graphical Statistics (JCGS)* 17:2 (2008), 473–491.

[Wilkinson et al. 05] Leland Wilkinson, Anushka Anand, and Robert Grossman. "Graph-Theoretic Scagnostics." In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 157–164, 2005.

[Wilkinson et al. 06] Leland Wilkinson, Anushka Anand, and Robert Grossman. "High-Dimensional Visual Analytics: Interactive Exploration Guided by Pairwise Views of Point Distributions." *IEEE Transactions on Visualization and Computer Graphics* 12:6 (2006), 1363–1372.

[Wilkinson 05] Leland Wilkinson. *The Grammar of Graphics*, Second edition. Springer-Verlag, 2005.

[Willett et al. 07] Wesley Willett, Jeffrey Heer, and Maneesh Agrawala. "Scented Widgets: Improving Navigation Cues with Embedded Visualizations." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2007)* 13:6 (2007), 1129–1136.

[Williams 08] Robin Williams. *The Non-Designer's Design Book*, Third edition. Peachpit Press, 2008.

[Wills 95] Graham J. Wills. "Visual Exploration of Large Structured Datasets." In *Proc. New Techniques and Trends in Statistics (NTTS)*, pp. 237–246. IOS Press, 1995.

[Wills 08] Graham J. Wills. "Linked Data Views." In *Handbook of Data Visualization*, *Computational Statistics*, edited by Antony Unwin, Chun houh Chen, and Wolfgang K. Härdle, pp. 216–241. Springer-Verlag, 2008.

[Yang et al. 02] Jing Yang, Matthew O. Ward, and Elke A. Rundensteiner. "InterRing: An Interactive Tool for Visually Navigating and Manipulating Hierarchical Structures." In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 77–84, 2002.

[Yang et al. 03a] Jing Yang, Wei Peng, Matthew O. Ward, and Elke A. Rundensteiner. "Interactive Hierarchical Dimension Ordering, Spacing and Filtering for Exploration Of High Dimensional Datasets." In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 105–112, 2003.

[Yang et al. 03b] Jing Yang, Matthew O. Ward, Elke A. Rundensteiner, and Shiping Huang. "Visual Hierarchical Dimension Reduction for Exploration of High Dimensional Datasets." In *Proc. Eurographics/IEEE Symp. Visualization (VisSym)*, pp. 19–28, 2003.

[Yang et al. 04] Jing Yang, Anilkumar Patro, Shiping Huang, Nishant Mehta, Matthew O. Ward, and Elke A. Rundensteiner. "Value and relation display for interactive exploration of high dimensional datasets." In *Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 73–80, 2004.

[Yi et al. 07] Ji Soo Yi, Youn Ah Kang, John T. Stasko, and Julie A. Jacko. "Toward a Deeper Understanding of the Role of Interaction in Information Visualization." *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis '07)* 13:6 (2007), 1224–1231.

[Young and Householder 38] G. Young and A. S. Householder. "Discussion of a set of points in terms of their mutual distances." *Psychometrika* 3:1.

[Zacks and Tversky 99] Jeff Zacks and Barbara Tversky. "Bars and Lines: A Study of Graphic Communication." *Memory and Cognition* 27:6 (1999), 1073–1079.

[Zhang and Norman 95] Jiajie Zhang and Donald A. Norman. "A Representational Analysis of Numeration Systems." *Cognition* 57 (1995), 271–295.

[Zhang 97] Jiajie Zhang. "The Nature of External Representations in Problem Solving." *Cognitive Science* 21:2 (1997), 179–217.

[Zuk et al. 08] Torre Zuk, Lothar Schlesier, Petra Neumann, Mark S. Hancock, and Sheelagh Carpendale. "Heuristics for Information Visualization Evaluation." In *Proc. AVI Workshop on BEyond time and errors: novel evaLuation methods for Information Visualization (BELIV)*. ACM, 2008. Article 9.