Extending ggplot2 for linked and animated web graphics

Carson Sievert
Department of Statistics, Iowa State University
Susan VanderPlas
Department of Statistics, Iowa State University
Jun Cai
Department of Earth System Science, Tsinghua University
Kevin Ferris
Baseball Operations Department, Tampa Bay Rays
Faizan Uddin Fahad Khan
Department of Computer Science & Engineering, IIT BHU
Toby Dylan Hocking
Department of Human Genetics, McGill University

March 12, 2018

Abstract

Interactive web graphics are great for communication and knowledge sharing, but are difficult to leverage during the exploratory phase of a data science workflow. Even before the web, interactive graphics helped data analysts quickly gather insight from data, discover the unexpected, and develop better model diagnostics. Although web technologies make interactive graphics more accessible, they are not designed to fit inside an exploratory data analysis (EDA) workflow where rapid iteration between data manipulation, modeling, and visualization must occur. To better facilitate 'production-ready' exploratory web graphics, we need better interfaces between statistical computing environments (e.g. the `R` language) and client-side web technologies. We propose the `R` package **animint** for rapid creation of linked and animated web graphics through a simple extension of **ggplot2**'s implementation of the Grammar of Graphics. The extension allows one to write **ggplot2** code and produce a standalone web page with multiple linked views.

Keywords: Animation, Multiple linked views, Statistical graphics, Exploratory data analysis, Web technologies, Grammar of graphics

1 Introduction

For more than a half century now, statisticians have designed, built, and used interactive graphics for exploring high-dimensional data and better informing their modeling process. In fact, the ASA maintains a video library (http://stat-graphics.org/movies/) to document and demonstrate applications of instrumental interactive statistical graphics systems such as `PRIM-9` (Fisherkeller and Tukey 1988), `Data Viewer` (Andreas Buja and McDonald 1988), `XGobi` (Swayne, Cook, and Buja 1998), `GGobi` (D. Cook and Swayne 2007), and `Mondrian` (Theus 2002). These, as well as other influential systems, such as `LISP-STAT` (Tierney 1990) and `MANET` (Unwin A. 1996), all have a rich support for accomplishing a wide variety of statistical analysis tasks, and most were developed before the web browser had rich graphics support.

All of these systems, as well as some more modern systems, such as **rggobi** (Duncan Temple Lang 2016), **iplots** (Urbanek 2011), **cranvas** (Xie et al. 2013), **loon** (Waddell and Oldford 2018), etc, require a heavy set of computational dependencies in order to view or interact with graphics. These requirements grant the freedom to leverage libraries with sophisticated statistical functionality on-demand, but it limits the ability to share or embed such graphics in a larger document. Some of these systems allow users to create the graphics from the command-line, which as Unwin and Hofmann (2009) points out, allows power users to combine the strengths of a programming interface (e.g., precise, repeatable, fast, and extensible) with the strengths of a graphical interface (e.g., intuitive, forgiving, and easy-to-use). Web technologies can certainly be used to build a similar class of system, but in order to capitalize on the key strength of web technologies (e.g., accessible, portable, and composable), we must be mindful of which technologies we are requiring in such a system, and minimize those requirements whenever possible.

Generally speaking, web graphics that use purely client-side technologies (i.e., `HTML`, `SVG`, `CSS`, and `JavaScript`) are desired over client-server web applications because of their relative ease of distribution and maintenance. This is why many web-based graphing libraries work entirely with client-side technologies, like `Vega` (Trifacta 2014) and `plotly.js` (Plotly 2015). Unfortunately, client-side technologies are not particularly well-suited for statistical computation, which we often want to leverage via dynamic controls in an interactive statistical graphics system. In this scenario, it often makes sense to introduce a client-server infrastructure to leverage functionality that is not natively supported by web browsers (e.g., `R`, `python`, etc).

Focusing solely on the `R` language, there are now numerous ways to develop web applications, including the `R` package **shiny** (RStudio 2013), which makes it easy for `R` users to take their existing scripting workflow and wrap a web interface around it. **Shiny** is great for quickly prototyping interactive webpages that re-execute R code on-demand, but that flexibility comes at the cost of requiring a complex web server framework, which can be hard to scale, maintain, and secure sensitive information. Unfortunately, all too often, a web application framework is used to implement linked and animated graphics that could more easily be described with an idiomatic `R` interface which produces a purely client-side result.

There are now many `R` packages that interface with purely client-side graphing libraries and give users the option of embedding these graphics in a larger web application. This movement has made interactive graphics a lot more accessible to newcomers and also allows power-users to combine the strengths of web technologies and statistical computing. In fact, this is a large enough use case that the `R` package **htmlwidgets** (Vaidyanathan et al. 2018) was created to make it easier to get these interfaces to work seamlessly in any context (e.g., **shiny**, **rmarkdown**, RStudio, terminal, Jupyter notebook, etc). In most cases, these `R` packages do not have great support for linking views, meaning a callback to server-side `R` (e.g., **shiny**) is typically required for such a task. Very recently, some **htmlwidgets** packages have gained **crosstalk** (Cheng 2016) support for linking views with purely client-side technologies, but the type of linking is purposefully restricted (e.g., 1-to-1 transient linking) since it's very difficult to standardize an API for linking arbitrary libraries. We think this is a great direction for 'production-ready' exploratory graphics, but also hope to see more opinionated approaches to this idea, like **plotly** (Plotly 2015), that focus more on statistical aggregations, missing values, and selection sequences (Hofmann and Theus 1998).

We propose an extension of **ggplot2**'s layered Grammar of Graphics API to create interactive web graphics that do not require a callback to server-side `R`. The core idea lies in attaching metadata to graphical marks that can be used to hide/show subsets of data. The resulting framework is quite similar to what D. Cook, Buja, and Swayne (2007) describe as brushing

Table 1: New features that animint adds to the grammar of graphics.

