

Extending ggplot2 for linked and dynamic 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 8, 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. Web technologies, however, are not designed to fit inside an exploratory data analysis workflow where rapid iteration between data manipulation, modeling, and visualization must occur. 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 users to use their existing **ggplot2** code, then leverage an idiomatic API for describing animations and graphical queries between multiple linked views.

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

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), **Dataviewer** (Andreas Buja and McDonald 1988), **XGobi** (Swayne 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), **loon** (Waddell and Oldford 2018), etc, require a heavy set of computational dependencies in order to view or interact with graphics. Although this allows the freedom to easily leverage libraries with sophisticated statistical functionality, it limits the ability to share or embed such graphics in a larger document. On the other hand, interactive web graphics are now ubiquitous because of their accessibility and composability.

Generally speaking, web graphics that use purely “client-side” technologies (i.e., **HTML**, **SVG**, **CSS**, and **JavaScript**) are desired for their sharability, scalability, and maintainability. This is why many web-based graphing libraries work entirely with client-side technologies, like **Vega** (Trifacta 2014), **plotly.js**, and **bokeh**.

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; but when necessary, one can leverage a client-server model.

Table X gives a high-level depiction of the different infrastructure required for a purely client-side (i.e. standalone) webpage versus a client-server web application. Focusing just on the R language, there are now a number of ways to develop web applications powered by R, including **FastRWeb** (Urbanek and Horner 2015), **opencpu** (Ooms 2014), and **plumber** (Trestle Technology, LLC 2017), but **shiny** (RStudio 2013) is most widely used since it provides a nice reactive programming framework in addition to client-server communication.

- Make an argument that we need more R -> HTML interfaces that don't require a callback to R.

D. Cook, Buja, and Swayne (2007) taxonomy motivated by data analysis tasks

Interactive graphics are an important part of applied statistical practice, because they can lead to a better understanding of high-dimensional data sets and models. Interactive graphics are useful in the context of EDA, model selection/validation, and presenting results of an analysis. Interactive graphics toolkits in R have been available for decades, but these approaches are often not easy to reproduce or distribute to a larger audience. It is true that most graphics generated during EDA are ultimately not useful, but sometimes, understanding gained during this phase is most easily shared via the interactive graphics themselves. Thus, there is value in being able to easily share and embed interactive graphics inside a larger report. Unfortunately, this is typically hard, if not impossible, using traditional interactive graphics toolkits. As a result, there is a large disconnect between the visualization tools that we use for exploration versus presentation.

One of the most widely used R packages is **ggplot2** (Wickham 2009), a data visualization package inspired by the grammar of graphics (Wilkinson et al. 2006). In fact, Donoho (2015) writes: “This effort may have more impact on today’s practice of data analysis than many highly-regarded theoretical statistics papers”. In our experience, **ggplot2** has made an impact thanks to its foundation in the grammar of graphics, carefully chosen defaults, and overall usability. This helps data analysts rapidly iterate and discover informative visualizations – an essential task in exploratory data analysis (EDA). When dealing with high-dimensional data, however, it is often useful to produce interactive and/or dynamic graphics, which **ggplot2** does not inherently support.

We aim to narrow this gap in visualization tools by extending **ggplot2**’s grammar of graphics implementation for interactive and dynamic web graphics. Our extension allows one to create animated transitions and perform dynamic queries via direct manipulation of linked views like those described in Ahlberg, Williamson, and Shneiderman (1991) and Buja et al. (1991). A conceptual model for our extension is provided in Section 3.1 and Section 3.3. In Section 3.2, we demonstrate our extension with an example. In Section ??, we outline design decisions

made in our implementation in the R package **animint**. In Section 4, we provide a sense of the scope of our system and its performance limitations through a handful of examples. In Section 5, we conduct a comparison study by replicating examples with other leading systems. Finally, in Section 6, we discuss future work and limitations of our current system.

2 Related work

We aim to provide a system which empowers **ggplot2** users to go beyond the confines of static graphics with minimal friction imposed upon their current workflow. We acknowledge that numerous systems which support similar visualization techniques exist outside of the R ecosystem, but we intentionally focus on R interfaces since the surrounding statistical computing environment is crucial for enabling an efficient EDA workflow.

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 sacrifices in performance, using the modern web browser as a canvas is more portable, accessible, and composable (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 3.3. 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 touring example discussed in Section 5.1. The **tweenr** package provides similar smooth transitions, by computing data values in R that interpolate between animation frames (in **animint**, these calculations are performed in the web browser).

The **ggvis** package is similar to our system in that it is also 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 (Trifacta 2014), 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 does come with a cost, though, as sharing and hosting ggvis graphics typically requires special web server softwares, even when the interaction logic could be handled entirely client-side. As we outline in Section ??, our system does not require a web server, but can also be used inside **shiny** web applications, when desired.

