The GGobi data pipeline

Michael Lawrence

A data pipeline transforms data, through a series of stages, into visualizations. Buja et al. [1] introduce the use of the pipeline design pattern for statistical data visualization. Figure 1 gives an overview of the design. They argue that an interactive and dynamic visualization requires real-time data processing. Their suggested pipeline is constituted by seven stages. The pipeline begins with the raw data. The second stage performs non-linear transformations on the data, if requested. The next stage standardizes the variables, and the following stage randomizes variables, so that they may serve as a graphical permutation test. After randomization comes the projection engine, which reduces the dimensionality of the data, either by selecting variables for the axes or projecting multiple variables onto a lower dimensional space, as in tours. The viewporting stage is next; it decides the visible range and scale of the data, which may be specified by a user through pan, zoom and scale controls. The final stage is the actual plot. Thus, the pipeline incorporates many of the common needs of dynamic statistical graphics: transformation, standardization, permutation, projection, and viewporting.

The Orca project [2] extends the Buja et al. pipeline to support multiple linked views. The linked views adhere to a model view controller (MVC) pattern, in that user manipulation, such as brushing, changes values in the underlying OrcaAppearance model, and these changes are then propagated to any views, normally plots, that are observers of the model. Although the views are linked, each view has a separate instance of the pipeline for the data it displays. Orca also adds special handling of variables based on their type. For example, when creating time series plots, the time variable is not grouped with the others during dimensionality reduction. The Orca pipeline implementation is object-oriented and follows many established software design patterns.

At the core of GGobi is a data pipeline based on the ideas of Buja et al. and Orca. It is outlined in Figure 1. The GGobi stages are raw, transform, world, planar, and screen, roughly corresponding to the Buja et al. stages raw, non-linear transformation, variable standardization, projection engine, and viewporting, respectively. The randomization step is omitted. The world stage, a conversion from real numbers to integers, only loosely maps to standardization. GGobi supports multiple views and improves on its predecessor Orca in that views of the same dataset share the same pipeline instance until the world stage, where the pipeline separates into branches for each plot. A more important innovation over Orca is that the GGobi pipeline is able to operate in reverse. This allows, among other things, the user to modify the raw data by interacting with a plot.

The most blatant deficiency of the existing GGobi pipeline is its lack of extensibility. GGobi is in some ways a regression from Orca with regard to its software design. Currently, the data is processed by invoking functions, corresponding to the pipeline stages, in order. The pipeline stages are not encapsulated as modules. This inflexibility has forced the categorical data processing in GGobi to exist outside of the pipeline, stunting the growth of categorical features. Plugins are unable to directly integrate their functionality with the pipeline. A redesigned pipeline, based on modern object oriented techniques, as in Orca, is necessary for GGobi to keep pace with the rapidly evolving field of statistical graphics.

1 The new modular pipeline

The central goal of the new pipeline design is modularity. The functions carrying the data from one stage to the next in the original pipeline are encapsulated into reusable and interchangeable modules that interconnect to form an emergent pipeline. This is reminiscent of the modular approach taken by Orca. The purpose of a module is to modify an input data matrix in some way, such as appending a column, filtering out a row, or transforming a value. A module must also reverse its modifications when changes to the data are sent back down the pipeline towards the root. Each module is independent of the others; it is completely unaware of both its position and the identity of the other modules in the pipeline. The functionality shared by all pipeline modules has a uniform interface across modules, so

Figure 1: Statistical graphics pipelines. The original pipeline by Buja et al. [1] is juxtaposed against the current GGobi pipeline. The analogous stages are aligned vertically.

