Computational Statistics manuscript No.
(will be inserted by the editor)

iPlots eXtreme - Next-generation Interactive Graphics
Design and Implementation of Modern Interactive Graphics

Simon Urbanek

Received: date / Accepted: date

Abstract Interactive graphics provide a very important tool that facilitates the process of exploratory data and model analysis which is a crucial step in real-world applied statistics. Only a very limited set of software exists that provides truly interactive graphics for data analysis, partially because it is not easy to implement. Very often specialized software is created to offer graphics for a particular problem, but many fundamental plots are omitted since it is not considered new research. In this paper we discuss a general framework that allows to create interactive graphics software on a sound foundation that offers consistent user interface, fast prototyping of new plots and extensibility to support interactive models.

In addition, we also discuss one implementation of the general framework: iPlots eXtreme - next-generation interactive graphics for analysis of large data in R. It provides most fundamental plot types and allows new interactive plots to be created. The implementation raises interactive graphics performance to an entirely new level. We will discuss briefly several methods that allowed us to achieve this goal and illustrate the use of advanced programmability features in conjunction with R.

Keywords interactive graphics · software · data analysis

1 Introduction

Interactive graphics play an important role in applied statistics, because the first step in any analysis is to get acquainted with the data in order to be able to devise adequate subsequent process. Interactive graphics are not limited to exploratory data analysis but provide also a powerful tool during the

Simon Urbanek
AT&T Labs - Research, Florham Park, NJ 07932
Tel.: +1-973-360-7056
E-mail: urbanek@research.att.com

course of choosing and validating models. However, there are not many modern interactive graphics software packages available that could keep up with the increasing data sizes. Many of the most widely used packages do not offer fully interactive graphics. One of the reasons is that creating a consistent, data-oriented interactive graphics is very difficult and most of the existing software is specialized only on a very specific task.

Our goal was to design a general framework for interactive graphics that is flexible enough to allow design of any useful interactive graphics yet enforces the necessary consistency that is important from the user interface point of view – this aspect is often overlooked. The framework implementation should then allow fast prototyping of new ideas for interactive plots.

Although general frameworks for static graphics have been proposed[1], they cannot be easily extended to cover interactive graphics or help with the design thereof. The main reason is that interactivity has to be introduced at a very fundamental level in order to be consistent across all graphics. Other interactive graphics packages such as GGobi[2] can be extended programmatically at data and drawing level using plug-ins, but do not provide an interactivity framework for creating new plots. In this paper we describe a general framework for creating interactive graphics that focuses on the interactivity but does not impede with the flexibility of graphics. We also describe both design and implementation of modern, extensible interactive graphics on the basis of the iPlots project.

In the next section we will focus on what constitutes interactive graphics and the challenges ahead. In section 3 we will discuss the design in all three main areas - plots, interactions and linking. The subsequent section 4 introduces the main goals of **iPlots eXtreme** and highlights implementation details with respect to those goals. It is followed by a section 5 illustrating the use of iPlots eXtreme on examples in R[3]. Finally, we summarize the main points in section 6.

2 Interactive Graphics

Interactive graphics and statistical plots that allow direct interaction of the user with the plot. The most basic functionality that must be supported by interactive statistical graphics are **selection** (highlighting), **queries**, **zooming**, **direct change of parameters** and **multiple views**[4]. The basic concept underlying selection and highlighting is linking, the most basic requirement for all interactions and ensures that any changes of state (e.g., selection of points) are propagated to all components (the same points are highlighted in all plots). Queries allow us to enrich plots with information without the limitations and clutter associated with static labels. Zooming (especially with logical and censored zooming) allows the user to look at very complex plots without the loss of context. Change of parameters enables the user to interactively influence properties of the plot (such as bin width in histograms) that allow viewing the data in different light. Multiple views have a similar purpose

but the different view can be different yet related plots (such as boxplots and dotplots). Interactive graphics are a much larger superset of dynamic graphics (such as continuous projection plots or 3D scatterplots), the latter offer only very limited interactions (in the previous example zooming and change of projection).

One major challenge of interactive graphics is the necessity for all operations to be instantaneous. This made it more difficult in the past to create interactive statistical software. However, recent advances in computing technology and software make it possible to tackle even large data challenges as we will see in the iPlots eXtreme implementation section.

