Extending ggplot2's grammar of graphics implementation for linked and dynamic graphics on the web

Carson Sievert, Toby Dylan Hocking, Susan VanderPlas, Jun Cai, Kevin Ferris

Abstract— The web is the most popular medium for sharing interactive data visualizations thanks to the portability of the web browser and the accessibility of the internet. Unfortunately, creating interactive web graphics often requires a working knowledge of numerous web technologies that are foreign to many people working with data. As a result, web graphics are rarely used for exploratory data analysis where quick iteration between different visualizations is of utmost importance. This is the core strength of ggplot2, a popular data visualization package for the world's leading open-source statistical programming language, R. The conceptual framework behind ggplot2 is based on the grammar of graphics, which lays a foundation for describing any static graphic as a small set of independent components. Perhaps the most fundamental component is the mapping from abstract data to the visual space, sometimes referred to as the aesthetic mapping. We propose adding two new aesthetics to the grammar, which together are sufficient for elegantly describing both animations and certain classes of coordinated linked views. We implement this extension in the open-source R package animint, which converts ggplot2 objects to interactive web visualizations via D3.

Index Terms—Interactive graphics, Statistical graphics, Exploratory data analysis, Web technologies

1 Introduction

The world's leading open source statistical programming language, R, has a rich history of interfacing with computational tools for the use of people doing data analysis and statistics research [R Core Team, 2015]. Understanding R's core audience is important, as they typically want to maximize their time working on data analysis problems, and minimize time spent learning computational tools. R excels in this regard, as it is designed specifically for interactive use, where users can quickly explore their data using highly expressive interfaces. Another key player in R's success story is its packaging infrastructure, which provides tools for distributing entire research conpendium(s) (code, data, documentation, auxiliary documents, etc) [Gentleman and Lang, 2004].

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 todays 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.

Interactive graphics toolkits in R have been used for decades to enhance the EDA workflow, 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 is 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.

We aim to narrow this gap in visualization tools by extending ggplot2's grammar of graphics implementation for interactive and dynamic web graphics. By adding two new aesthetics to the grammar, `clickSelects` and `showSelected`, we are able to express animated transitions and perform database queries via direct manipulation of linked views like those described in [Ahlberg et al., 1991] and [Buja et al., 1991]. Other aesthetics, such as `tooltip` and `href`, can be used to identify data values and link to web pages. The rest of the paper will proceed as follows: Section 3 discusses the `clickSelects` and `showSelected` aesthetics in detail, Section 4 discusses our current implementation in the R package animint, Section 5 compares our implementation to similar work, Section 6 provides a handful of examples, ...

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 maintaining a practical exploratory data analysis 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, 2015]. Neither grid, nor base R graphics, have strong support for handling user interaction creating a need for add-on packages. There are a number of approaches these packages take to render graphics, each with their own benefits and drawbacks. These packages traditionally build on low-level R interfaces to graphical systems such as GTK+ [Lawrence and Temple Lang, 2010], Qt [Lawrence and Sarkar, 2016a,b], or Java GUI frameworks [Urbanek, 2016]. In general, the resulting system can be very fast and flexible, but sharing and reproducing output is difficult due to the heavy software requirements. Although there may be sacrifice in performance, using the modern web browser as a canvas is more portable, accessible, and composable (graphics can be embedded within larger frameworks/documents).

- *Carson Sievert is with the Department of Statistics at Iowa State University. E-mail: sievert@iastate.edu*
- *Toby Dylan Hocking is with the McGill University department of Human Genetics. E-mail: toby.hocking@mail.mcgill.ca*
- *Susan VanderPlas is with the Department of Statistics at Iowa State University. E-mail: srvanderplas@gmail.com*
- *Jun Cai is with the Center for Earth System Science at Tsinghua University. E-mail: cai-j12@mails.tsinghua.edu.cn*
- *Kevin Ferris is with the Baseball Operations Department of the Tampa Bay Rays. E-mail: kevin.ferris10@gmail.com*

Manuscript received 31 March 2013; accepted 1 August 2013; posted online 13 October 2013; mailed on 4 October 2013.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

Base R does provide a Scalable Vector Graphics (SVG) device, `svg()`, via the Cairo graphics API [Cairo, 2016]. The R package SVGAnnotation [Nolan and Lang, 2012] provides functionality to post-process `svg()` output in order to add interactive and dynamic features. 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, anyone wishing to extend or alter the core functionality needs a deep understanding of base graphics and SVG.