Feature	Type	Description
`clickSelects`	aesthetic	value(s) to select on click
`showSelected`	aesthetic	value(s) attached to mark(s) that determine when they are shown
`key`	aesthetic	value(s) attached to mark(s) for smooth transitions.
`tooltip`	aesthetic	information to display on hover
`href`	aesthetic	URL link to open on click
`first`	option	what value(s) should be selected by default?
`time`	option	delay between animation frames.
`duration`	option	to specify smooth transitions.
`selector.types`	option	should selections accumulate?
`selectize`	option	include a dropdown widget to set selection value(s) indirectly?

in multiple linked views as a database query. The assignment of data to graphical marks is done through *aesthetic mappings*, which is a term the Grammar of Graphics Wilkinson et al. (2006) uses for mapping data to visual attributes (e.g., color, shape, x, y, etc). Typically aesthetic mappings are visual, meaning they can be easily seen in a static graphic, but our proposed aesthetic mappings control interactive properties, so they are not necessarily easily seen, but visual cues may be added to guide the user interaction. To give a small example, Figure 1 depicts a graphical query made by assigning metadata to graphical marks via the `clickSelects` and `showSelected` aesthetics. These aesthetics are essentially used to create a primary key between two tables of data, and as the name `clickSelects` suggests, queries are made by clicking directly on graphical marks, but other aesthetics could be used to support other direct manipulation events (e.g., `hoverSelects`, `brushSelects`, etc).

In addition to graphical queries, our extension supports a number of other interactive features, including animation, tooltips, and hyperlinks. A summary of these extensions and relevant additional options are provided in Table 1. There are a number of other options that can be used to control things specific to our implementation in the `R` package **animint** which are included with the supplemental materials.

2 Related work

In the last section, we motivated the need for `R` packages that create linked interactive graphics using client-side web technologies. We also proposed an extension to **ggplot2**'s API that supports a class of graphical queries. To help further explain where our work makes contributions to the field, this section further explores related work.

It is important to acknowledge that **ggplot2** is built on top of the `R` package **grid**, a low-level graphics system, which is now bundled with `R` itself (R Core Team 2017). Neither **grid**, nor **base** `R` graphics, have strong support for handling user interaction, which creates a need for add-on packages. There are a number of approaches these packages take to rendering, each with their own benefits and drawbacks. Traditionally, they build on low-level `R` interfaces to graphical systems such as GTK+ (Lawrence and Temple Lang 2010), Qt (Lawrence and Sarkar 2016), or Java GUI frameworks (Urbanek 2016). In general, the resulting system can be very fast and flexible, but sharing and reproducing output is usually a problem due to the heavy software requirements. Although there may be some sacrifices in performance, using the modern web browser as a rendering platform is more portable, accessible, and composable (i.e., graphics can be embedded within larger frameworks/documents).

Base `R` does provide a Scalable Vector Graphics (`SVG`) device, `svg()`, via the Cairo graphics API (Cairo 2016). The `R` package **SVGAnnotation** provides functionality to post-process `svg()` output in order to add interactive and dynamic features (Nolan and Lang 2012). This is a powerful approach, since in theory it can work with any `R` graphic, but the package is self-described as a proof-of-concept which reverse-engineers poorly-structured `svg()` output. As a result, it is not straightforward to extend this system for linked data visualizations with advanced functionality (multiple layers, multiple plots, multiple selection variables).

The lack of well-structured `SVG` for `R` graphics motivated the **gridSVG** package which provides sensible structuring of `SVG` output for **grid** graphics (Murrell and Potter 2015). This package also provides some low-level tools for animating or adding interactive features, where grid objects must be referenced by name. As a result, use of this interface to add interactivity to a **ggplot2** plot requires understanding of the grid naming scheme **ggplot2** uses internally. An interface where interactivity can be expressed by referencing the data to be visualized, rather than the building blocks of the graphics system, would be preferable since the former interface is decoupled from the implementation and does not require knowledge of **grid**.

In terms of the animation API, the `R` package **gganimate** is very similar to our system (Robinson 2016). It directly extends **ggplot2** by adding a new aesthetic, named `frame`, which splits the data into subsets (one for each unique value of the frame variable), produces a static plot for each subset, and uses the **animation** package to combine the images into a key frame animation (Xie 2013). This is quite similar but not as flexible as our system's support for animation, which we fully describe in Section 3.5. Either system has the ability to control the amount of time that a given frame is displayed, but our system can also animate the transition between frames via the `d3.transition()` API (Bostock, Oglevetsky, and Heer 2011). Smooth transitions help the animation viewer track positions between frames, which is useful in many scenarios, such as the World Bank example in Section 3.2. The **tweenr** package is similar in scope to `d3.transition()`, but operates on data frames instead of `SVG` elements (Pedersen 2016). One could actually use **tweenr** to implement smooth transitions in **animint**, but it would require pre-computing, storing, and loading an unnecessarily large amount of data.

Smooth transitions are also useful for touring data – a suite of statistical techniques for visualization of high-dimensional data. The supplementary materials show how to implement a tour in a standalone web page via **animint** and **tourr** (Wickham et al. 2011), but it's worth noting that projections (i.e., animation frames) must be pre-computed, so the functionality is limited compared to other solutions. The open-source software `GGobi` is currently the most fully-featured toolkit for touring data and has support for interactive techniques such as linking, zooming, panning, and identifying (D. Cook and Swayne 2007). The `R` package **rggobi** provides an `R` interface to `GGobi`'s graphical interface, but it unfortunately has many software requirements. Furthermore, sharing the interactive versions of these graphics are not possible. The `R` package **cranvas** aims to be the successor to `GGobi`, with support for similar interactive techniques, but with a more flexible interface for describing plots inspired by the Grammar of Graphics. **Cranvas** also has many software requirements which limits its portability and accessibility.

The `R` package **ggvis** is another interactive web graphics interface inspired by the Grammar of Graphics (Chang and Wickham 2015). It does not directly extend **ggplot2**, but instead provides a brand new purely functional interface which is designed with interactive graphics in mind. It currently relies on `Vega` to render the `SVG` graphics from `JSON`, and the `R` package **shiny** to enable many of its interactive capabilities (RStudio 2013). The interface gives tremendous power to `R` users, as it allows one to write `R` functions to handle user events. This power often comes with a cost, though, as **ggvis** uses callbacks to `R` via **shiny** to accomplish interactivity such as linked brushing. As we outline in our supplemental materials, our system does not require server-side `R`, but it can also be used inside **shiny** web applications.

Another `R` package for interactive graphics is **iplots** (Urbanek 2011), which has several important differences compared to **animint**. Brushing of linked iplots is supported for single-layer plots such as scatterplots or barplots, but it is not easy to define new multi-layer interactive plots. Furthermore since **iplots** does not use the Grammar of Graphics, it is difficult to create legends and multi-panel plots. Finally since **iplots** requires compiled C++ code for rendering on the local machine, its graphics are not as easy to share as **animint** graphics which can be viewed in a web browser.