Another major challenge is to provide an intuitive and consistent user interface. This aspect is often overlooked and leads to poor adoption of specialized interactive graphics software. We want this consideration to be part of the interactive graphics framework such that best practices in user interfaces can be included in the general implementation such that this consideration does not need to be explicitly coded again and again when prototyping new ideas and plots.

Historically, interactive graphics software consisted of stand-alone programs of fixed functionality. It was not possible to extend existing plots or to create new ones. With the new, generalized framework we want to take into consideration the possibility to enhance existing plots by adding new graphical elements as well as the construction of new types of interactive graphics. This allows the implementations of this framework to be both consistent yet highly extendable.

3 Design

All interactive graphics consist of three fundamental parts: **plot**, **interaction** and **linking**.

The plot is the actual visual representation of the graphic very much like its static counterpart, but the major difference is that every single part of the plot must be aware of interactions and linking — it must be designed accordingly. Attempts to add interactions to existing static plots such as in R have failed for all but very trivial interactions exactly because of the lack of fundamental interactive building blocks (thus requiring re-implementation for each view) and the inability to trace for the graphical representation back to the data.

Interactions are actions of the user such as using mouse or keyboard that solicit responses from the plot. Some basic interactions are selection (such as dragging the mouse to create a selection rectangle), query initiation (moving a mouse over an item while holding a modifier key) or invocation of pop-up menus.

Linking ensures consistency by propagating changes from one object to all related objects. The most apparent case of linking is to propagate changes in the selection set to all plots. There are many other types such as, for example, the linking of category permutations in categorical variables (re-ordering

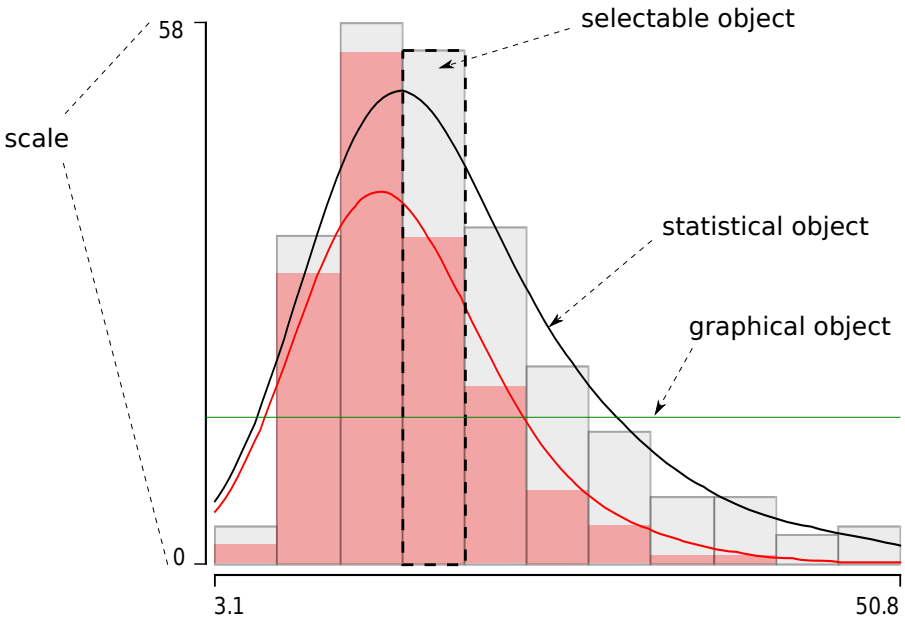

Fig. 1 Most basic objects in a plot

bars in a barchart changes the order in all plots with that variable including mosaic plots, spine plots, etc.) or linking between datasets (such as from gene expression data to pathways or chromosomes).

The most natural approach for a framework for interactive graphics is to be object-oriented. Each displayed element is an object with properties and can respond to interactions. The properties can be of graphical nature such as position on the screen or its color, but can be arbitrarily complex such as binning of the underlying data or a statistical model. Objects can react to direct user interaction such as mouse or keyboard use, but in most cases on a more general level to selection, linking change or events from other objects. In the following we will describe the most basic classes of objects in a plot as the visual representation of the graphic.

3.1 Plots

Each plot is a *c*ontainer that can hold arbitrary many other objects, also known as *c*omponents. Each interactive plot is built from basic component classes that make it easier to create new plot types. The most basic object types are **scales**, **graphical objects**, **statistical objects** and **selectable objects**. The composition of a plot is illustrated on an interactive histogram in Figure 1.