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, if one wanted to use this interface to add interactivity to a ggplot2 plot, they must know and understand the naming scheme ggplot2 uses internally and hope it does not change down the road. 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 user interface, 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 3.2. 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 et al., 2011]. Smooth transitions help us track positions between frames, which is useful in many scenarios, such as the touring example discussed in Section 6.

Tours are 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 tool for touring data, but also provides the ability to produce more standard statistical graphics, and general interactive techniques such as linking, zooming, panning, and identifying [Cook and Swayne, 2007]. The R package rggobi [Wickham et al., 2008] 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 which draws design inspiration from the grammar of graphics is ggvis [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 may require special web server software, even when the interaction logic could be handled entirely client-side. As we outline in Section 4, our system does not require a web server, but can be used inside shiny web applications, when desired.

3 Extending the Layered Grammar of Graphics

In this section, we propose an extension to the layered grammar of graphics [Wickham, 2010] which enables declarative expression of animations and database queries via direct manipulation. In the ggplot2 system, there are five essential components that define a layer of graphical markings: 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 [Andreas Buja and McDonald, 1988, 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 et al., 1991] and [Buja et al., 1991].

3.1 Direct Manipulation of Database Queries

Cook and Swayne [2007] use SQL queries to formalize the direct manipulation methods discussed in Ahlberg et al. [1991] and Buja et al. [1991]. As it turns out, we can embed this framework inside the layered grammar of graphics with two classes of new aesthetics: one class to define a selection source and one to define a target. This is most easily seen using our animint implementation, which has a `clickSelects` aesthetic for defining the selection source (and modifying the selection via mouse click) and a `showSelected` aesthetic for defining the target. Here 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, as shown in the video in Figure 3.1. As a result, we can quickly see how the relationship among tip amount and total bill amount depends on whether the customer is smoker.

```
library(animint)
p1 <- ggplot() + geom_bar(
  data = reshape2::tips,
  aes(x = smoker, clickSelects = smoker),
)
p2 <- ggplot() + geom_point(
  data = reshape2::tips,
  aes(x = total_bill, y = tip,
      showSelected = smoker)
)
animint2dir(list(p1 = p1, p2 = p2))
```

In essense, the R code above allows us to use direct manipulation to dynamically perform SQL queries of the form:

```
SELECT total_bill, tip FROM tips
    WHERE smoker IN clickSelects
```

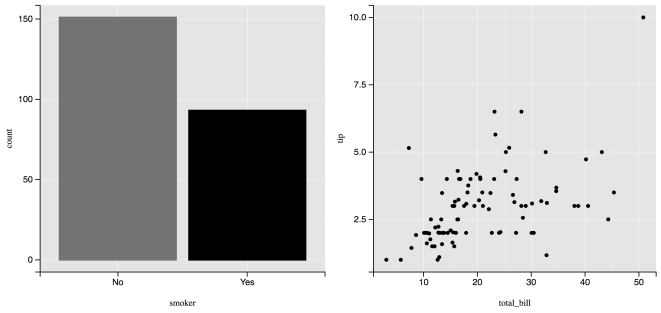

Fig. 1. Video demonstrating linked database querying via direct manipulation using animint. You can view this movie by opening the pdf document with Adobe Reader and clicking on the figure above, or view it online at `https://vimeo.com/160496419`

In this example, `clickSelects` is either "Yes" or "No", but as we show in later examples, `clickSelects` can also be an array of values. Although `clickSelects` is tied to a mouseclick event, this same framework supports other selection events, such as hover or click+drag. Statistically speaking, this is useful for visualizing and navigating through joint distributions conditional upon discrete values. In this sense, our extension is closely related to the same a basis which leads to trellis displays [Becker et al., 2010] and linked scatterplot brushing [Becker and Cleveland, 1987]. The major difference are that conditioning: is layer (i.e., not plot) specific, is not tied to a particular geometry, and can be controlled through direct manipulation or animation controls.

