A grammar of graphics

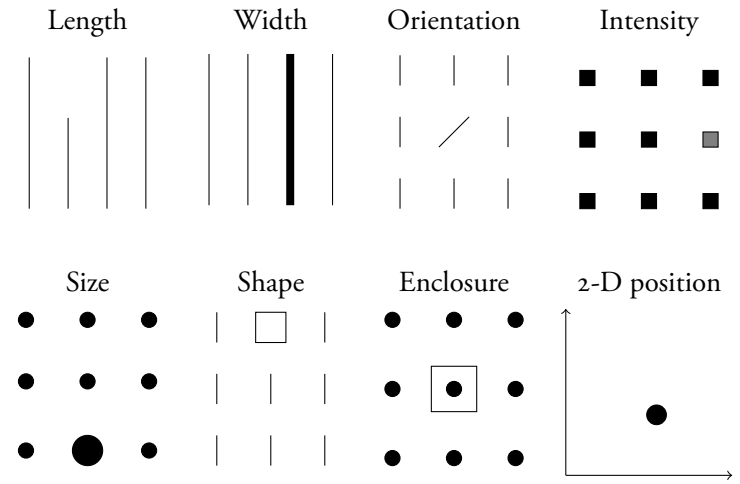
2012 November 9

Kevin J. Lynagh

Keming Labs

Data visualization is the mapping of quantitative values—sales by region, hits per second, CPU utilization—to visual aesthetics such as position, shape, size, and color. To effectively communicate data visually, you must understand basic rules about the human perceptual system. In particular, that we are better at "seeing" quantitative values encoded in certain aesthetics versus others. For instance, we can readily estimate how much longer one object is than another, whereas it is much more difficult to estimate relative areas. Some aesthetics do not even have a natural quantitative value (e.g., which is greater: ◯ or □ ?).

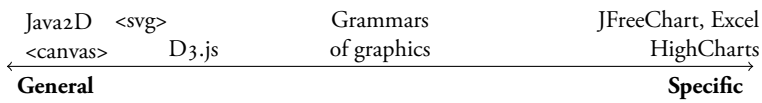
To make a good visualization, you must first decide *what* you want to communicate and then *how* to best show that data visually. The tables below show the most commonly used visual aesthetics. Position and length are two of the most effective visual aesthetics, so it's difficult to beat a simple bar chart, scatter plot, or line graph. Aesthetics can be combined to show multiple data dimensions at once, but techniques like small multiples tend to be more effective.



Attribute	Quantitative?
Form	
Length	Yes
Width	Limited
Orientation	No
Size	Limited
Shape	No
Enclosure	No
Color	
Hue	No
Intensity	Limited
Position	
2-D position	Yes

Table and figures adapted from Stephen Few's *Show Me the Numbers*. Few's book is an excellent introduction for practitioners. For a more in-depth review of human perceptual research, see *Information Visualization: Perception for Design*, by Colin Ware.

Balancing expressiveness with ease of use and simplicity is a central tension of programming. This tension informs the design of statistical graphics libraries, which can be arranged on a continuum:



At one extreme you draw directly on the underlying canvas or scenegraph via low-level calls to, e.g, Java2D's or HTML's `drawRect()` and `drawLine()` methods. This gives the most flexibility, at the great cost of doing all of the hard work yourself to iterate over data, choose tick marks, build legends, and so on. At the other extreme, your library provides "canned" graphics. This is very easy, as long as the library provides a visualization for your exact needs—just call the `barChart()` function and you're done. However, if you want to customize something; draw a bar graph with error bars, draw a best-fit curve through a scatter plot, or highlight a single datum red to call it out, and the library does not allow you to do it, you are out of luck.

This handout accompanies *Building a grammar of graphics with Clojure*, a talk at Oredev 2012 in Malmö, Sweden.

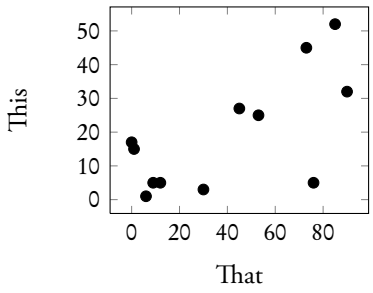
Slides: `keminglabs.com/talks/`
Email: `kevin@keminglabs.com`
Twitter: `@lynaghk`
Office: +1 888 502 1042

A grammar of graphics explores the middle ground: it can never be as expressive as directly manipulating the low-level scenegraph, but aims to be more flexible than a fixed-menu of chart types. The grammar splits a graphic apart into orthogonal pieces:

- Data: typically rectangular (an array of maps or a SQL table).
- Geometry: the visual representation onto which the data will be mapped: points, lines, polygons, &c. Each "geom" has its own set of aesthetics: points have 2-D position, bars have length, and so on.
- Grouper: (optionally) subset the data into different groups.
- Statistic: (optionally) transform or summarize the grouped data.
- Mapping: maps the data dimensions (or the computed values of the statistic) onto the geom's aesthetics.

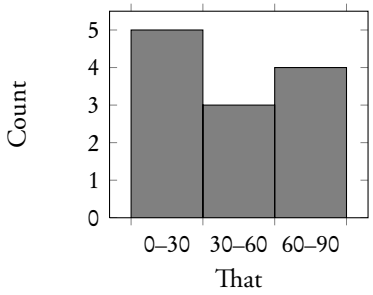
The Grammar of Graphics was originally a book by Leland Wilkinson. Wilkinson's ideas became popular largely through Hadley Wickham's `ggplot2` library for the R statistical programming language, which implements a simplified "layered grammar". This talk outlines `C2PO`, a variation on the layered grammar. (The name is tentative—Hadley insisted on a *Star Wars*-themed name to match his current `R2D3` project `=P`)

We can concisely describe a scatter plot using the grammar:




```
{:data [{:this 15 :that 1}, ...]
 :geom #point {:fill "black"}
 :mapping {:y :this :x :that}}
```

Now consider a histogram: a set of bars showing the number of data that occur in a certain range along the abscissa (x-axis). Rather than use some special `histogram` function, we describe what we want using the grammar:




```
{:data [{:this 15 :that 1}, ...]
 :mapping {:y :this :x :that}
 :geom #bar {:fill "gray"}
 :stat :sum
 :group #bin {:num-bins 3}}
```

Of course, the real power of a grammar of graphics is not to create graphics that we already have names for. Rather, it is to provide a framework with which we can create effective graphics tailored to our specific data sets. Breaking apart the concerns of visual geometry, data grouping, and statistics allows us to reconfigure the pieces in powerful, novel combinations.

This handout is a very brief and necessarily incomplete introduction to data visualization. I hope that it serves as a humble starting point for you to explore what I find to be very exciting topics. Good luck!

D3: JavaScript library for bespoke data graphics	`http://d3js.org`
C2: Clojure(Script) port of D3	`https://github.com/lynaghk/c2`
`ggplot2`: R grammar of graphics	`http://ggplot2.org`
`c2po`: Grammar of graphics on the JVM	`http://keminglabs.com/c2po/`
Mark Volkmann's Clojure intro:	`http://java.ociweb.com/mark/clojure/article.html`
Himera: try ClojureScript directly in your browser:	`http://himera.herokuapp.com/index.html`
The Joy of Clojure, by Houser and Fogus	`http://joyofclojure.com/`
Clojure/core's TV channel	`http://blip.tv/clojure`