Scales map from the data space to the graphical coordinates - very much like in static graphics[1]. They allow to position objects on the surface of the

plot relative to each other such that the representation is sensible with respect to the displayed data. For example in a histogram of a variable V one scale maps from the data space of the displayed variable V directly to the x-axis of the graphical coordinates. In this case the scale is simply a linear function mapping elements of V into real values representing the horizontal position on the screen. Since the scale depends directly on the variable V, we can create a link between the two. The other scale in a histogram maps counts in the bins to the graphical y-axis position on the screen. This scale is indirect (virtual) since it does not depend directly on the variable V but only on the counts in the bins. Scales are not necessarily linear – for example discrete scales map ordinal categories to their positions on the screen which can be permuted and the categories can be spread equidistantly or according to their size.

Graphical objects are the most basic objects that are displayed on the screen. They do not have any associated high-level interactions and do not depend on the data directly. They can, however, have access to scales such that graphical object behave reasonably to changes in scale such as zooming. In our histogram example we have a simple horizontal green line which will appear at the right place even if we rescale the histogram, but it will not react to selection or changes in the data.

Statistical objects are objects that can be traced back to the original data or models. This allows them, among other things, to be updated when the underlying data or model changes. Density estimation polylines in the histogram are statistical objects as they are tied to the density estimation 'model'. Although statistical objects do not react to selection, they support queries.

Selection objects are special statistical objects that are linked back to the data and know how to display highlighting and react to selection. In the histogram example a bar is a selection object – it knows which cases in the data it represents (namely those corresponding to that particular bin) and thus whenever the highlighting changes[1] it knows how to fill the rectangle with highlighting color to denote the currently selected cases in that bin. In addition, statistical objects are involved when the user performs a selection (e.g., by dragging a mouse and creating a selection rectangle) – if the object is included in the selection then the cases it represents are selected automatically.

This means that the interactive selection is handled by the framework: implementation of selection tools, computation of the resulting highlighting and a combination of operations and queries. Each selectable object is only concerned with the graphical representation of the data. Examples of such objects are rectangles (highlighting is parallel to one of the edges, used in area-based plots such as barchars, spineplots, mosaic plots, fluctuation diagrams, ...), polylines (used in parallel coordinate plots), points (used in scatterplot, dotplots, ...) and polygons (used in maps).

Typically, each plot creates scales (including their representation as graphical objects for labeling: *axes*) and selection objects representing the data it

[1] We use highlighting for illustration purposes, but the concept can be generalized for arbitrary properties associated with the data such as color brushing or weighting – and our implementation uses this general concept.

displays. In the case of a histogram the data is binned and one "selectable rectangle" is created for each bin. All basic interactions such as selection and highlighting are then handled entirely by the framework. The parameters of a histogram are bin width and anchor point. When those change, the plot's responsibility is the re-compute the binning and adjust the statistical objects accordingly.

To illustrate this approach on other plots, parallel coordinate plots create either one (common) scale for the y axis or p independent scales where p is the number of variables (coordinates). Then the x scale is a discrete scale mapping the coordinate permutation to the corresponding graphical positions of the coordinates. The graphical objects are the lines representing the axes and statistical polylines each representing one case in the dataset with nodes computed by applying scales to the data values. Advanced features such as semi-transparency are implemented by simply adjusting the opacity property of the objects and are also common to all plots.

The availability of statistical and selection objects is a crucial distinction between an interactive framework such as iPlots and a post-hoc implementation of interactive graphics on top of a graphical system. In the latter case all objects are merely graphical objects and the interaction has to be defined explicitly for each plot, object and interaction. This fails to ensure consistency in the system. For example, a selection in one plot could behave differently from another plot due to the lack of global oversight. Delegation of that responsibility to the interactive framework guarantees consistency of the user interface, higher efficiency since the system can be tuned as a whole and much easier implementation of new plot types.

3.2 Interactions

The goal of the interactive framework is to maintain consistency for the most important and commonly used interactions such as selection, zoom and queries. The benefit is twofold – authors of new plots do not need to spend time on re-implementation of those interactions and at the same time the user will have the same user experience in all plots and thus find it easier to work with the system.

As described in the previous section a key to this consistency is the decision to let the framework handle the graphical representation of the selection and let the selection objects decide whether they are included in the current operation or not. In the case of the selection, the details of handling the mouse events while the user is dragging it to create a selection rectangle are left to the system. The statistical objects are then queried whether they intersect the final rectangle. The system itself adjusts the highlighting and notifies any objects that need to know as we will describe in the following section.