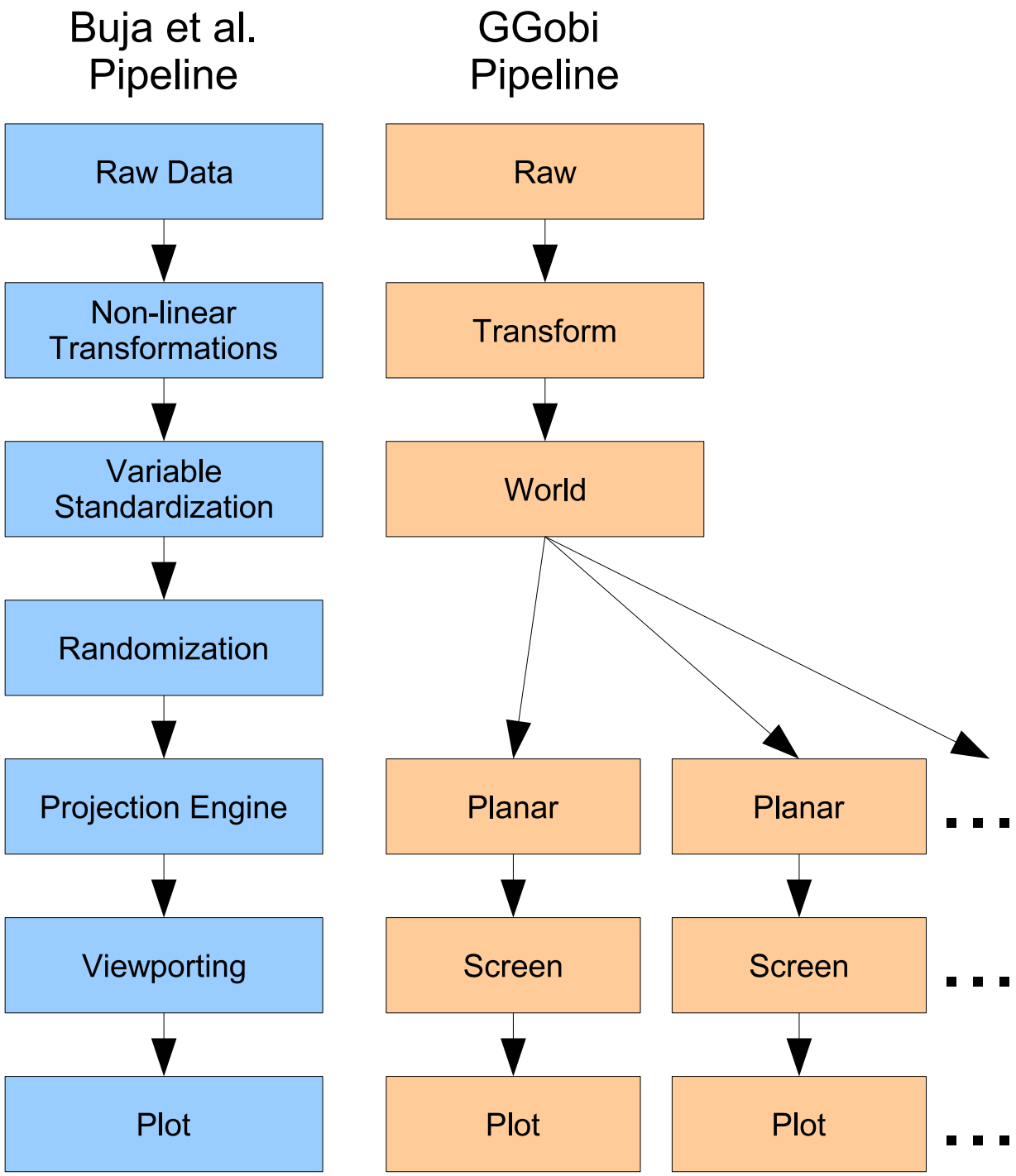
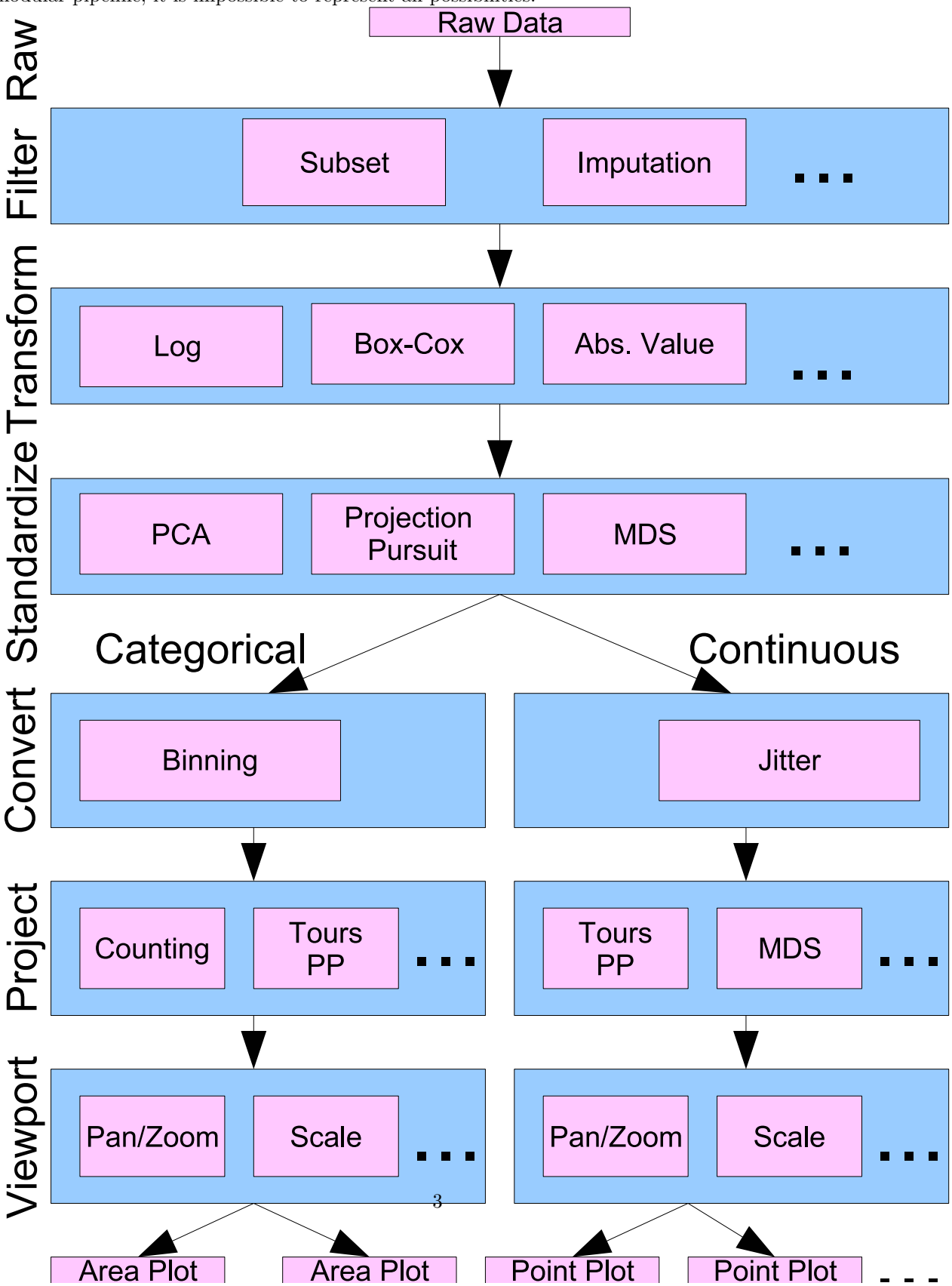


