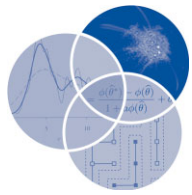
Advanced Review

The grammar of graphics

Leland Wilkinson*



The grammar of graphics (GoG) denotes a system with seven classes embedded in a data flow. This data flow specifies a strict order in which data are transformed from a raw dataset to a statistical graphic. Each class contains multiple methods, each of which is a function executed at the step in the data flow corresponding to that class. The classes are orthogonal, in the sense that the product set of all classes (every possible sequence of class methods) defines a space of graphics which is meaningful at every point. The meaning of a statistical graphic is thus determined by the mapping produced by the function chain linking data and graphic. *WIREs Comp Stat* 2010 2 673–677 DOI: 10.1002/wics.118

Keywords: visualization; statistical graphics

INTRODUCTION

The grammar of graphics (GoG) denotes a system with seven orthogonal classes.[1] The term *orthogonal* means that each class contains one or more methods (functions) as elements, and all tuples in the seven-fold product of these sets of functions produce meaningful graphs. A consequence of this orthogonality is a high degree of expressiveness: we can produce a huge variety of graphical forms or chart types in such a system. In fact, it is claimed that virtually the entire corpus of known charts can be generated by this relatively parsimonious system, and perhaps a great number of meaningful but undiscovered chart types as well.

A second claim of GoG is that this system describes the *meaning* of what we do when we construct statistical graphics. It is more than a taxonomy. It is a computational system based on the underlying mathematics of representing statistical functions of data.

THE GoG DATA FLOW

Figure 1 shows a *data flow* diagram that contains the seven GoG classes. This data flow is a chain that describes the sequence of mappings needed to produce a statistical graphic from a set of data. The first class (Variables) maps data to an object called a *varset* (a set of variables). The next two classes (Algebra, Scales) are transformations on varsets. The next class (Statistics) takes a varset and creates a statistical graph (a statistical summary). The next class (Geometry) maps a statistical graph to a geometric graph. The next (Coordinates) embeds a graph in a coordinate space. And the last class (Aesthetics) maps a graph to a visible or perceivable display called a graphic.

The data flow architecture implies that the subtasks needed to produce a graphic from data must be done in this specified order. Changes to this ordering can produce meaningless graphics. For example, if we compute certain statistics on variables (e.g., sums) before scaling them (e.g., log scales), we can produce statistically meaningless results because the log of a sum is not the sum of the logs.

The data flow in Figure 1 has many paths through it because of the multiple methods (functions) in each stage. We can choose different algebraic designs (factorial, nested, . . .), scales (log, probability, . . .), statistical methods (means, medians, modes, smoothers, . . .), geometric objects (points, lines, bars, . . .), coordinate systems (rectangular, polar, . . .), and aesthetics (size, shape, color, . . .). These paths reveal the richness of the system.

Variables

We begin with data. We assume the data that we wish to graph are organized in one or more tables. The columns of each table are called *fields*, with each field containing a set of measurements or attributes. The rows of each table are called *records*, with each record containing the measurements of an object on each field. Usually, a relational database management system (RDBMS) produces such a table from organized queries specified in structured query language (SQL) or some other relational language. Other data sources (object, streaming, . . .) are mapped to tables through similar methods.

*Correspondence to: Leland.wilkinson@gmail.com

Department of Computer Science, University of Illinois at Chicago, Chicago, IL, USA

DOI: 10.1002/wics.118

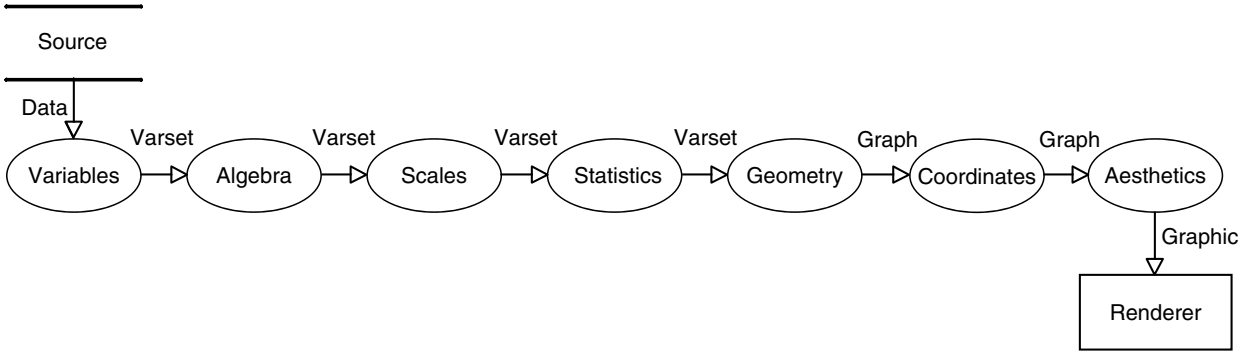


FIGURE 1 | The grammar of graphics data flow.