Similarly, a query of a statistical object does not have to rely on the plot to construct the result. Since the system knows which data points are represented

by a given statistical object it can use this information to create a default query or enhance an existing query.

In addition to interactions handled by the system, each graphical object as well as the plot itself can be designed to handle its own special interactions. This retains the flexibility of any object oriented system to be extended in a very flexible way.

3.3 Highlighting

The concept of highlighting allows to visually distinguish cases of interest from other cases in the data. Such subset is usually defined using a sequence of selection operations. It can be described as a state associated with each case. This 'highlighted' state is then used by all plots to visualize each case in accordance with its state. Other commonly used complementary states are *ghosting* (associated cases are hidden) and *brushing* (associated cases are represented as a group, usually using a different color). Highlighting is transient and has higher precedence than brushing. For example, an area plot will first represent all highlighted cases regardless of their group using highlighting color and subsequently only non-highlighted cases will be represented using their corresponding brushed color. The management of the states is performed by the *marker* object which updates the states according to selections and notifies interested parties (usually selectable objects) about any changes to facilitate linking.

In the same sense that the concept of 'objects' is fundamental to the framework, so is also the generalization of the Model-View-Controller[5] concept used implicitly throughout. In the case of the highlighting the state vector is the model, views are selectable objects representing the highlighting and controller is the marker object performing selection and notification. For plots the data is the model, a collection of statistical objects is the view and the managing plot class is the controller.

3.4 Linking

Linking is one of the most essential part of an interactive graphics system, because the benefit of interactive graphics comes from the ability to display conditional relationships very quickly and in a very intuitive manner. A subgroup under consideration is immediately highlighted in all other views. In the most basic interactive graphics linking is used to synchronize the highlighting between all plots. This can be considered as a general concept of tracking a property (in this case the highlighted state) by objects interested in the property (here plots).

The linking is not limited to the highlighting state but can be applied to any property in the system. Other examples are permutations of categories or coordinates, scale parameters (c.f. linked scales) and cross-dataset linking. In

our framework each interested object registers with the property it wants to track and receives a notification whenever this property changes. For example, selectable components register automatically with the marker such that they are informed whenever the highlighting changes. Discrete scales register with the permutation property of the categorical variable they represent such that they can re-arrange the position of object on the screen should the order of the categories change. The marker of one dataset can register with the marker of another dataset to keep their highlighting states in sync.

This design allows not only 1:1 linking but also 1:n or even m:n linking since the link function is defined by the notification response. It could be purely informative (such as for selectable objects) or change properties of the object that requests to be notified (as in the case of linking datasets).

One slight complication is the fact the notification could potentially change the property again, thus triggering a possibly infinite notification loop. A partial solution is to disallow the notification to change the property by which it was triggered. Nonetheless, an indirect change could still occur by changing another property which will trigger further notifications that may change the first property. Although this may sound like an artificially constructed case, it can occur frequently (albeit not infinitely) when linking between hierarchical datasets (such selection of a gene triggers a selection of a protein which in turn may select more genes). Although we detect such loops in our system by tracking the notification chain, links need to be carefully designed to prevent situations that may be counter-intuitive for the user.

4 iPlots eXtreme

The framework described above is fairly general and has been first implemented in a slightly limited form as the iPlots [6] package. Statistical objects were linked to data and only available as selection objects then called *plot primitives*. Subsequent evolution of the framework to support visualization of models was one aspect that has led us to think about a new implementation.

We have taken the opportunity to create an entirely new interactive graphics implementation **iPlots eXtreme** by focusing on several aspects: **performance**, **interactive models** and **intuitive use**.

Performance was the key design goal in order to allow analysis of large datasets which at the time of writing means millions of data points. The use of highly-optimized native code (a subset of C++) along with OpenGL allows for very fast graphics. The iPlots eXtreme framework can also be used in R as a package. In that case the use of native code allows us to share data directly with R such that no displayed data needs to be copied. This allows us to reduce the memory usage dramatically. In addition, we have created our own high-performance object system for iPlots eXtreme such that we do not have to rely on any external libraries, including STL to avoid issues associated with it. R objects fit seamlessly into that object system allowing tighter integration.