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

Feature	Description
<code>clickSelects</code>	aesthetic for a geom which can be clicked to update the current selection.
<code>showSelected</code>	aesthetic for a geom which plots only the data corresponding to the current selection.
<code>selector.types</code>	global option to specify single or multiple selection for each interaction variable.
<code>first</code>	global option to specify first selection.
<code>time</code>	global option to specify delay between animation frames.
<code>duration</code>	global option to specify smooth transitions.
<code>key</code>	aesthetic specifying variable to use for matching data in smooth transitions.

The tour is a useful visualization technique for exploring high-dimensional data which requires interactive and dynamic graphics. 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`** (Duncan Temple Lang 2016) provides an R interface to `ggobi`’s graphical interface, but unfortunately, the software requirements for installation and use of this toolchain are heavy and stringent. 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 (Xie et al. 2013). **`Cranvas`** also has heavy and stringent software requirements which limits the portability and accessibility of the software.

Another R package for interactive graphics is **`iplots`** (Urbanek 2011), which has several important differences compared to **`animint`**. Brushing/highlighting 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 propose several extensions to the layered grammar of graphics (Table~1). Our extensions enable declarative expression of animations and dynamic queries via direct manipulation. 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 easily understood 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 in Ahlberg, Williamson, and Shneiderman (1991) and Buja et al.

(1991).

3.1 Direct manipulation of dynamic queries

Direct manipulation, as discussed in Ahlberg, Williamson, and Shneiderman (1991), is a graphical interface for interacting with databases. Direct manipulation interfaces include a graphical representation of queries to the database, which can be manipulated using the mouse. In the context of statistical graphics, direct manipulation refers to interfaces in which clicking the plotted representation of the data (such as lines or points) changes the plot. In contrast, indirect manipulation interfaces use widgets (such as buttons or menus) to change the plot.

D. Cook and Swayne (2007) use SQL queries to formalize the direct manipulation methods discussed in Ahlberg, Williamson, and Shneiderman (1991) and Buja et al. (1991). We propose to embed this direct manipulation framework inside the layered grammar of graphics with two new aesthetics, `clickSelects` and `showSelected`.

- `clickSelects` defines a selection source via a mouse click. For example `aes(clickSelects=year)` designates a plot element which, when clicked, will change the selected value of the `year` variable.
- `showSelected` defines a selection target. For example `aes(showSelected=year)` means to only show a plot element for data with the currently selected value of the `year` variable.

Below, we use **animint** to create a linked view between a bar chart and a scatter plot, where the user can click on bars to control the points shown in the scatterplot. This is shown in the video in Figure 1. As a result, we can quickly see how the relationship among tip amount and total bill amount depends on whether the customer is smoker.

```
gg.bar <- ggplot() + geom_bar(  
  data = reshape2::tips,  
  aes(x = smoker, clickSelects = smoker)  
)
```

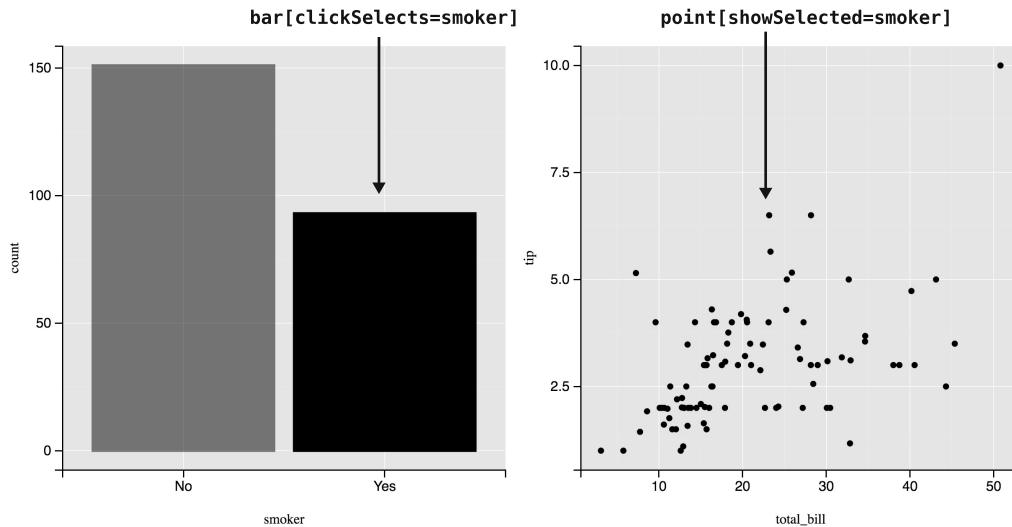



Figure 1: Linked dynamic querying via direct manipulation using animint. The `clickSelects` aesthetic designates a clickable geom that can change a selection variable, and the `showSelected` aesthetic designates a geom that responds by showing only the data which corresponds to the current selection. A video demonstration can be viewed online at <https://vimeo.com/160496419>

```
gg.scatter <- ggplot() + geom_point(
  data = reshape2::tips,
  aes(x = total_bill, y = tip,
      showSelected = smoker)
)
```