Figure 2: Diagram of the redesigned GGobi pipeline. This is an attempt to confer the potential of a modular pipeline; it is impossible to represent all possibilities.



that the treatment of modules may be generalized. A module informs any interested observers, such as the next stage, of changes to its state. Thus, it plays the role of the model in the model-view-controller (MVC) pattern. The pipeline may be regarded as a long chain of data models, each a proxy of the one before it, and views and controllers may be attached at any point.

The purpose of this work is to refactor the pipeline at the software level. There is no intent to modify or extend the abstract conceptual framework for generating statistical graphics from data. Rather, the aim is to simplify and make more extensible the software implementation of those ideas. Improving the flexibility of the pipeline at the software level will greatly benefit categorical data visualization in GGobi, as the current pipeline is designed only for point plots. A modular pipeline will allow plugins to redefine the behavior of the pipeline by incorporating custom stages and reordering existing stages. For example, a plugin could attach a graph layout algorithm to the pipeline, which would add variables specifying the coordinates of each point in the layout.

The modular pipeline is based on an abstract stage class, from which all pipeline modules extend. The stage objects are connected to form a tree structure. The only restriction on the order of the stages is that the root of the tree must be a dataset. Each descendant of the root has a single parent stage. When data is requested of a stage, the stage may, in turn, request the data from its parent. The parent value may then be modified in some way before it is returned to the caller. Alternatively, the stage could return a cached value. This behavior is dependent on the stage implementation. A change to the data at a particular stage is communicated to children of the stage via signals. The change propagates, as appropriate, until reaching a terminal leaf stage, which is usually a plot. Changes may result from direct modification of the data or, more commonly, from manipulation of the pipeline that affects its transformation of the data. For example, the user could apply a log transformation to a variable or add a new filter rule to the subset stage.