3 Extending the layered grammar of graphics

In this section, we describe in detail our extension of **ggplot2**'s layered grammar of graphics implementation (Wickham 2010). In **ggplot2**, there are five essential components that define a layer of graphical makings: data, mappings (i.e., aesthetics), geometry, statistic, and position. These simple components are easy to understand in isolation and can be combined in many ways to express a wide array of graphics. For a simple example, here is one way to create a scatterplot in **ggplot2** of variables named `<X>` and `<Y>` in `<DATA>`:

```
ggplot() + layer(
  data = <DATA>,
  mapping = aes(x = <X>, y = <Y>),
  geom = "point",
  stat = "identity",
  position = "identity"
```

```
)
```

For every geometry, **ggplot2** provides a convenient wrapper around `layer()` which provides sensible defaults for the statistic and position (in this case, both are "identity"):

```
ggplot() + geom_point(
  data = <DATA>,
  aes(x = <X>, y = <Y>)
)
```

A single **ggplot2** plot can be comprised of multiple layers, and different layers can correspond to different data. Since each graphical mark within a **ggplot2** layer corresponds to one (or more) observations in `<DATA>`, aesthetic mappings provide a mechanism for mapping graphical selections to the original data (and vice-versa) which is essential to any interactive graphics system (Wickham et al. 2010). Thus, given a way to combine multiple **ggplot2** plots into a single view, this design can be extended to support a notion of multiple linked views, as those discussed by Ahlberg, Williamson, and Shneiderman (1991) and Buja et al. (1991).

3.1 Linking views via aesthetic mappings

D. Cook and Swayne (2007) use `SQL` queries to formalize the linked views infrastructure discussed in Ahlberg, Williamson, and Shneiderman (1991) and Buja et al. (1991). We use a similar approach to show how aesthetic mappings can be used to assign data values to graphical marks via **ggplot2** to support similar graphical queries. It's worth noting that, since these aesthetics effectively define a set of database queries that are known at print time, these queries can be made by direct manipulation of graphical marks and/or indirect manipulation via a dropdown widget, as discussed in Section 3.4. It's also worth noting that these aesthetics could be defined in such a way that they are not solely restricted to any particular direct manipulation event (e.g. mouse click), but for sake of demonstration, we restrict focus to our **animint** implementation, which has `clickSelects` and `showSelected` aesthetics.

Consider the R code below which uses these aesthetics to describe the interactive graphic shown in Figure~1.[1].

```
data(tips, package = "reshape2")
bar <- ggplot(tips) +
  geom_bar(aes(x = sex, clickSelects = sex))
scatter <- ggplot(tips) +
  geom_point(aes(x = total_bill, y = tip), alpha = 0.3) +
  geom_point(aes(x = total_bill, y = tip, showSelected = sex))
```

The `clickSelects` aesthetic is used to declare a geom which can be clicked to set selection value(s). For the example in Figure 1, we declare a bar with `clickSelects=sex` and refer to the current set of selection value(s) as `selected_sex`. The value of `selected_sex` can take on any subset of the unique values of `sex`, and in this case, the unique values are `Male` and `Female`. The selection set, `selected_sex`, is then used to select subsets of data for geoms with the `showSelected=sex` aesthetic. In this case there is a geom point with `showSelected=sex`, so clicking on a bar results in an update to the displayed subset of points.

In the R code above, the `geom_bar()` layer in the left-hand plot is linked to the 2nd `geom_point()` layer in the right-hand plot since the `clickSelects` and `showSelected` aesthetics are mapped to a common variable, `sex`. Note how the first `geom_point()` layer does not have a `showSelected` mapping, but has a bit of alpha transparency, so all the data is shown in light gray, and the current selection is portrayed in black. In other words, when a bar is clicked, in order to update the second layer of points, our system performs an SQL query of the form:

```
SELECT * FROM tips
  WHERE sex IN selected_sex
```

In this example, `selected_sex` is either `Male` or `Female` (a single selected value), but as we show in later examples, a selection set can also be multiple values. Although the `clickSelects` aesthetic is tied to a mouse click event, other aesthetics could easily be created

[1]Interactive versions of all of the figures mentioned in this paper can be found at http://members.cbio.mines-paristech.fr/~thocking/animint-paper-figures/