One further benefit of the high-performance architecture of iPlots eXtreme is that it can be leveraged by R for its own purposes as well. iPlots eXtreme includes an implementation of the R graphics device API such that R graphics can take advantage of the performance offered by this architecture. In addition, this allows arbitrary combination of iPlots eXtreme interactive plots and R graphics since both are components in the same framework and thus can be even mixed in the same window.

Interactive models are statistical models that can be visualized in plots and modified interactively. For example, it is possible to display predicted values of a polynomial spline regression model in one variable as a polyline in a scatterplot. Such visual representation can then be used interactively: the coefficients of the model can be queried or the degree of the polynomial can be modified interactively to re-fit the model. The idea is to not only represent the model visually but also to allow interactions with the model.

Intuitive use is important in interactive graphics, because the range of interactions is usually large and the user needs to keep in mind the context of the work. In iPlots there are two levels at which the user interacts with the software: interactivity in the plots and command line R interface. In iPlots eXtreme we want to make sure it is intuitive at both levels.

The general framework behind iPlots enforces user interface guidelines such that the same interactions are possible in all plots where such interaction is meaningful. It is desirable for the plot controls to be subtle enough to not distract from the data yet readily available to allow an efficient workflow. In the following section we highlight some of the ideas used to address the key goals.

4.1 Layers

Interactive graphics require very fast response, especially during interactive operations such as selection. The user expects a smooth resizing of the selection rectangle or movement of the brush – the response must be in the order of dozens of milliseconds. Generally graphics display systems (including R, Processing[7] or OpenGL) are based on the ink-on-paper concept which means that objects cannot be simply moved on the screen – only new 'paint' can be applied to the screen. In order to move an object, it often means that the whole plot needs to be redrawn. However, this is a very expensive operation especially when drawing glyph-based plots for many datapoints.

We address this problem by introducing graphical *layers* to our system[8]. For performance reasons the layers are sequential in that the layers can be peeled off from top to bottom. Restoring a given layer also restores all layers below. The plot can then be separated into logical units that correspond to layers. The minimal requirement is to have four layers in the following order: background, data objects, highlighting and interaction as illustrated in Figure 2. Additional custom layers are also allowed.

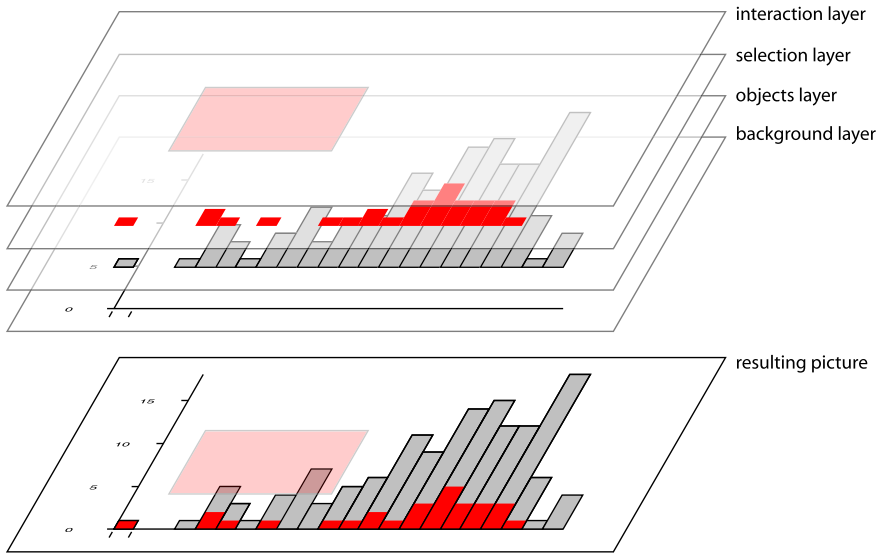

Fig. 2 Interactions layers

When the user drags a selection rectangle the highlighting layer is restored and the new rectangle is drawn on interaction layer. This means that no data points or highlighting needs to be drawn since they are already present in the underlying layers. The cost is therefore reduced to restoring the highlighting layer and drawing a new selection rectangle.

4.2 High-performance graphics

The major bottleneck of all interactive graphics systems is the actual drawing on the screen. Fortunately for us a multi-billion dollar industry is concerned with the exact same problem – the computer games industry. This has made high-performance graphics processing units (GPU) affordable and commodity in modern computers. It has also helped to create an international standard Open Graphics Library (OpenGL)[9] which is now supported practically by all commonly used computers and operating systems. This has direct impact on our ability to use interactive graphics for analysis of large data since the size of interactively displayable datasets has increased by the use of OpenGL in iPlots eXtreme by orders of magnitude as illustrated in Table 1.