In the R code above, the proposed `showSelected` and `clickSelects` aesthetics are used to link the two ggplots, since they refer to a common variable, `smoker`. Each variable that is used in a `clickSelects` or `showSelected` aesthetic is treated as a selection variable with an interactively updated set of selected values that is used to dynamically update the plots in response to user input. In the first plot above, we have used `aes(clickSelects=smoker)` to specify a bar with direct manipulation (mouse clicks on the barplot) that dynamically changes the `smoker` selection variable. In the second plot above, we have used `aes(showSelected=smoker)` to specify that we only want to show data points for the current selection of the `smoker` variable. In response to a user clicking on the bars in the first plot, our system essentially performs

the SQL query below in order to generate the data to display in the second plot:

```
SELECT * FROM tips
WHERE smoker IN smoker_selection
```

In this example, `smoker_selection` is either “Yes” or “No” (a single selected value), but as we show in later examples, `smoker_selection` can also be an array of values (multiple selected values). Although the `clickSelects` aesthetic is tied to a mouse click event, other aesthetics could easily be created to support other selection events, such as hover or click+drag. Statistically speaking, this type of visualization 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

Figure 2 shows an interactive animation of the World Bank data set created with our **animint** implementation (World Bank 2012). The visualization helps us explore the change in the relationship between life expectancy and fertility over time for 205 countries. By default, the year 1979 and the countries United States and Vietnam are selected, but readers are encouraged to watch the video of the animation and/or interact the visualization using a web browser.¹ In the interactive version, the selected value of the year variable is automatically incremented every few seconds, using animation to visualize yearly changes in the relationship between life expectancy and fertility rate.

We anticipate that some **ggplot2** users will be able to reverse engineer the **animint** 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

¹<http://bl.ocks.org/tdhock/raw/8ce47eebb3039263878f/>

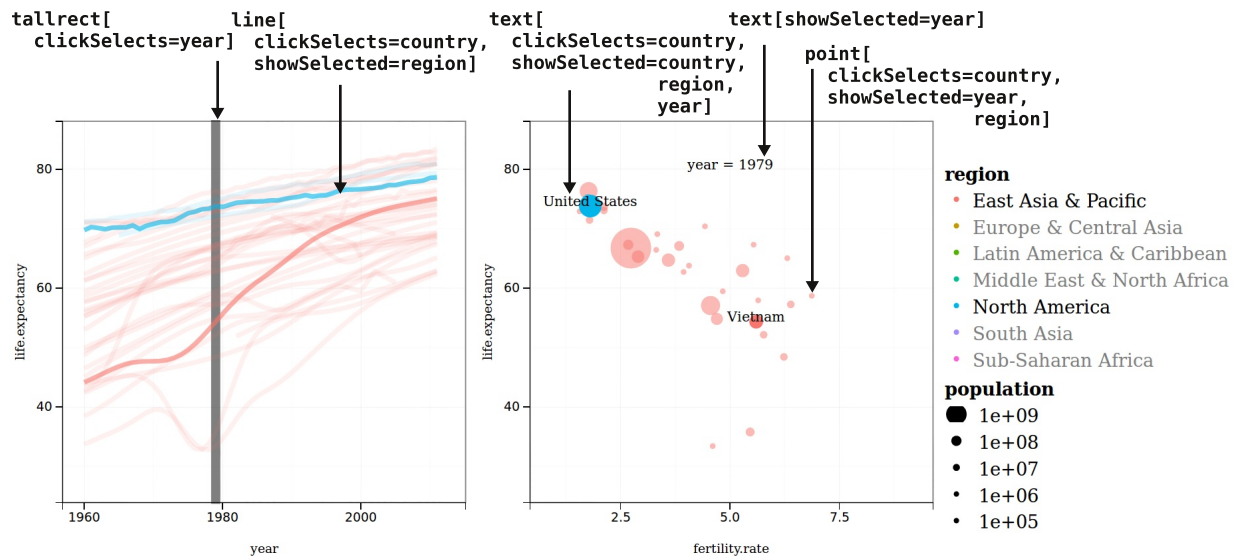


Figure 2: An interactive animation of World Bank demographic data of several countries, designed using `clickSelects` and `showSelected` keywords (top). Left: a multiple time series from 1960 to 2010 of life expectancy, with bold lines showing the selected countries and a vertical grey tallrect showing 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}

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

We want to provide a visual cue for the selected year in the time series, so in the code below we add some 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 year_selection
AND region IN region_selection
```

Below, the text layer for annotating selected countries is essentially the same as the point layer, except we map 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)
)
```

3.3 Linking, animation and other options

Linking is declared 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 as the first two elements of the following list:

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

Note that the animint `viz` list above can also contain rendering options, as we discuss below (see Table~1 for a summary of new interactive features in **animint**). For example, the `selector.types` option controls whether or not selections for a given variable can accumulate (single or multiple selected values).

```
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 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).

Another example is the `first` option, which controls selected values when the visualization is first rendered. The code below declares that the first selection of the `country` variable is the set of two countries, United States and Vietnam.

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

Animation is declared using the `time` option, which specifies a selection variable which 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 bivariate 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.

3.4 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:

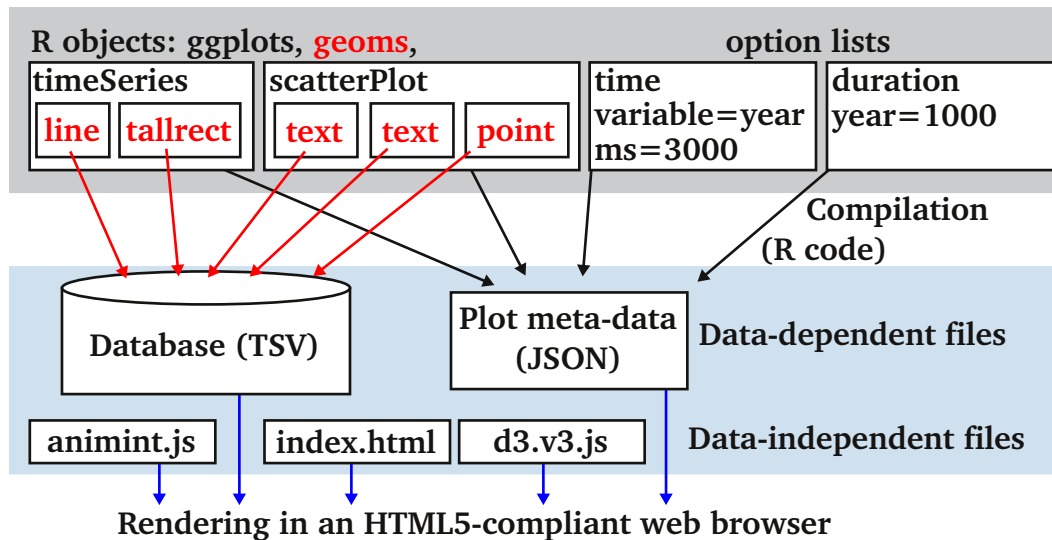


Figure 3: A schematic explanation of compilation and rendering in the World Bank visualization. Top: the interactive animation is a list of 4 R objects: 2 ggplots and 2 option lists. Center: **animint** R code compiles data in ggplot geoms to a database of TSV files (\rightarrow). It also compiles plot meta-data including ggplot aesthetics, animation time options, and transition duration options to a JSON meta-data file (\rightarrow). Bottom: those data-dependent compiled files are combined with data-independent JavaScript and HTML files which render the interactive animation in a web browser (\rightarrow).