A stage holds a list of variable objects. Each variable object describes a column in the data matrix. It serves essentially the same purpose as the old variable structure, except it has been refactored for better encapsulation and synchronization with other objects. The description of a variable includes its name, type, and limits. Any attributes of the variable that depend on the data, such as the limits, are recalculated in response to changes to the corresponding data column. The type of a variable may serve as a flag for special handling. Two examples of built-in variable types are real and categorical. The variable type is dynamic, so it is possible, for example, to treat a real variable as categorical, on the fly.

User control of the pipeline may be graphical or based on a command line interface, such as the R console via rggobi. The graphical controls may be directly tied to pipeline parameters by binding widget properties to stage properties. This is convenient to implement with the GObject library, since it is possible to observe changes in any object property. For example, the Boolean property of whether the tour is activated could be bound to the checked property of a checkbox widget with only a few lines of code. This simplicity greatly facilitates the construction of task-specific GUI's on top of the pipeline. GGobi, while a general purpose tool, has a well-defined purpose that is reflected in its GUI. This approach differs from that of Orca, in which the pipeline constructs a GUI by placing controls corresponding to parameters into a provided container. The decision to deviate from Orca and leave the GUI construction to the client is based on the recognition that, in order to maximize usability, a GUI must be designed for the task at hand.

The diagram in Figure 2 organizes potential modules in the pipeline according to the abstract stages proposed by Buja et al., with two additional steps. The first new step, filtering, occurs just after raw and involves operations like removing unwanted observations and imputing missing values. The second additional category includes stages that convert between categorical and continuous variables. This organization is only conceptual; it is not enforced at the object level. The pipeline in the next version of GGobi will include all of the stages from the original pipeline. In addition, there will be stages for functionality that has existed separately from the pipeline proper, such as jittering, subsetting, and categorical data handling.

In contrast to the original pipeline, the new pipeline is branched for separate handling of categorical and continuous variables. A conversion stage at the beginning of each branch ensures that every variable

is of the appropriate type. The continuous branch is essentially the same as the original pipeline, but the categorical branch is largely new. The categorical variables, including binned continuous variables, are passed to a counting stage that produces tables of counts according to some ordered combination of variables. The count dataset could include the counts for each level of the hierarchy. For example, if variable A is crossed with B, there would be counts for each level of A and each combination of levels from A and B.

An important deviation from the original GGobi design is the treatment of visual attributes, such as color, as ordinary columns in the data matrix. The rationale is that the attributes and actual data values all work together to specify a visualization. There are three major advantages to treating everything as a column. First, stages may transform attributes using the same mechanism as the data values. For example, a pipeline stage could set the glyph size according to some continuous variable. In the original pipeline, visual attributes are not subject to processing. Second, it is trivial to treat an attribute as actual data, when desired. For example, one could convert color to a categorical variable and plot observations according to their color. Finally, when storing the data, attributes may be written along with the real data, without any special handling. The distinction between measured variables and artificial attributes is still important, however, so there is a flag on the variable structure that indicates whether a column is artificial.

In summary, the redesigned GGobi pipeline improves on the original by borrowing the general idea of modular stages from Orca. The modular pipeline increases the extensibility of GGobi. In particular, it facilitates adding better support for categorical data. It also better serves as a model for various views and controls, such as the GGobi GUI and embedding environments like rggobi. All of the original innovations of GGobi over Orca are retained, like the reverse pipeline. Finally, the new pipeline is able to treat visual attributes as if they were actual data. Altogether, these improvements significantly expand the domain of possible visualizations generated by the GGobi pipeline.

References

[1] A. Buja, D. Asimov, C. Hurley, and JA McDonald. *Dynamic Graphics for Statistics*, chapter Elements of a viewing pipeline for data analysis, pages 277–308. Wadsworth, Brooks, Cole, 1988.

[2] P. Sutherland, A. Rossini, T. Lumley, N. Lewin-Koh, J. Dickerson, Z. Cox, and D. Cook. Orca: A visualization toolkit for high-dimensional data. *Journal of Computational and Graphical Statistics*, 9(3):509–529, 2000.