In addition to providing general high-performance graphics, it also provides a flexible layout system that allows other components to be combined with plots. One example use of this general framework is the R device component which allows the placement of R graphics directly in iPlots eXtreme windows alongside with interactive plots. This enable R to leverage the speed of the graphics system provided by iPlots eXtreme as well as the direct combination

Table 1 Time spent to re-draw a plot in existing interactive graphics packages

	GGobi	iPlots	iPlots eXtreme
scatterplot 1,000,000 points	1,440 ms	990 ms	57 ms
parallel coordinates plot 100,000 cases	1,530 ms	1,710 ms	27 ms

(All tests were performed on Apple Mac Pro 2.66GHz, GeForce GT 120, R 2.10.0, Mac OS X 10.5.8 with GGobi 2.1.8, iPlots 1.1-3 and iPlots eXtreme 3.0. All plots were scaled the same net area as used by the actual plot corresponding to 800 by 600 pixels iPlots eXtreme display. Both iPlots and iPlots eXtreme were performing anti-aliasing (smoother result but slower rendering) and alpha blending (semi-transparency) – GGobi does not support either. The real time spent to re-draw a plot was measured.)

of R graphics and iPlots. Although it is not the main focus of iPlots, various 'widgets' can be implemented in this framework such as buttons, drop-down menus or checkboxes, allowing the construction of application-specific user interfaces.

5 Examples

The iPlots eXtreme system can be used as a stand-alone software, but its true flexibility is leveraged when integrated with R. The aim is to keep a flat learning curve for casual users on the one hand, yet provide versatility and extensibility for advanced users. Basic tasks such as creating a plot should be familiar to R users as it entails prepending the commonly used plot commands with an `i` for 'interactive'. For example a regular scatterplot would be created using `plot(x, y)` and interactive scatterplot is created with `iplot(x, y)`. Analogously `hist` becomes `ihist`, although there are shorter versions for certain plot types such as `ibar` for a barchart or `ipcp` for parallel coordinates plot.

The difference between R plot commands and iPlots is that all commands in iPlots return an object that represents the plot and can be used to interact with it or change its parameters. This allows the co-existence of multiple plots at the same time and the ability to address a particular plot directly (as opposed to the concept of a 'current' device in R).

Analogously to the naming convention of plots commands, we use the same principle for commands creating new objects resulting in `ilines`, `iabline`, `ipolygon` and others. Unlike their non-interactive counterparts those commands return objects that can be modified as well, such as moving a line or changing the shape or color of a polygon.

In iPlots eXtreme we have shifted entirely to the object oriented paradigm such that every plot or graphics element is an object that can be operated on. This allows very terse yet intuitive syntax such as

```
iplot(x,y) + lm(y~x)
```

to create a scatterplot plot and add a linear model representation to it. Additional objects such as lines or polygons can be added or removed in a similar manner.

In addition, we have introduced the concept of *virtual attributes* which allow direct access to plot parameters using the `$` extraction operator. Each plot provides a set of properties that can be accessed using names, for example:

```
> h <- ihist(x)  # create an interactive histogram
> h$bin.width   # retrieve the current bin width
[1] 0.3272727
> h$bin.width <- 0.4  # set the bin width programmatically
```

All such properties are "live" which means that interactive changes are reflected immediately in the values of the property. Assignment of values to those properties triggers an immediate change in the plot such as change of the bin width in the above example. Other examples include change of scales (`xlim`, `ylim`) or even properties of graphical objects such as color:

```
> iplot(x, y)
Scatterplot x vs y
> l <- ilines(lowess(x, y))
> l$color
[1] "#000000"
> l$color = "red"
```

Virtual attributes also provide a user-friendly facility to offer callback functions. Let's say we want to add a graphical object to the plot that also depends on the state of the plot, i.e., we want a function to be called whenever the plot changes. This is easily done by using the `onChange` virtual attribute of an object as the following example of a density estimator illustrates:

```
h <- ihist(x)                    # create a histogram
dp <- iPolygon()                 # create an empty polygon object
# on change compute density and adjust the polygon
dp$onChange <- function(pp) {
    h <- pp$plot                 # get the plot object
    print(h$bin.width)           # print current bin width
    d <- density(x, h$bin.width) # compute new density
    pp$x <- d$x                  # update the points accordingly
    pp$y <- d$y * length(x) * h$bin.width
}
add(h, dp)                      # add the polygon to the plot
```

In a similar manner callback function can be specified for different events such as highlighting (marker) change or user interactions (keyboard and mouse events).