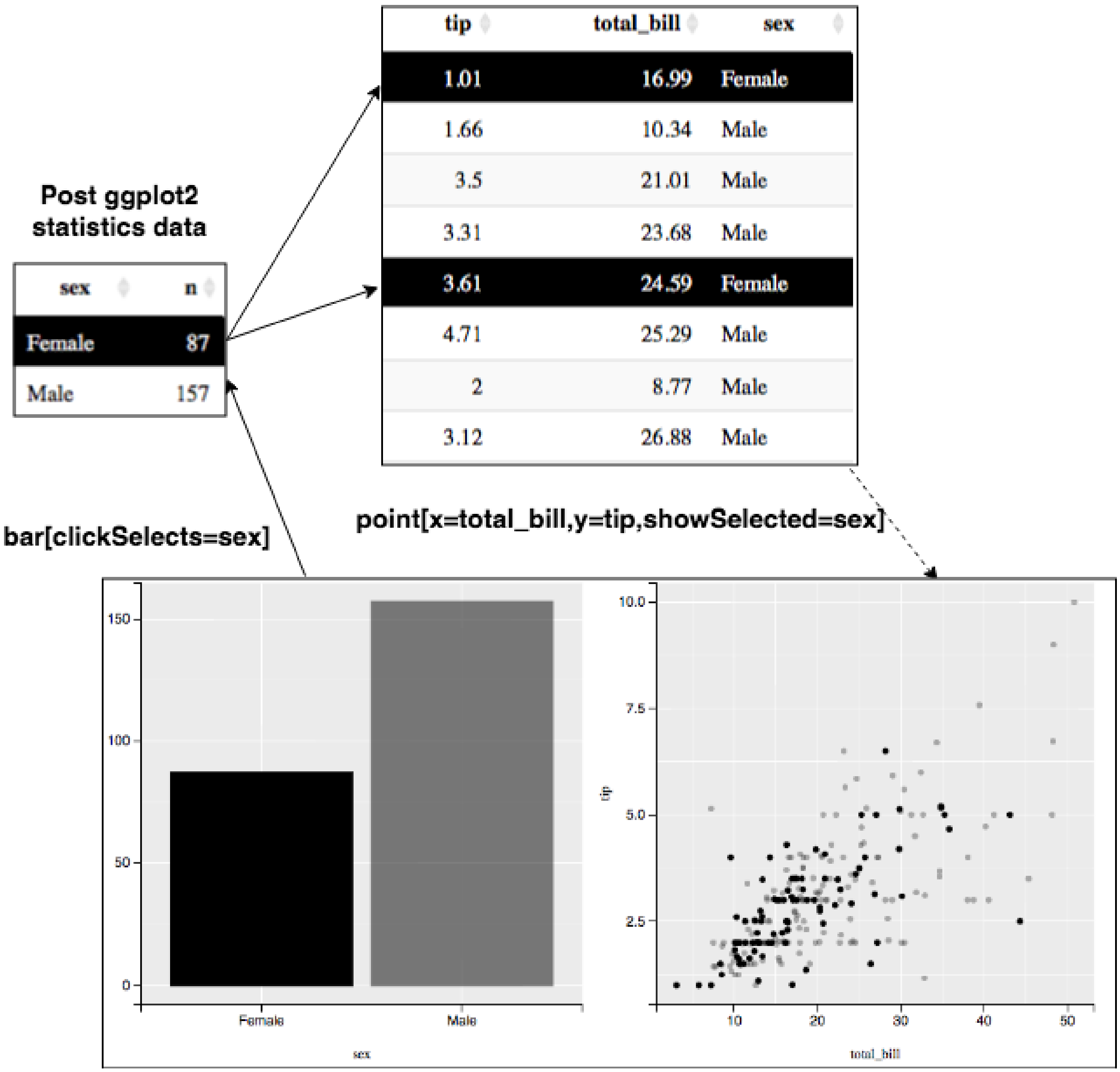

Figure 1: A graphical query of tips data set. Left: the `clickSelects` aesthetic designates a clickable geom bar that can change a selection variable. Right: the `showSelected` aesthetic designates a geom point that responds by showing only the data which corresponds to the current selection.

to support other selection events, such as hover or click+drag. Statistically speaking, this type of interaction is useful for navigating through joint distributions conditional upon discrete values. In this sense, our extension is closely related to trellis displays (Becker, Cleveland, and Shyu 2010) and linked scatterplot brushing (Becker and Cleveland 1987). The major differences are that our conditioning is layer-specific (not plot-specific), is not tied to a particular geometry, and can be controlled through direct manipulation or animation controls.

3.2 World Bank example

This section uses the linking framework introduced in the previous section to visualize a more complex data set provided by the World Bank. The interactive version of Figure 2 fosters exploration of the relationship between life expectancy and fertility rate over time for 205 countries. The year 1979 and the countries United States and Vietnam are selected in the static version of Figure 2, but readers are encouraged to change the selection by clicking on the interactive version, which is provided in the supplementary materials. The interactive version also makes use of additional animation options (explained later in Section 3.5), allowing us to visualize the evolution of the relationship between life expectancy and fertility rate.

We anticipate that some **ggplot2** users will be able to reverse engineer the code which creates Figure 2, simply by looking at it. In fact, this is a big reason why **ggplot2** is so widely used: it helps minimize the amount of time required to translate an idea for a figure into computer code. Note that, in the left-hand plot of Figure 2, we have a time series of the life expectancy where each line is a country (i.e., we `group` by country) and lines are colored by region. By clicking on a line, we also want the country label to appear in the right-hand plot, so we also need to set `clickSelects=country`. Lastly, by setting `showSelected=region` and `color=region`, we can hide/show lines by clicking on the color legend entries.

```
timeSeries <- ggplot() + geom_line(
  data = WorldBank,
  aes(x = year, y = life.expectancy,
      group = country, color = region,
```

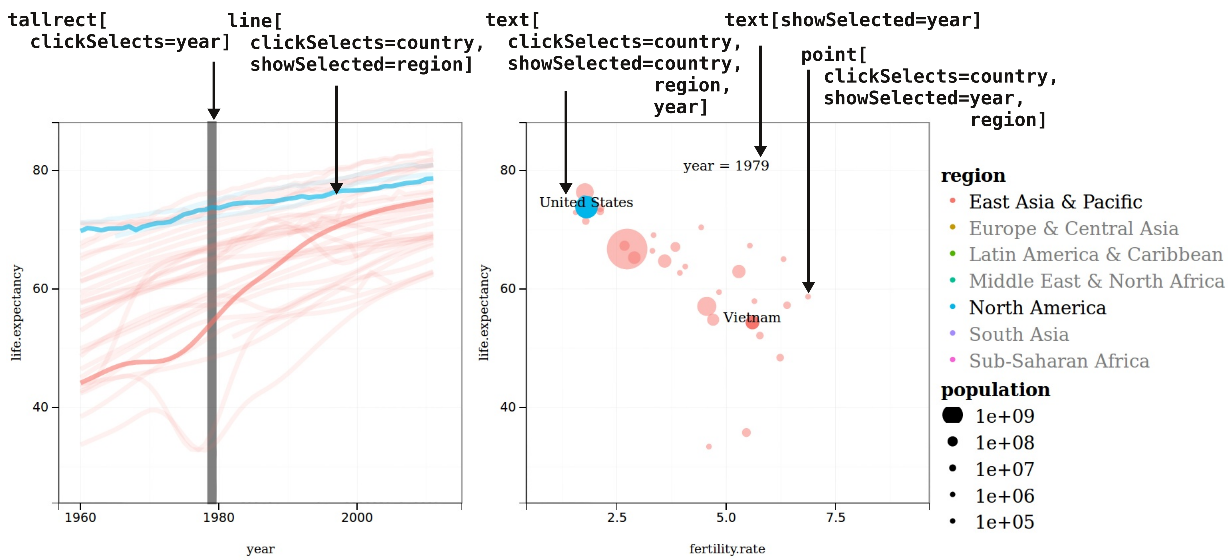

Figure 2: An interactive animation of World Bank demographic data of several countries, designed using `clickSelects` and `showSelected` aesthetics (top). Left: a multiple time series from 1960 to 2010 of life expectancy, with bold lines showing the selected countries and a vertical grey tallrect showing the selected year. Right: a scatterplot of life expectancy versus fertility rate of all countries. The legend and text elements show the current selection: year=1979, country={United States, Vietnam}, and region={East Asia & Pacific, North America}

```
      clickSelects = country,
      showSelected = region)
)
```