```
animint2dir(viz)
```

As shown in Figure 3, the **animint** system is implemented in 2 parts: the compiler and the renderer. The compiler is implemented in 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 our current implementation of **animint** 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 performance & scope with examples

This section attempts to demonstrate the range of visualizations that are supported by **animint** with more examples.

4.1 Tornadoes in the United States

Figure 4 depicts an interactive animation with data from all US tornadoes from 1950 to 2012, but graphical queries restrict focus to tornado paths from 1982 and tornado counts within the state of Texas. In the interactive version, one may (1) click on any state to view its tornado counts throughout the years, (2) click on any bar to view the spatial distribution of tornado paths for that year, and (3) press play to animate the paths throughout time. These interactive techniques can be useful for discovering unusual patterns or even help to suggest a reasonable model for describing/predicting tornado paths.

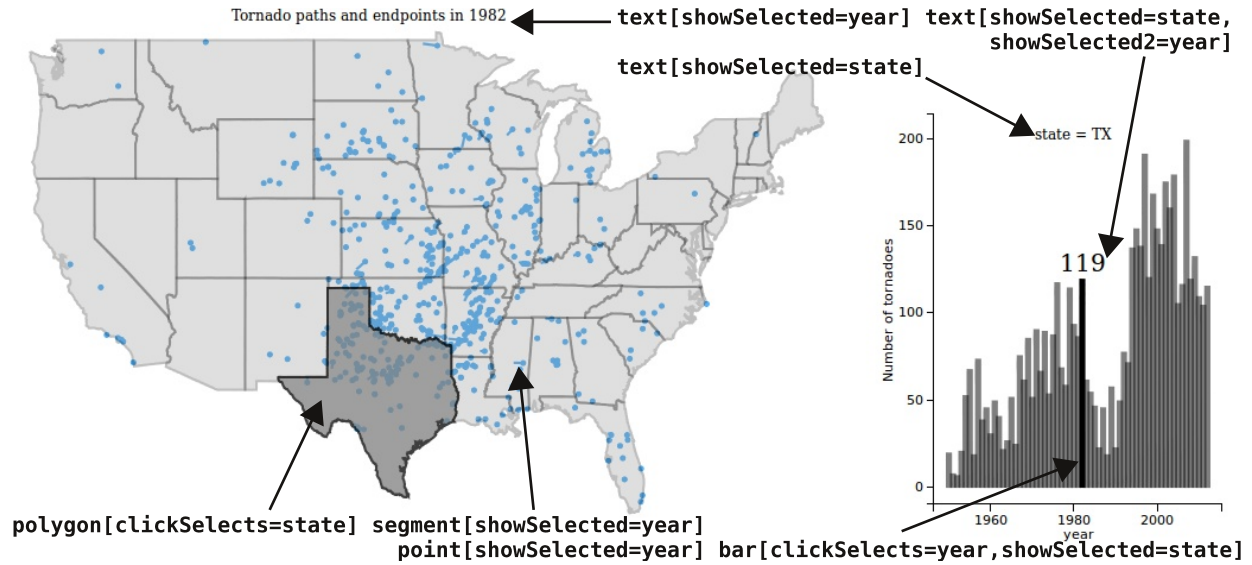


Figure 4: Interactive animation of US tornadoes from 1950 to 2012. This diagram depicts a scenario where the user queried Texas (by clicking the map), and the year 1982 (by clicking the bar chart). In addition to the graphical elements being (automatically) highlighted as a visual clue of what query is being made, this visualization includes dynamic labels reflecting the query. Particular queries may also be stored and shared via a URL, for example: <https://bl.ocks.org/faizan-khan-iit/raw/b3912f21ec2750f96e8d1bd4b66463b2/#year=%7B1982%7Dstate=%7BTX%7D>. In this link to the interactive version, we also demonstrate the ability to dynamically rescale axes when a new query is triggered. The middle and right panel display the same data, but use different scaling: the middle panel reflects the ‘global’ range (US) while the right panel reflects the ‘local’ range (Arkansas). Furthermore, when an axis update is triggered, it smoothly transitions from one state to next (preserving object constancy in the axis ticks). This helps the viewer better perceive/understand how the range has changed from one state to the next.

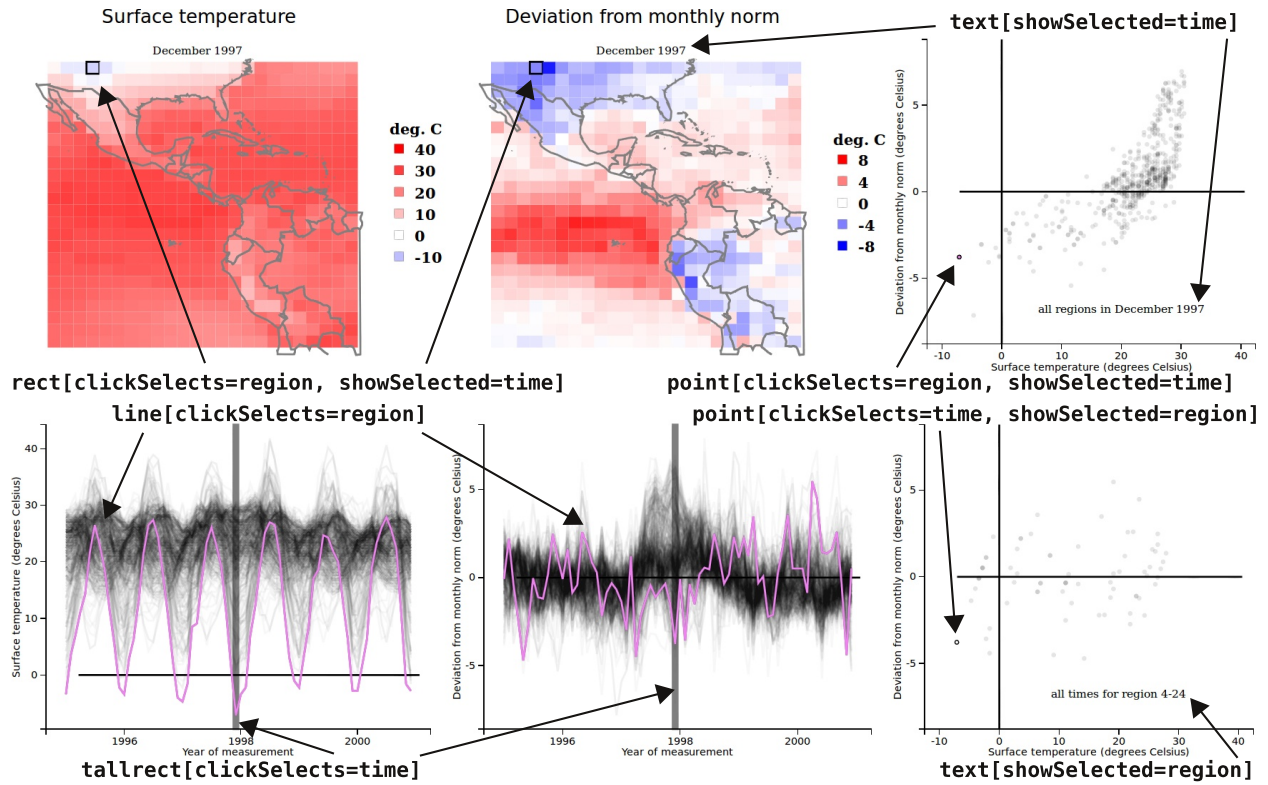


Figure 5: Visualization containing 6 linked, interactive, animated plots of Central American climate data. Top: for the selected time (December 1997), maps displaying the spatial distribution of two temperature variables, and a scatterplot of these two variables. The selected region is displayed with 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.

4.2 Central American Climate data

Figure 5 depicts an interactive animation of climate time series data observed in Central America. Two maps display the spatial distribution of sea surface temperature as well as its deviation from its monthly norm. Conditioning on a particular cell/region highlights that region’s corresponding time series below the map, allowing one to assess whether a particularly low/high value at one time point remains so over time. Scatterplots also show the relationships between the two temperature variables for the selected time and region, allowing us to further examine how unusual a particular query is with respect to both time and space.