Our first step is to convert a table of data to a *varset*. A varset is a set of one or more variables. While a column of a table of data might superficially be considered to be a variable, there are differences. A variable is both more general (in regard to generalizability across samples) and more specific (in regard to data typing and other constraints) than a column of data. First, we define a variable, then a varset.

Variable

A *variable* X is a mapping $f: O \to V$, which we consider as a triple:

$$X = [O, V, f].$$

The domain O is a set of objects.
The codomain V is a set of values.
The function f assigns to each element of O an element in V.

The image of O under f contains the *values* of X. We denote a possible value as x, where $x \in V$. We denote a value of an object as $X(o)$, where $o \in O$. A variable is *continuous* if V is an interval. A variable is *categorical* if V is a finite subset of the integers (or there exists an injective map from V to a finite subset of the integers).

Variables may be multidimensional. $\boldsymbol{X}$ is a p-dimensional variable made up of p one-dimensional variables:

$$\begin{aligned}\boldsymbol{X} &= (X_1, \ldots, X_p) \\ &= [O, V_i, f\,], \quad i = 1, \ldots, p \\ &= [O, \boldsymbol{V}, f\,].\end{aligned}$$

The element $\boldsymbol{x} = (x_1, \ldots, x_p)$, $\boldsymbol{x} \in \boldsymbol{V}$, is a p-dimensional value of $\boldsymbol{X}$. We use multidimensional variables in *multivariate analysis*.

Varset

We call the triple

$$\mathrm{X} = [V, \widetilde{\mathrm{O}}, f]$$

a *varset*. The word *varset* stands for *variable set*. If X is multidimensional, we use boldface $\boldsymbol{X}$. A varset inverts the mapping used for variables. That is,

The domain V is a set of values.
The codomain $\widetilde{O}$ is a set of all possible ordered lists of objects.
The function f assigns to each element of V an element in $\widetilde{O}$.

We invert the mapping customarily used for variables in order to simplify the definitions of graphics algebra operations on varsets. In doing so, we also replace the variable's set of objects with the varset's set of ordered lists. We use lists in the codomain because it is possible for a value to be mapped to an object more than once (as in repeated measurements).

Algebra

Given one or more varsets, we now need to operate on them to produce combinations of variables. A typical scatterplot of a variable X against a variable Y, for example, is built from tuples (x_i, y_i) that are elements in a set product. We use graphics algebra on values stored in varsets to make these tuples. There are three binary operators in this algebra: *cross*, *nest*, and *blend*.

Cross $(*)$

Cross joins the left argument with the right to produce a set of tuples stored in the multiple columns of the new varset:

x
y
z

$*$

a
a
b

$=$

x	a
y	a
z	b

The resulting set of tuples is a subset of the product of the domains of the two varsets. The domain of a varset produced by a cross is the product of the separate domains. One may think of a cross as a horizontal concatenation of the table representation of two varsets, assuming the rows of each varset are equivalent and in the same order.

Nest (/)

Nest partitions the left argument using the values in the right:

x		*a*		*x*	*a*
y	/	*a*	=	*y*	*a*
z		*b*		*z*	*b*

Although it is not required in the definition, we assume the nesting varset on the right is categorical. The name *nest* comes from design-of-experiments terminology. We often use the word *within* to describe its effect. For example, if we assess schools and teachers in a district, then *teachers within schools* specifies that teachers are nested within schools. Assuming each teacher in the district teaches at only one school, we would conclude that if our data contain two teachers with the same name at different schools, they are different people.

Blend (+)

Blend produces a union of varsets:

x		*a*		*x*
y	+	*a*	=	*y*
z		*b*		*z*
				a
				a
				b

Blend is defined only if the order of the tuples (number of columns) in the left and right varsets is the same. Furthermore, we need to restrict blend to varsets with composable domains. It would make little sense to blend Age and Weight, much less Name and Height. The Scales class, in the next section, throws an exception if we attempt to blend varsets across different types of scales.

Scales

Before we compute summaries (totals, means, smoothers, ...) and represent these summaries using geometric objects (points, lines, ...), we must scale our varsets. In producing most common charts, we do not notice this step. When we implement log scales, however, we notice it immediately. We must log our data before averaging logs. Even if we do not compute nonlinear transformations, however, we need to specify a measurement model.

The measurement model determines how distance in a frame region relates to the ranges of the variables defining that region. Measurement models are reflected in the axes, scales, legends, and other annotations that demarcate a chart's frame. Measurement models determine how values are represented (e.g., as categories or magnitudes) and what the units of measurement are.