To help point out the currently selected year, we also provide a visual cue in the form of tall rectangles to the time series plot. These tall rectangles will also serve as a way to directly modify the selected year. The tallrect geometry is a special case of a rectangle that automatically spans the entire vertical range, so we just have to specify the horizontal range via `xmin` and `xmax` aesthetics. Also, since the layered grammar of graphics allows for different data in each layer, we supply a data frame with just the unique years in the entire data for this layer.

```
years <- data.frame(year = unique(WorldBank$year))
timeSeries <- timeSeries + geom_tallrect(
  data = years,
  aes(xmin = year - 0.5, xmax = year + 0.5,
      clickSelects = year)
)
```

As for the right-hand plot in Figure 2, there are three layers: a point layer for countries, a text layer for countries, and a text layer to display the selected year. By clicking on a point, we want to display the country text label and highlight the corresponding time series on the left-hand plot, so we set `clickSelects=country` in this layer. Furthermore, we only want to show the points for the selected year and region, so we also need `showSelected=year` and `showSelected2=region`.

```
scatterPlot <- ggplot() + geom_point(
  data = WorldBank,
  aes(x = fertility.rate, y = life.expectancy,
      color = region, size = population,
      clickSelects = country,
      showSelected = year,
      showSelected2 = region)
```

```
)
```

Note that any aesthetics containing the substring `showSelected` (including `showSelected2`) are interpreted as `showSelected` variables, and combined together using the intersection operation. In the example above, that means that a point will be drawn for the currently selected combination of year and region, as in the following SQL query,

```
SELECT * FROM WorldBank
  WHERE year   IN selected_year
  AND   region IN selected_region
```

Below, the text layer for annotating selected countries is essentially the same as the point layer, except we assign the country name to the `label` aesthetic.

```
scatterPlot <- scatterPlot + geom_text(
  data = WorldBank,
  aes(x = fertility.rate, y = life.expectancy,
      label = country,
      showSelected = country,
      showSelected2 = year,
      showSelected3 = region)
)
```

Lastly, to help identify the selected year when viewing the scatterplot, we add another layer of text at a fixed location.

```
scatterPlot <- scatterPlot + geom_text(
  data = years, x = 5, y = 80,
  aes(label = paste("year =", year),
      showSelected = year)
)
```

In summary, this section shows an example of how the proposed `clickSelects` and `showSelected` aesthetics can be used with several different geoms (line, point, text, tallrect), each of which can potentially display a different data set. In each case we use `clickSelects` to declare a geom that when clicked updates the current selection, and we use showSelected to declare a geom which responds to such changes by updating the set of displayed data. In the next sections, we further options that allow us to accumulate selections, update selections indirectly, and automate selection updates.

3.3 Linking and multiple selection

Linking is declared in R code by putting ggplots with common clickSelects and showSelected aesthetics together in a list. For example, we can link the ggplots from the previous section by including them together in the following list:

```
viz <- list(
  timeSeries = timeSeries,
  scatterPlot = scatterPlot
)
```

Linking is accomplished because the two ggplots declared clickSelects and showSelected aesthetics that refer to common variable names (region, year, country). For each such selection variable, our system updates the set of selected values in response to mouse clicks on clickSelects geoms, and then updates the corresponding data which is displayed for showSelected geoms.

Note that the viz list above can also contain numerous options which are listed in Table 1. For example, the selector.types option controls whether or not selections for a given variable accumulate (single or multiple selected values). This sort of logic has also been interpreted as transient versus persistent selection (D. Cook and Swayne 2007).

```
viz$selector.types <- list(
  year = "single",
  country = "multiple",
  region = "multiple"
)
```

The code above declares year as a single selection variable, which means that only a single

Figure 3: Animint provides a menu to update each selection variable. In this example, after typing "th" the country menu shows the subset of matching countries.

year may be selected at a time (clicking a geom with `clickSelects=year` will change the selection to the corresponding year). The `country` and `region` variables are declared as multiple selection variables, which can have multiple selected values at a time (clicking a geom with `clickSelects=country` will add/remove that country to/from the selection set).

3.4 Direct versus indirect manipulation

When viewing the interactive version of Figure 2, suppose we wish to select Thailand. Direct manipulation is not very useful in this case since it is not easy to identify and select Thailand based on graphical marks on a plot. For this reason, **animint** also provides dropdown menu(s) for each selection variable to aid the selection process. Figure 3 shows what the user sees after typing ''th" in the search box. Note that these dropdowns support selection of multiple values and coordinate sensibly with selections made via direct manipulation.

3.5 Animation and smooth transitions

Animation is declared using the `time` option, which specifies a selection variable that will be automatically updated over time, as well as a time delay in milliseconds. The code below declares the `year` variable to be animated every 3 seconds.

```
viz$time <- list(variable = "year", ms = 3000)
```

Animation is useful in the World Bank data visualization because it shows how the bi-variate relationship between fertility rate and life expectancy changes over time. Animation clearly shows how many countries progress from low life expectancy and high fertility rate in early years, to high life expectancy and low fertility rate in later years.

Finally, the `duration` option specifies the amount of time used to smoothly transition between selections (with linear easing). Smooth transitions help the viewer track geoms before and after an update to the selection set. For example in the code below we declare a 1 second smooth transition on the `year` variable, in order to more easily track the points on the scatterplot.

```
viz$duration <- list(year = 1000)
```

Note that for accurate interpretation of smooth transitions, the new `key` aesthetic must be specified. The `key` aesthetic is used to match data elements before and after the smooth transition. In the World Bank example, we would need to specify `aes(key=country)` for the points and text in the scatterplot.

3.6 Storing and re-storing state

When sharing an interactive visualization with others, it can often be helpful to share interesting state(s) of the visualization. In **animint**, states can be serialized in a URL link and/or specified at the command line via the `first` option. The code below declares that the first selection of the `country` variable is the set of two countries, United States and Vietnam.

```r
viz$first <- list(country = c("United States", "Vietnam"))
```

3.7 Compiling and rendering

Supplying the `viz` list of ggplots and rendering options to the `animint2dir()` function will save all the files necessary for rendering the visualization:

```r
animint::animint2dir(viz)
```

As shown in Supplementary Figure 1, the **animint** system consists of 2 parts: the compiler and the renderer. The compiler is `R` code that converts a list of ggplots and options to a JSON plot meta-data file and a tab-separated values (TSV) file database. The renderer consists of HTML and JavaScript files, which can be easily hosted along with the TSV and JSON files on any web server. The interactive plots can be viewed by opening the `index.html` page in any modern web browser. Note that **animint** currently depends on a fork of **ggplot2**[2] that contains some minor modifications which are needed to support interactive rendering on web pages. Additional implementation details are available in the supplementary materials.