3.2 Adding animation

In some sense, the `showSelected` aesthetic splits the layer into subsets – one for every unique value of the `showSelected` variable. The `clickSelects` aesthetics provides a mechanism to alter the visibility of those subset(s) via direct manipulation, but our animint implementation can also loop through selections to produce animation(s). We acheive this by reserving the name `time` to specify which variable to select as well as the amount of time to wait before changing the selection (in milliseconds). We also reserve the name `duration` to specify the amount of time used to smoothly transition between frames (with linear easing). The code below was used to generate Figure 3.2 which demonstrates a simple animation with smooth transitions between 10 frames of a single point. Note that the resulting web page has controls for interactively altering the `time` and `duration` parameters.

```
d <- data.frame(v = 1:10)
plotList <- list(
  plot = ggplot() + geom_point(
    data = d, aes(x=v, y=v, showSelected=v)
  ),
  time = list(variable = "x", ms = 1000),
  duration = list(x = 1000)
)
animint2dir(plotList)
```

3.3 World Bank Example

Figure 3 shows an interactive animation of the World Bank data set [World Bank, 2012] created with our animint implementation. 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.

[1] `http://bl.ocks.org/tdhock/raw/8ce47eebb3039263878f/`

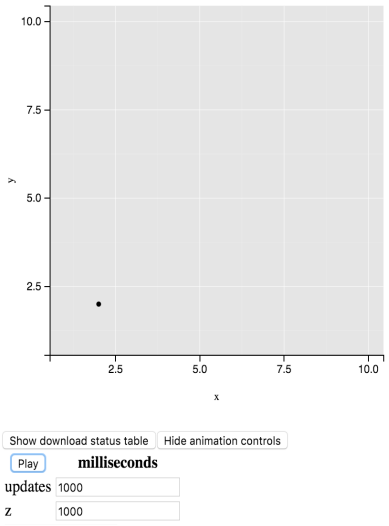

Fig. 2. Video demonstrating a simple animation with smooth transitions and interactively altering transition durations. You can view this movie by opening the pdf document with Adobe Reader and clicking on the figure above, or view it online at `https://vimeo.com/160505146`

We anticipate that some ggplot2 users will be able to reverse engineer the animint code which creates Figure 3, simply by looking at it. In fact, this is one of the wonderful things about ggplot2, it helps minimize the amount of time required to translate a figure in your head into code (TODO: more on this in section X). In the left hand plot, we have a time series of the life expectancy where each line (`group`) is a country and lines are `color`ed by region. By clicking on a line, we 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 clue for the selected year in the time series, so we also layer some "tall rectangles" onto the time series. 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`. 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 3, there are three layers: a point layer for countries, a text layer for countries, and a text layer