In constructing scales for statistical charts, we need to know the function used to assign values to objects. S.S. Stevens developed a taxonomy of such functions based on axioms of measurement.[2] Stevens identified four basic scale types: *nominal*, *ordinal*, *interval*, and *ratio*. These scales are widely cited in introductory statistics books and in some visualization schemes.[3]

For graphics grammar, we employ a more restrictive classification based on units of measurement. Unit scales permit standardization and conversion of metrics and raise exceptions when improper blends are attempted. The International System of Units (SI) unifies measurement under transformation rules encapsulated in a set of base classes.[4] Most of the measurements in the SI system fit within the interval and ratio levels of Stevens' system. There are other scales fitting Stevens' system that are not classified within the SI system, however. These involve units such as *category* (state, province, country, color, species, ...), *order* (rank, index), and *measure* (probability, proportion, percent, ...). Also, there are some additional scales that are in neither the Stevens nor the SI system, such as *partial order*.

Statistics

The statistics component receives a varset, computes statistical summaries, and outputs another varset. In the simplest case, the statistical method is an identity. We do this for scatterplots. Data points are input and the same data points are output. In other cases, such as histogram binning, a varset with n rows is input and and a varset with k rows is output, where k is the number of bins ($k < n$). With smoothers (regression or interpolation), a varset with n rows is input and a varset with k rows is output, where k is the number of knots in a mesh over which smoothed values are computed. With point summaries (means, medians, ...), a varset with n rows is input and a varset with one row is output. With regions (confidence intervals, ranges, ...), a varset with n rows is input and a varset with two rows is output.

Statistical methods are members of their own object, so they are independent of the other elements in the system. There is no necessary connection between regression methods and curves or between confidence intervals and error bars or between histogram binning and histograms. We can represent the same statistic with a variety of different geometric objects.

Geometry

Geometric graphs are produced by graphing functions $F: B^m \to \mathbb{R}^n$ that injectively map an m-dimensional bounded region to an n-dimensional real space and that have geometric names like *line*() or *bar*(). A geometric graph is the image of F. Geometric graphs are not visible; they are geometric sets.

- The *point*() graphing function produces a geometric point, which is an n-tuple. This function can also produce a finite set of points, called a *multipoint* or a *point cloud*. The set of points produced by *point*() is called a *point* graph.
- The *line*() graphing function is a bit more complicated. Let B^m be a bounded region in $\mathbb{R}^m$. Consider the function $F: B^m \to \mathbb{R}^n$, where $n = m + 1$, with the following additional properties:

  1. the image of F is bounded, and
  2. $F(x) = (\boldsymbol{v}, f(\boldsymbol{v}))$, where $f: B^m \to \mathbb{R}$ and $\boldsymbol{v} = (x_1, \ldots, x_m) \in B^m$.

  If $m = 1$, this function maps an interval to a functional curve on a bounded plane; and if $m = 2$, it maps a bounded region to a functional surface in a bounded 3D space. The *line*() graphing function produces these graphs. Like *point*(), *line*() can produce a finite set of lines. A set of lines is called a multiline. We need this capability for representing multimodal smoothers, confidence intervals on regression lines, and other multifunctional lines.
- The *area*() graphing function produces a graph containing all points within the region under the *line* graph.
- The *path*() graphing function is similar to a line, but it is not ordered on x. A *path*() produces a path that connects points such that each point touches no more than two line segments. Thus, a path visits every point in a collection of points only once. If a path is closed (every point touches two line segments), we call it a circuit.
- The *bar*() or, equivalently, *interval()* graphing function produces a set of closed intervals. An interval has two ends. Ordinarily, however, bars are used to denote a single value through the location of one end. The other end is anchored at a common reference point (usually zero).
- A *schema* is an element that includes both general and particular features in order to represent a distribution. We have taken this usage from Tukey,[5] who invented the schematic plot, which has come to be known as the box plot because of its physical appearance. The *schema*() graphing function produces a collection of one or more points and intervals.
- The *polygon*() graphing function partitions a surface or space with polygons. Examples are tessellations and boundary polygons in maps.
- A *contour*() graphing function produces contours, or level curves. A contour graph is used frequently in weather and topographic maps. Contours can be used to delineate any continuous surface.
- The *edge*() graphing function joins points with line segments (edges). Although edges join points, a point graph is not needed in a frame in order to make an edge.

Coordinates

Most popular charts employ Cartesian coordinates. The same real tuples in the graphs underlying these graphics can be embedded in many other coordinate systems, however. Examples are polar, fisheye, and geographic (spherical) projections. The GoG coordinates object is a rich utility for radically changing the appearance of a statistical graphic. A software system implementing GoG can change a statistical graphic's form in one line of code.