4 Exploring scope with examples

This section attempts to demonstrate the range of visualizations that are supported by the system we propose. In particular because of its support for interaction and animation, it excels at display of interactive maps with time-varying data. We give two such examples below.

4.1 Tornadoes in the United States

One of the strong points of the system we propose is display of multi-layer plots such as maps with time-varying data. For example, Figure 4 shows a visualization of US tornado

[2] https://github.com/faizan-khan-iit/ggplot2/tree/validate-params

data from 1950 to 2012. This data visualization consists of two multi-layer plots with two interaction variables, `year` and `state`.

The left plot is a map which shows state borders using a polygon with `clickSelects=state`. The currently selected state is shown using semi-transparency, and other states can be selected by clicking them. The state map plot uses geoms with `showSelected=year` to show tornado paths (segment geom) and endpoints (point geom) for the currently selected year (which is emphasized with a text geom above the map).

The right plot uses several geoms to show details for the currently selected state and year. A bar geom shows a time series of tornado counts for the selected state (`showSelected=state`), which can be clicked to change the currently selected year (`clickSelects=year`). A text geom at the top of the plot shows the currently selected state (`showSelected=state`), and a text geom at the bottom emphasizes the tornado count for the selected year (using `showSelected` variables for both `state` and `year`).

These interactions can be useful for discovering patterns in the data, and for suggesting models that can describe or predict tornado paths.

4.2 Central American climate data

A more complex map data visualization example is shown in Figure 5, which depicts climate time series data observed in Central America. There are two interaction variables, `time` and `region`.

Two maps in the upper left display borders of the countries in and near Central America. Unlike the previous example with US states, the country borders are static (clicking has no effect). For the currently selected time, rect geoms with `showSelected=time` show the spatial distribution of sea surface temperature as well as its deviation from the monthly norm. Since `clickSelects=region` is specified, clicking a rect changes the currently selected region, which is emphasized with a black border. These plots facilitate visualization of the spatial distribution of the climate variables, and how they change over time.

The plots below the maps use lines to show time series of the climate variables. Since

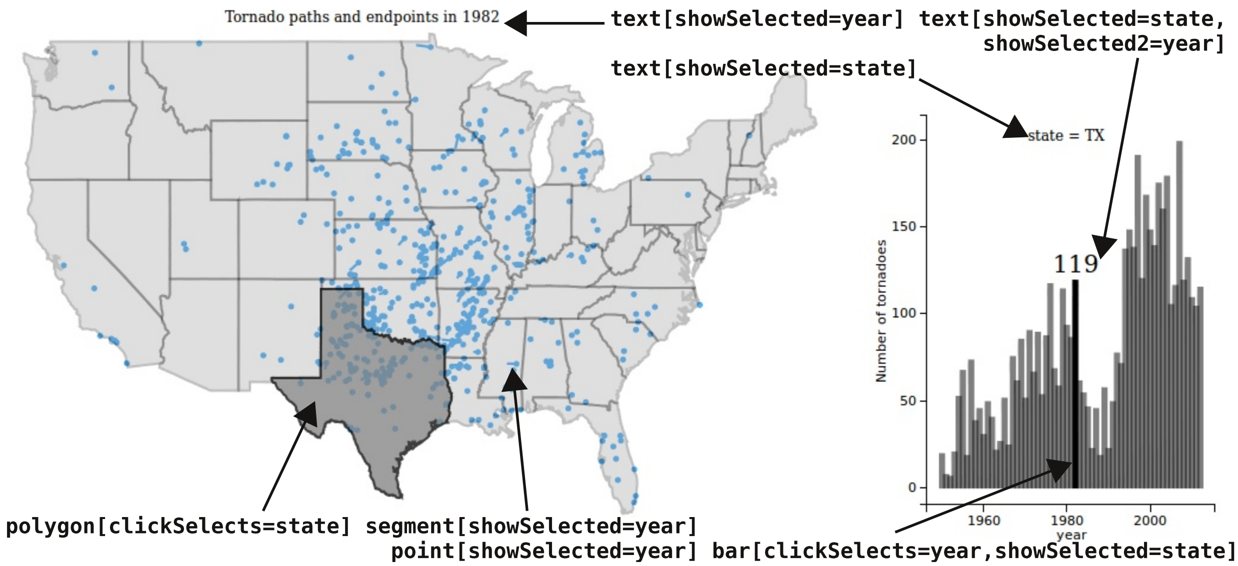

Figure 4: Interactive animation of US tornadoes from 1950 to 2012. This figure depicts a scenario where the user queried Texas (by clicking the map), and the year 1982 (by clicking the bar chart). In addition to the graphical elements being highlighted as a visual clue of what query is being made, this visualization includes dynamic text labels reflecting the query.

`clickSelects=region` is specified, clicking a line changes the currently selected region, which is emphasized with a purple color. A semi-transparent tallrect shows the currently selected time; other tallrects can be clicked to update the time (`clickSelects=time`). These plots make it easy to select different times and regions, and to make comparisons between times and regions.

Scatterplots on the right use `showSelected` variables with point and text geoms, to show the joint distribution of the two temperature variables for the selected time (top) and region (bottom). The plots use `clickSelects` to emphasize the currently selected region (top) and time (bottom), and are useful for visualizing normality and outliers in the joint distribution.

5 Limitations and future work

The system we have proposed enables graphical queries and animation via an extension to **ggplot2** using entirely client-side web technologies. Often times in interactive statistical graphics, we wish to dynamically compute a statistical aggregation(s) as a function of a user