Summary statistics describing complexity and performance of examples in this paper, as well as other **animint** examples, are displayed in Table 2. The climate data visualization has noticeably slow animations, since it displays about 88,980 geometric elements at once (<http://bit.ly/QcUrhn>). We observed this slowdown across all browsers, which suggested that there is an inherent bottleneck when rendering large interactive plots in web browsers using JavaScript and SVG. Another **animint** with a similar amount of total rows is based on the evolution data (<http://members.cbio.ensmp.fr/~thocking/animint/evolution/viz.html>), but since it shows less data onscreen (about 2,703 elements), it exhibits faster responses to interactivity and animation.

animint is still useful for creating interactive but non-animated plots when there is not a time variable in the data. In fact, 7 of the 11 examples in Table 2 are not animated. For example, linked plots are useful to illustrate complex concepts such as a change point detection model in the breakpoints data (<http://members.cbio.ensmp.fr/~thocking/animint/breakpoints/viz.html>). The user can explore different model parameters and data sets since these are encoded as **animint** interaction variables.

²<https://github.com/faizan-khan-iit/ggplot2/tree/validate-params>

Table 2: Characteristics of 11 interactive visualizations designed with **animint**. The interactive version of these visualizations can be accessed via <http://members.cbio.ensmp.fr/~thocking/animint/>. From left to right, we show the data set name and Figure number in this paper (Figure), the lines of R code (LOC) including data processing but not including comments (80 characters max per line), the amount of time it takes to compile the visualization (seconds), the total size of the uncompressed TSV files in megabytes (MB), the total number of data points (rows), the median number of data points shown at once (onscreen), the number of data columns visualized (vars), the number of `clickSelects/showSelected` variables (int), the number of linked panels (plots), if the plot is animated.

Figure	LOC	seconds	MB	rows	onscreen	vars	int	plots	animated?
worldPop	17	0.2	0.1	924	624	4	2	2	yes
WorldBank 2	20	2.3	2.1	34132	11611	6	2	2	yes
evolution	25	21.6	12.0	240600	2703	5	2	2	yes
change	36	2.8	2.5	36238	25607	12	2	3	no
tornado 4	39	1.7	6.1	103691	16642	11	2	2	no
prior	54	0.7	0.2	1960	142	12	3	4	no
compare	66	10.7	7.9	133958	2140	20	2	5	no
breakpoints	68	0.5	0.3	4242	667	13	2	3	no
climate 5	84	12.8	19.7	253856	88980	15	2	6	yes
scaffolds	110	56.3	78.5	618740	9051	30	3	3	no
ChIPseq	229	29.9	78.3	1292464	1156	44	4	5	no

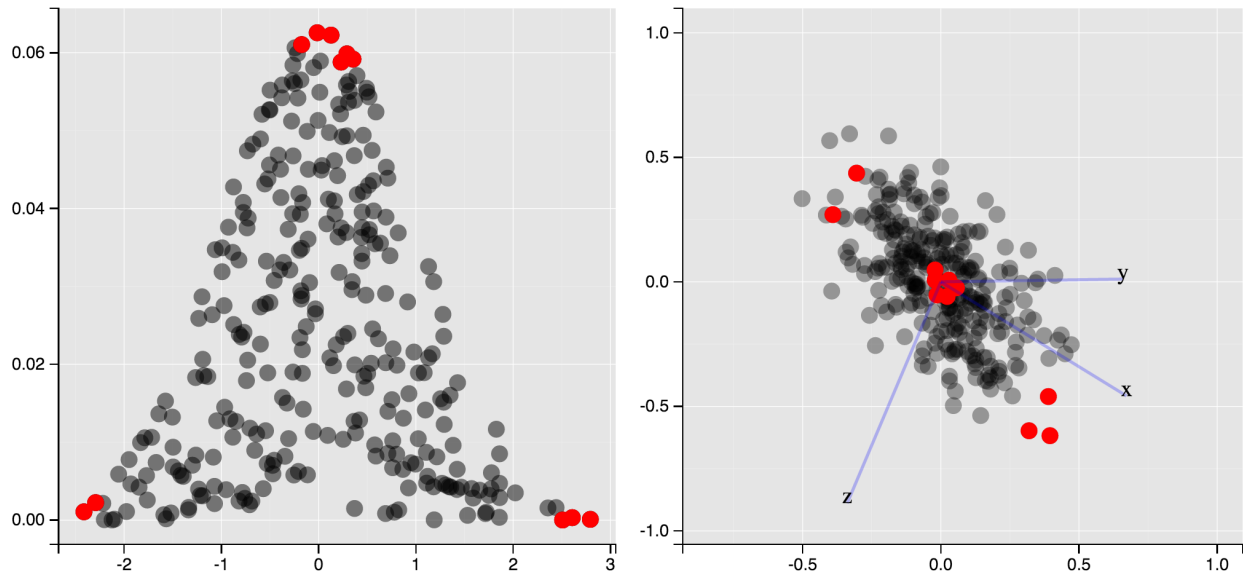


Figure 6: Linked selection in a grand tour with **animint**. A video demonstration can be viewed online at <https://vimeo.com/160720834>

5 Comparison study

In this section we compare our **animint** implementation with other similar leading systems by creating a given visualization in each system and discussing the pros and cons of the different approaches.

5.1 The Grand Tour

The Grand Tour is a well-known method for viewing high dimensional data which requires interactive and dynamic graphics (Asimov 1985). Figure 6 shows a grand tour of 300 observations sampled from a correlated tri-variate normal distribution. The left hand view shows the marginal density of each point while the right hand view “tours” through 2D projections of the 3D data. There are many ways to choose projections in a tour, and many ways to interpolate between projections, most of which can be programmed fairly easily using R and relevant add-on packages. In this case, we used the R package **tourr**, which uses the geodesic random walk (i.e., random 2D projection with geodesic interpolation) in its grand tour algorithm (Wickham et al. 2011).