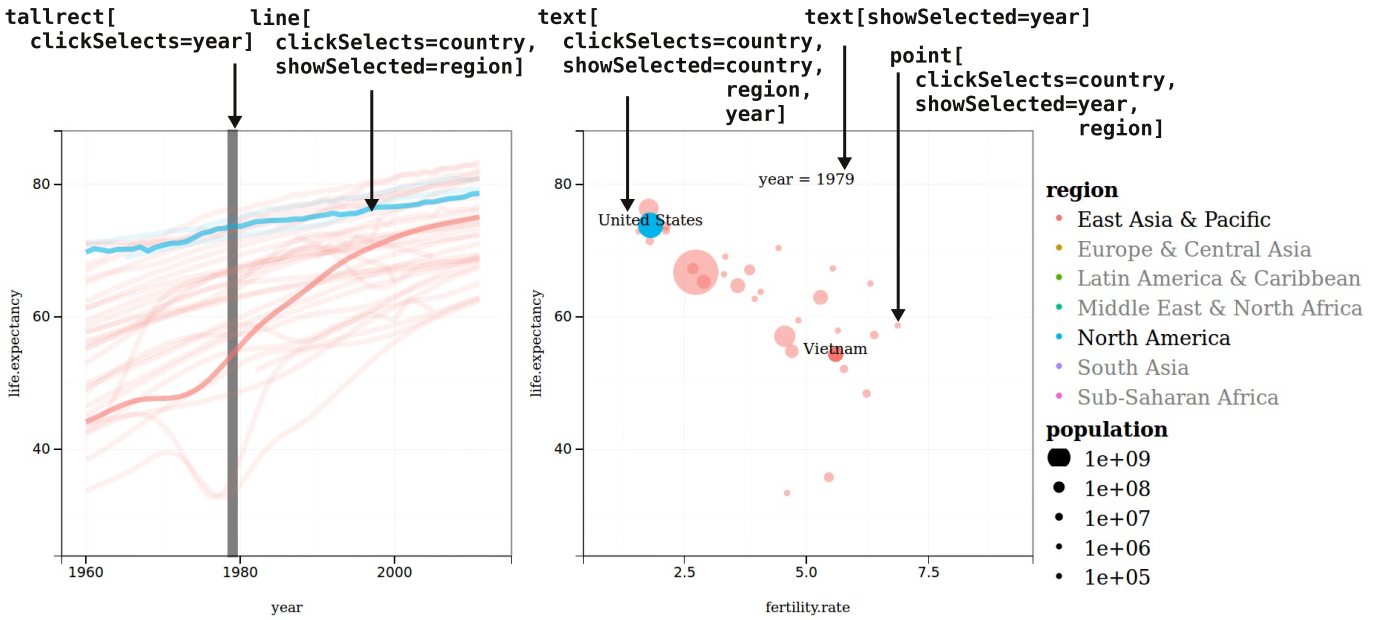

Fig. 3. An interactive animation of World Bank demographic data of several countries, designed using `clickSelects` and `showSelected` keywords (top). **Left**: a multiple time series from 1960 to 2010 of life expectancy, with bold lines showing the selected countries and a vertical grey tallrect showing 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}

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 right 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)
)
```

The text layer for annotating selected countries looks quite similar to the point layer with a few important differences:

```
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)
)
```

Now that we have defined the plots in Figure 3, we can set the `time` and `duration` options (introduced in Section 3.2) to control the animation parameters. Our animint implementation also respects a `selector.types` option which controls whether or not selections for a given variable can accumulate.[2] By default, supplying the list of plots and additional options to `animint2dir()` will write all the files necessary to render the visualization to a temporary directory and prompt the browser to open the relevant HTML file.