Since plot objects reflect all changes immediately, they behave as mutable objects – similar to environments. This enables general-purpose storage for

custom use. To avoid collisions with virtual attributes custom attribute names should by convention start with a capital letter or a dot.

We have also taken the opportunity to re-design the interface with the windowing system. iPlots eXtreme are based on a very flexible layout system that allows multiple components (such as interactive plots, R graphics, user interface controls) in the same window. All plot commands allow the specification of the enclosing container as well as dimensions and resizing behavior. If not specified, a new window is created to be the container of the plot object and the plot component will fill out the entire window.

The layout system is based on the bars-and-springs system used in iWidgets[10]. Cross-platform compatibility is guaranteed by the reliance on the established OpenGL standard and a thin layer that maps system-dependent interactions with the windowing system to iPlots eXtreme object. Therefore only one fairly small class needs to be implemented to support any new platform. iPlots eXtreme currently support Windows, Mac OS X (both with native toolkits) and unix (using GLUT[11]).

6 Summary

We have described a general framework for implementing interactive graphics which started as the basis of iPlots, icp and has been refined for iPlots eXtreme. The object-oriented framework extends the concepts known from static plots such as scales with objects allowing construction of consistent interactive graphics. Selectable objects fully support selection, highlighting and brushing. Statistical objects are directly liked to the data or models and define the behavior with respect to user interactions. Graphical objects are programmable, but not directly aware of data-affecting interactions such as selection or highlighting. A basic set of selectable objects (rectangles, points, lines, polygons) covers most of the commonly used interactive plots and the framework allows for additional, custom objects. The use of such building blocks guarantees consistency in the user interface and makes it very easy to build new types of interactive graphics.

This general framework for interactive graphics has been implemented in iPlots eXtreme – an infrastructure for high-performance interactive graphics for the analysis of large data that lifts the scalability of interactive graphics to a new level. iPlots eXtreme use OpenGL for hardware-accelerated graphics display, optimized object system for performance and direct access to R object for tight integration with R where desired. Nonetheless the iPlots eXtreme system can be also used independently of R for high-performance visualization.

In addition to the focus on performance iPlots eXtreme offer several new concepts over iPlots: direct access to plot parameters and other objects using the `$` extraction operator, callback functions, simplified syntax by overloading addition `+` and subtraction operators `-`, support for formula plot specifications, flexible layout system and the ability to act as an R graphics device. iPlots eXtreme also introduce the ability to represent statistical models interactively.

The most recent documentation and implementation is freely available from http://www.iplots.org/.

References

1. Wilkinson Leland, The Grammar of Graphics, Springer (1999)
2. Swayne Deborah F, Temple Lang Duncan, Buja Andreas, Cook Dianne, GGobi: Evolving from XGobi into an Extensible Framework for Interactive Data Visualization, *Computational Statistics and Data Analysis*, 43, p.423-444 (2003)
3. R Development Core Team, R: A Language and Environment for Statistical Computing, http://www.R-project.org, R Foundation for Statistical Computing, Vienna, Austria (2009)
4. Unwin Antony, Requirements for Interactive Graphics Software for Exploratory Data Analysis, *Computational Statistics*, 14, p.7–22 (1999)
5. Reenskaug, Trygve : Models - Views - Controllers. Technical note, Xerox PARC, December 1979.
   http://heim.ifi.uio.no/~trygver/mvc/index.html
6. Urbanek Simon Theus Martin, iPlots - High Interaction Graphics for R, *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)* (2003)
7. Processing language and environment, http://www.processing.org/
8. Wilhelm Adalbert FX, Interactive Statistical Graphics: The Paradigm of Linked Views. Habilitationsschrift, University of Augsburg (1999)
9. Shreiner, Dave and OpenGL Architecture Review Board, OpenGL reference manual: the official reference document to OpenGL, version 1.4, Addison-Wesley, Boston (2004)
10. iWidgets - basic user interface widgets for R, R-package, http://rforge.net/iWidgets/
11. GLUT - The OpenGL Utility Toolkit,
    http://www.opengl.org/resources/libraries/glut/