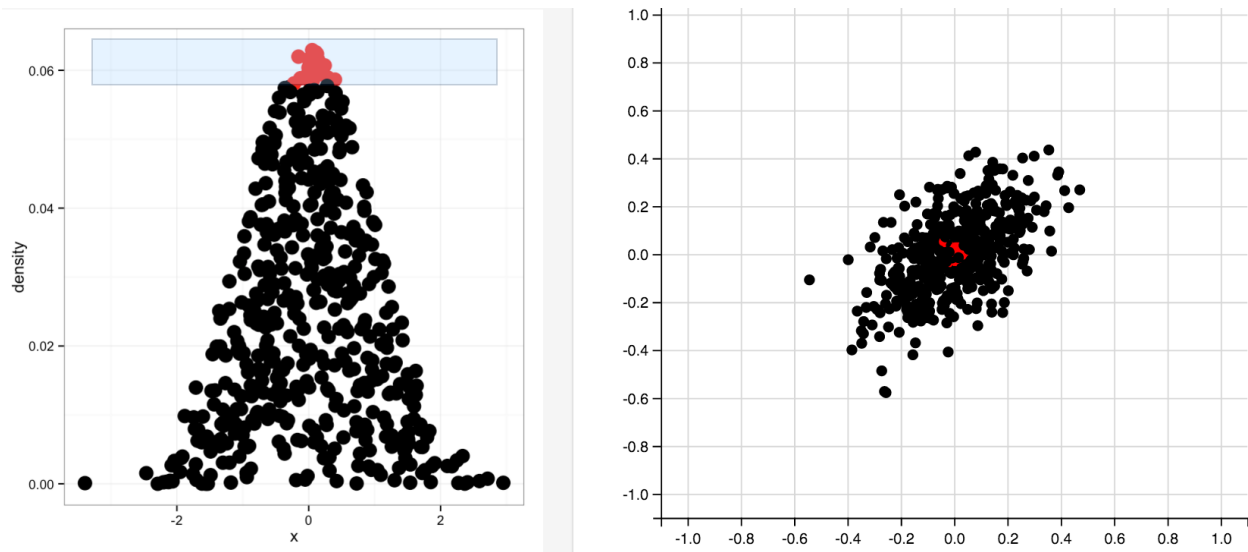


Figure 7: Linked selection in a grand tour with **ggvis** and **shiny**. A video demonstration can be viewed online at <https://vimeo.com/160825528>

When touring data, it is generally useful to link low-dimensional displays with the tour itself. The video in Figure 6 was generated with our current **animint** implementation, and points are selected via mouse click which reveals that points with high marginal density are located in the ellipsoid center while points with a low marginal density appear near the ellipsoid border. In this case, it would be convenient to also have brush selection, as we demonstrate in Figure 7 which implements the same touring example using the R packages **ggvis** and **shiny**. The brush in Figure 7 is implemented with **shiny**'s support for brushing static images, which currently does not support multiple brushes, making it difficult to select non-contiguous regions.

This example helps point out a few other important differences in using **animint** versus **ggvis+shiny** to implement “multiple linked and dynamic views” as described in Ahlberg, Williamson, and Shneiderman (1991) and Buja et al. (1991). Maintaining state of the linked brush in Figure 7 requires both knowledge and clever use of some sophisticated programming techniques such as closures and reactivity. It also requires knowledge of the **shiny** web application framework and a new approach to the grammar of graphics. On the other hand, maintaining state in Figure 6 requires a few different `clickSelects/showSelected` mappings. As a result, we believe **animint** provides a more elegant user interface for this application.

The touring example also helps point out important consequences of the design and implementation of these two different systems. As mentioned in Section ??, our current **animint** implementation requires every subset of data to be precomputed before render time. For visualizations such as tours, where it is more efficient to perform statistical computations on-the-fly, this can be a harsh restriction, but this is a restriction of our current implementation (not a restriction of the framework itself). As a result, when touring a large high-dimensional space, where many projections are needed, **ggvis+shiny** may be desirable since the projections are computed on the server and sent to the browser in real-time. This works fine when the application is running and viewed on the same host machine, but viewing such an application hosted on a remote machine can produce staggered animations since client-server requests must be performed, processed, and rendered roughly 30 times a second. Also, generally speaking, the **animint** system results a more pleasant experience when it comes to hosting and sharing applications since it doesn't require a Web Server with R and special software already installed.

5.2 World Bank example

We also recreated Figure 2 using **ggvis+shiny** (see <http://bit.ly/1SsJKlN>) and Tableau (see <http://bit.ly/worldBank-tableau>). Even as experienced **ggvis+shiny** users, we found it quite difficult to replicate this example, and were not able to completely replicate it due to a lack of a mechanism for coordinating indirect and direct manipulations. Overall the visualization is pretty similar, but lacks a few important features. In particular, there is no way to control the selected year using both the slider (indirect) and clicking on the ggvis plot (direct). It also lacks the ability to click on a country time series and label the corresponding point on the scatterplot. This might be possible, but we could not find a way to update a plot based on a click event on a different plot. Even with this lack of functionality, the **ggvis+shiny** is significantly more complicated and requires more code (about 100 lines of code compared to 30).