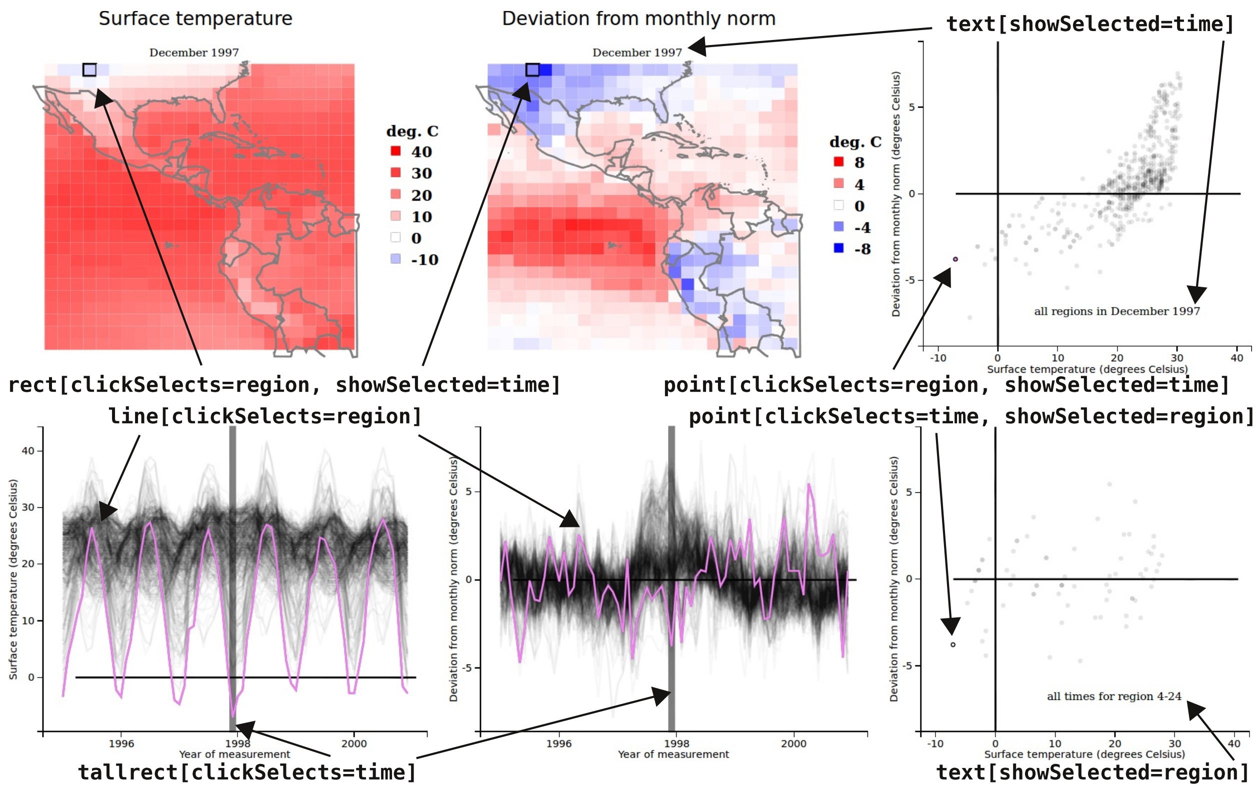

Figure 5: Visualization containing 6 linked, interactive, animated plots of Central American climate data. Top: for the selected time (December 1997), maps displaying the spatial distribution of two temperature variables, and a scatterplot of these two variables. The selected region is displayed with a black outline, and can be changed by clicking a rect on the map or a point on the scatterplot. Bottom: time series of the two temperature variables with the selected region shown in violet, and a scatterplot of all times for that region. The selected time can be changed by clicking a background tallrect on a time series or a point on the scatterplot. The selected region can be changed by clicking a line on a time series.

selection (e.g., compute a new linear model based on a set of newly brushed points). Using such a client-side linking framework requires one to pre-compute every possible aggregation ahead of time. If the number of states is fairly small, this isn't much of a problem, but it doesn't scale very well. For an example of pre-computing states and exploring those limitations, see the touring example in the supplementary materials.

It's possible that future implementations could dynamically recompute statistical aggregated specified via a **ggplot2** layer, but it would be an immense amount of work to translate so many `R` functions to `JavaScript`. In this scenario, it may be worthwhile to try compile `R` functions so that the web browser can run them using a technology like `WebAssembly`

Our current implementation provides a visual indication of the current selection via semi-transparency of `clickSelects` geoms. In future work we would like to explore more obvious visual cues that can be used to quickly show the user the links between plots and possible interactions.

A number of limitations in our current implementation derive from the fact that some plot features are computed once during the compilation step, and remain static on a rendered plot. For example, users are unable to change variable mappings after compilation. Also, when different data subsets have very different ranges of values, it may be preferable to recompute scales when `clickSelects` selection(s) change. Some of these limitations can be resolved by adding interactive widgets to "recompile" components hard-coded in the plot meta information. In fact, **animint** makes it easy to embed visualizations inside of **shiny** web applications, and we have an example of interactively redefining variable mappings.

Our compiler also currently takes advantage of **ggplot2** internals to compute statistics and positional adjustments before rendering. As a result, statistics/positions will not dynamically recompute based on selections. In other words, using `clickSelects/showSelected` with non-identity statistic(s)/position(s) may not generate a sensible result. It would be possible, but a significant amount of work, to transfer these computations from the compiler to the renderer.

Another set of limitations derive from our current restriction that all subsets (corresponding to each possible selection) must be precomputed before render time. As elucidated in our supplementary materials, if there is a large space of possible selections, it is impractical to precompute every subset before viewing. Therefore, for future work it would be useful if the renderer could dynamically compute subsets when new selections are made.

Our implementation is also limited to two specific types of direct manipulation: selecting graphical elements via mouse click (`clickSelects`), and showing/hiding related elements (`showSelected`). However, the framework described in Section 3.1 is not restricted to a particular event type, so `hoverSelects` and `brushSelects` aesthetics could be added, for instance. There are other types of interaction that could be added, that wouldn't require additional extensions to the Grammar of Graphics, such as: zooming, panning, and plot re-sizing.

6 Conclusion

We have proposed several extensions to **ggplot2**'s layered grammar of graphics in order to support a declarative approach to producing interactive and dynamic web graphics. By adding `clickSelects` and `showSelected` aesthetics to specify selection source(s) and target(s), **ggplot2** users can quickly and easily create animations with smooth transitions and perform dynamic queries via direct manipulation of linked views. As a result, **animint** is a useful tool not only for EDA, but also for the presentation and distribution of interactive statistical graphics.

Interactive figures and reproducible research statement

The source code to create this paper and its figures is online at https://github.com/tdhock/animint-paper/ and the interactive figures can be viewed at http://members.cbio.mines-paristech.fr/~thocking/animint-paper-figures/

Acknowledgements

The authors wish to thank **animint** users MC Du Plessis, Song Liu, Nikoleta Juretic, and Eric Audemard who have contributed constructive criticism and helped its development.

References

Ahlberg, Christopher, Christopher Williamson, and Ben Shneiderman. 1991. "Dynamic Queries for Information Exploration: An Implementation and Evaluation." In *ACM Chi '92 Conference Proceedings*, 21:619–26.

Andreas Buja, Catherine Hurley, Daniel Asimov, and John A. McDonald. 1988. "Elements of a Viewing Pipeline for Data Analysis." In *Dynamic Graphics for Statistics*, edited by William S. Cleveland and Marylyn E. McGill. Belmont, California: Wadsworth, Inc.