```
viz <- list(
  timeSeries = timeSeries,
  scatterPlot = scatterPlot,
  time = list(variable = "year", ms = 3000),
  duration = list(year = 1000),
  selector.types = list(
    year = "single",
    country = "multiple",
    region = "multiple"
  )
)
animint2dir(viz)
```

Direct manipulation is useful when it is easy to identify and select a graphical marks on a plot, but that is not always easy. In these cases, it is convenient to have a dropdown menu to aid the selection process. For this reason, the animint system automatically provides selection menu(s) for each `showSelected` variable. For an example, suppose that we would like select a country by name in Figure 3. Figure 4 shows what the user sees after typing "th" in the search box.

3.4 Implementation details

As shown in Figure 5, the animint system is implemented in 2 parts: the compiler and the renderer. The compiler is implemented in about 2000 lines of 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 (Figure 5).

[2]We maintain a complete list of options here – `https://github.com/tdhock/animint/wiki/Advanced-features-present-animint-but-not-in-ggplot2`

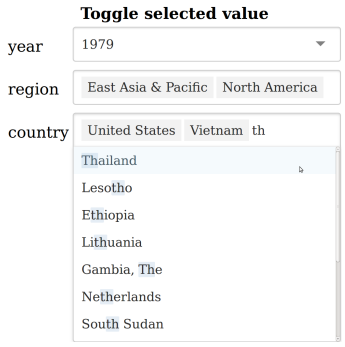

Fig. 4. Animint provides a menu to update each selection variable. In this example, after typing "th" the country menu shows the subset of matching countries.

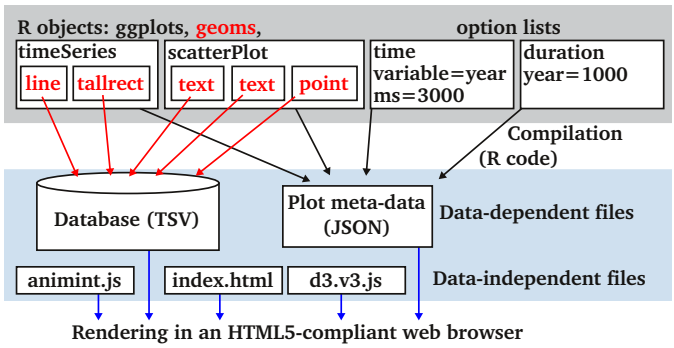

Fig. 5. Schematic explanation of compilation and rendering the World Bank visualization shown in Figure 1. **Top**: the interactive animation is a list of 4 R objects: 2 ggplots and 2 option lists. **Center**: animint R code compiles data in ggplot geoms to a database of TSV files (⇾). It also compiles plot meta-data including ggplot aesthetics, animation time options, and transition duration options to a JSON meta-data file (⇾). **Bottom**: those data-dependent compiled files are combined with data-independent JavaScript and HTML files which render the interactive animation in a web browser (⇾).

Table 1. Implementation complexity and features of the World Bank data visualization using several libraries that can create interactive animations. For each library we show the number of lines of code (LOC), the on-screen objects that can be clicked, the number of interaction variables, and URL of the interactive version.

library	LOC	click on	interaction vars
animint	20	plotted data	several
animation	38	play/pause	1 = time
ggvis/shiny	84	widgets	several
Tableau		widgets/plot	several

The compiler scans the aesthetics in the ggplots to determine how many selection variables are present, and which geoms to update after a selection variable is updated. It uses ggplot2 to automatically calculate the axes scales, legends, labels, backgrounds, and borders. It outputs this information to the JSON plot meta-data file.

The compiler also uses ggplot2 to convert data variables (e.g. life expectancy and region) to visual properties (e.g. y position and color). The data for each layer/geom are saved in several TSV files, one for each combination showSelected values. Thus for large data sets, the web browser only needs to download the subset of data required to render the current selection [Liu et al., 2013].