It was also impossible to completely replicate Figure 2 using Tableau essentially because the example requires a *layered* approach to the grammar of graphics. In particular, since

graphical marks and interaction source/target(s) must derive from the same table in Tableau, it was impossible to control the clickable multiple time series and the clickable tallrects in different ways based on the two different selection variables. In other words, in Tableau, selections are managed on the plot level, but in **animint**, selections are specific to each graphical layer.

6 Limitations and future work

The system we have proposed provides linked interactive plots via the new **showSelected** and **clickSelects** aesthetics. The linking between plots is rather flexible, but is limited to interactions which are specified by the plot designer at compile-time. 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 (<http://bit.ly/animint-shiny>).

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 Section 5.1, 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 resizing.

7 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.

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.
- Asimov, Daniel. 1985. “The Grand Tour: A Tool for Viewing Multidimensional Data.” *SIAM J. Sci. Stat. Comput.* 6 (1). Philadelphia, PA, USA: Society for Industrial; Applied Mathematics:128–43. <https://doi.org/10.1137/0906011>.
- 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>.
- 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/>.

- Donoho, David. 2015. “50 years of Data Science.” <https://dl.dropboxusercontent.com/u/23421017/50YearsDataScience.pdf>.
- Duncan Temple Lang, Hadley Wickham, Debby Swayne. 2016. *Interface Between R and 'Ggobi'*.
- Fisherheller, 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.
- 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/>.
- Ooms, Jeroen. 2014. “The Opencpu System: Towards a Universal Interface for Scientific Computing Through Separation of Concerns.” *arXiv:1406.4806 [stat.CO]*. <http://arxiv.org/abs/1406.4806>.
- 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, Cook, D. F., 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.
- Trestle Technology, LLC. 2017. *Plumber: An Api Generator for R*. <https://CRAN.R-project.org/package=plumber>.
- Trifacta. 2014. “Vega: A Declarative Visualization Grammar.” <http://trifacta.github.io/vega/>.
- 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>.
- Urbanek, Simon, and Jeffrey Horner. 2015. *FastRWeb: Fast Interactive Framework for Web Scripting Using R*. <https://CRAN.R-project.org/package=FastRWeb>.
- Waddell, Adrian, and R. Wayne Oldford. 2018. *Loon: Interactive Statistical Data Visualization*. <https://CRAN.R-project.org/package=loon>.
- Wickham, Hadley. 2009. *ggplot2: elegant graphics for data analysis*. Springer New York. <http://had.co.nz/ggplot2/book>.
- 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.
- World Bank. 2012. “World Development Indicators.” <http://data.worldbank.org/>

[data-catalog/world-development-indicators](#).

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*.