Many peculiar names are popularly given to charts that are simple coordinate transformations of other popular charts. Spider (radar) charts are polar parallel coordinate plots. Rose charts are polar bar charts. Pie charts are polar divided bar charts. Sparklines are similarity transformations of ordinary line charts.

Aesthetics

An *aesthetic* is a function that maps a graph to a perceivable graphic. Seven of the aesthetic functions in GoG are derived from Bertin's *visual variables*[6]: *position* (position), *size* (taille), *shape* (forme), *orientation* (orientation), *brightness* (valeur), *color* (couleur), and *granularity* (grain). In GoG, color

is separated into three components. Additional GoG aesthetics involve dimensions such as blur, sound, and motion.

SOFTWARE

Wilkinson[1] states that GoG is not a taxonomy or a semiotic system. It is claimed to be *the* mathematical foundation of statistical graphics. It is also claimed that statistical graphics differ fundamentally from other visualizations such as maps, charts, diagrams, and volumetric 3D realizations. Furthermore, GoG is claimed to be *the* appropriate class hierarchy for an object-oriented graphics system capable of producing statistical graphics.

Consequently, software systems that implement GoG have classes named Algebra, Scales, Statistics, Geometry, Coordinates and Aesthetics. These classes have multiple methods to produce the rich variety of statistical graphics. Implementing these classes so that they preserve orthogonality of their methods is difficult. Testing orthogonality is easy, however. In a correctly designed GoG system, we ought to be able to combine any method in any class with any method in any other class to produce a meaningful graphical result. Those implementing GoG have discovered that some of these combinations produce novel and intriguing visualizations. And Graham Wills has demonstrated that GoG can produce in a simple specification some of the most elaborate statistical visualizations presented at the InfoVis conference in recent years.[7]

There have been several software systems based on the GoG foundation. The earliest was the Java Graphics Production Language GPL (not to be confused with the Gnu Public License GPL) developed by Rope and Wilkinson.[8] GPL was intended to complement the Bureau of Labor Statistics' Table Producing Language TPL. GPL was eventually marketed by Illumitek Inc. under the name nViZn and the GPL language was incorporated into the first edition of GoG.

Rope, Wills, Dubbs, Wilkinson, and other members of the visualization team at SPSS rewrote nViZn to incorporate an XML interface and to refine GPL. This system serves as the underlying graphic engine for various IBM analytic products. The visualization team implemented a GPL interpreter that can execute examples from the second edition of the book. The interpreter is rather well-hidden among the many programming options inside SPSS.

The Polaris system at Stanford[9] implemented a GoG architecture for producing visualizations from a multidimensional relational database. This software evolved into the Tableau visualization package distributed by Tableau Software and used in the Hyperion OLAP.

Hadley Wickham has developed a GoG package called ggplot2 for the R platform.[10] It implements many of the graphics found in Ref 1 using a simple functional language.

Finally, the Protovis project at Stanford[11] is implementing most of the grammar in a Web-based platform that uses Javascript for specifying geometric elements. This system makes it especially easy to insert statistical graphics into Web pages.

CONCLUSION

The GoG is both the title of a book and a term for an object-oriented graphics system for creating and automating statistical graphics.

REFERENCES

1. Wilkinson L. *The Grammar of Graphics*. 2nd ed. New York: Springer-Verlag; 2005.
2. Stevens SS. On the theory of scales of measurement. *Science* 1946, 103:677–680.
3. Velleman P, Wilkinson L. Nominal, ordinal, interval, and ratio typologies are misleading for classifying statistical methodology. *Am Stat* 1993, 47:65–72.
4. Taylor BN, Thompson A (eds). *The International System of Units (SI)*. Gaithesburg, MD: National Institute of Standards and Technology; 2008.
5. Tukey JW. *Exploratory Data Analysis*. Reading, MA: Addison-Wesley Publishing Company; 1977.
6. Bertin J. *Sèmiologie Graphique*. Paris: Editions GauthierVillars; 1967.
7. Wills G, Zhang C. Grammatical visualization of statistical models. Joint Statistical Meetings, Salt Lake City, UT, 2007.
8. Wilkinson L, Rope DJ, Carr DB, Rubin MA. The language of graphics. *J Comput Graph Stat* 2000, 9:530–543.
9. Stolte C, Tang D, Hanrahan P. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans Vis Comput Graph* 2002, 8:52–65.
10. Wickham H. *ggplot2*. New York: Springer-Verlag; 2009.
11. Bostock M, Heer J. Protovis: a graphical toolkit for visualization. *Proceedings of the IEEE Information Visualization Conference*, Atlantic City, NJ, 2009.