Becker, RA, and WS Cleveland. 1987. "Brushing Scatterplots." *Technometrics* 29 (2):127–42.

Becker, Richard A., William S. Cleveland, and Ming-Jen Shyu. 2010. "The Visual Design and Control of Trellis Displays." *Journal of Computational and Graphical Statistics* 19 (1). Taylor & Francis:3–28.

Bostock, Michael, Vadim Oglevetsky, and Jeffrey Heer. 2011. "D3 Data-Driven Documents." *IEEE Transactions on Visualization and Computer Graphics* 17 (12):2301–9.

Buja, Andreas, John Alan McDonald, John Michalak, and Werner Stuetzle. 1991. "Interactive data visualization using focusing and linking." *IEEE Proceedings of Visualization*, February, 1–8.

Cairo. 2016. "Cairo: A Vector Graphics Library." http://cairographics.org/.

Chang, Winston, and Hadley Wickham. 2015. *ggvis: Interactive Grammar of Graphics*. https://CRAN.R-project.org/package=ggvis.

Cheng, Joe. 2016. *Crosstalk: Inter-Widget Interactivity for Html Widgets*. https://CRAN.R-project.org/package=crosstalk.

Cook, Dianne, Andreas Buja, and Deborah F Swayne. 2007. "Interactive High-Dimensional Data Visualization." *Journal of Computational and Graphical Statistics*, December, 1–23.

Cook, Dianne, and Deborah F. Swayne. 2007. *Interactive and Dynamic Graphics for Data Analysis : With R and GGobi*. Use R ! New York: Springer. http://www.ggobi.org/book/.

Duncan Temple Lang, Hadley Wickham, Debby Swayne. 2016. *Interface Between R and GGobi*.

Fisherkeller, Friedman, M. A., and J. W. Tukey. 1988. "PRIM-9, an Interactive Multidimensional Data Display and Analysis System." In *Dynamic Graphics for Statistics*, 91–109.

Hofmann, Heike, and Martin Theus. 1998. "Selection sequences in MANET." *Computational Statistics*, January, 1–12.

Lawrence, Michael, and Deepayan Sarkar. 2016. *Interface Between R and Qt*. https://github.com/ggobi/qtbase.

Lawrence, Michael, and Duncan Temple Lang. 2010. "RGtk2: A Graphical User Interface Toolkit for R." *Journal of Statistical Software* 37 (8):1–52. http://www.jstatsoft.org/v37/i08/.

Murrell, Paul, and Simon Potter. 2015. *gridSVG: Export 'grid' Graphics as SVG*. https://CRAN.R-project.org/package=gridSVG.

Nolan, Deborah, and Duncan Temple Lang. 2012. "Interactive and Animated Scalable Vector Graphics and R Data Displays." *Journal of Statistical Software* 46 (1):1–88. http://www.jstatsoft.org/v46/i01/.

Pedersen, Thomas Lin. 2016. *Tweenr: Interpolate Data for Smooth Animations*. https://CRAN.R-project.org/package=tweenr.

Plotly. 2015. "Collaborative Data Science." Montreal, QC: Plotly Technologies Inc. 2015. https://plot.ly.

R Core Team. 2017. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. http://www.R-project.org/.

Robinson, David. 2016. *gganimate: Create easy animations with ggplot2*. http://github.com/dgrtwo/gganimate.

RStudio. 2013. "Shiny: Easy Web Applications in R." http://www.rstudio.com/shiny/.

Swayne, DF, D Cook, and A Buja. 1998. "XGobi: Interactive Dynamic Data Visualization in the X Window System." *Journal of Computational and Graphical Statistics* 7 (1):113–30.

Theus, M. 2002. "Interactive Data Visualization Using Mondrian." *Journal of Statistical Software* 7 (11):1–9. http://www.jstatsoft.org/v07/i11.

Tierney, Luke. 1990. *LISP-Stat: An Object Oriented Environment for Statistical Computing and Dynamic Graphics.* Wiley-Interscience, New York.

Trifacta. 2014. "Vega: A Declarative Visualization Grammar." http://trifacta.github.io/vega/.

Unwin, Antony, and Heike Hofmann. 2009. "GUI and Command-line - Conflict or Synergy?" *Proceedings of the St Symposium on the Interface*, September, 1–11.

Unwin A., Hofmann H., Hawkins G. 1996. "Interactive Graphics for Data Sets with Missing Values - Manet." *Journal of Computational and Graphical Statistics* 4 (6).

Urbanek, Simon. 2011. "IPlots eXtreme: Next-Generation Interactive Graphics Design and Implementation of Modern Interactive Graphics." *Computational Statistics* 26 (3):381–93. https://doi.org/10.1007/s00180-011-0240-x.

———. 2016. *rJava: Low-level R to Java Interface.* https://CRAN.R-project.org/package=rJava.

Vaidyanathan, Ramnath, Yihui Xie, JJ Allaire, Joe Cheng, and Kenton Russell. 2018. *htmlwidgets: HTML Widgets for R.* https://github.com/ramnathv/htmlwidgets.

Waddell, Adrian, and R. Wayne Oldford. 2018. *loon: Interactive Statistical Data Visualization.* https://CRAN.R-project.org/package=loon.

Wickham, Hadley. 2010. "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics* 19 (1). Taylor & Francis:3–28.

Wickham, Hadley, Dianne Cook, Heike Hofmann, and Andreas Buja. 2011. "tourr: An R Package for Exploring Multivariate Data with Projections," April, 1–18.

Wickham, Hadley, Michael Lawrence, Dianne Cook, Andreas Buja, Heike Hofmann, and Deborah F Swayne. 2010. "The Plumbing of Interactive Graphics." *Computational Statistics*, April, 1–7.

Wilkinson, Leland, D Wills, D Rope, A Norton, and R Dubbs. 2006. *The Grammar of Graphics*. Springer.

Xie, Yihui. 2013. "animation: An R Package for Creating Animations and Demonstrating Statistical Methods." *Journal of Statistical Software* 53 (1):1–27. http://www.jstatsoft.org/v53/i01/.

Xie, Yihui, Heike Hofmann, Di Cook, Xiaoyue Cheng, Barret Schloerke, Marie Vendettuoli, Tengfei Yin, Hadley Wickham, and Michael Lawrence. 2013. *Interactive Statistical Graphics Based on Qt*.