When repeated data would be saved in each of the TSV files, an extra common TSV file is created so that the repeated data only need to be stored and downloaded once. In that case, the other TSV files do not store the common data, but are merged with the common data after downloading. This method for constructing the TSV file database was developed to minimize the disk usage of animint, particularly for ggplots of spatial maps as in Figure 6.

Finally, the rendering engine (`index.html`, `d3.v3.js`, and `animint.js` files) is copied to the plot directory. The `animint.js` renderer is implemented in about 2200 lines of JavaScript/D3 code that renders the TSV and JSON data files as SVG in a web browser. Importantly, animation is achieved by using the JavaScript `setInterval` function, which updates the `time` selection variable every few seconds. Since the compiled plot is just a directory of files, the interactive plots can be hosted on any web server. The interactive plots can be viewed by opening the `index.html` page in any modern web browser.

4 Comparison Study

To compare our animint implementation with similar leading systems, we reimplemented Figure 3 using Tableau as well as the R packages animation and ggvis. We also walk through examples of the grand tour with both animint and ggvis which helps point out important consequences of each design approach.

4.1 World Bank Example

As shown in Table 1), animint requires significantly fewer lines of code and produces interactive plots with more articulatory directness [Hutchins et al., 1985]. Note that it is possible to implement the World Bank visualization in pure D3, but would require significantly more code.

We designed a version of the WorldBank visualization with limited interactivity, using 38 lines of R code and the animation package. The main idea behind this approach is to use an imperative programming style with for loops to create a static PNG image for each year of the data, and then show these images in sequence. The main drawback to this approach is that the resulting plot is only interactive with respect to the year variable. In other words, the designer must select some countries to emphasize, and the user can not change that selection. Another drawback is that R package animation does not support smooth transitions between animation frames. In contrast, using only 20 lines of the animint DSL, the animint package achieves smooth transitions and interaction with respect to both year and country variables.

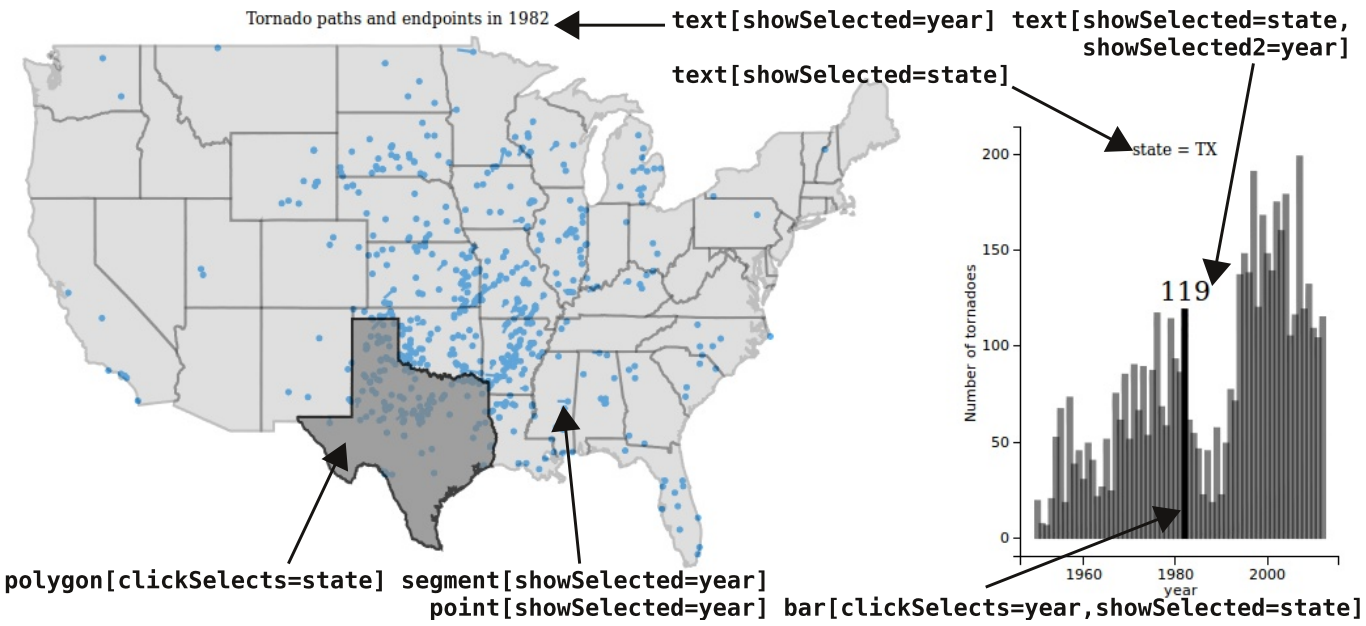

Fig. 6. Interactive animation of tornadoes recorded from 1950 to 2012 in the United States. **Left**: map of the lower 48 United States with tornado paths in 1982. The text shows the selected year, and clicking the map changes the selected state, currently Texas. **Right**: time series of tornado counts in Texas. Clicking a bar changes the selected year, and the text shows selected state and the number of tornadoes recorded there in that year (119 tornadoes in Texas in 1982).

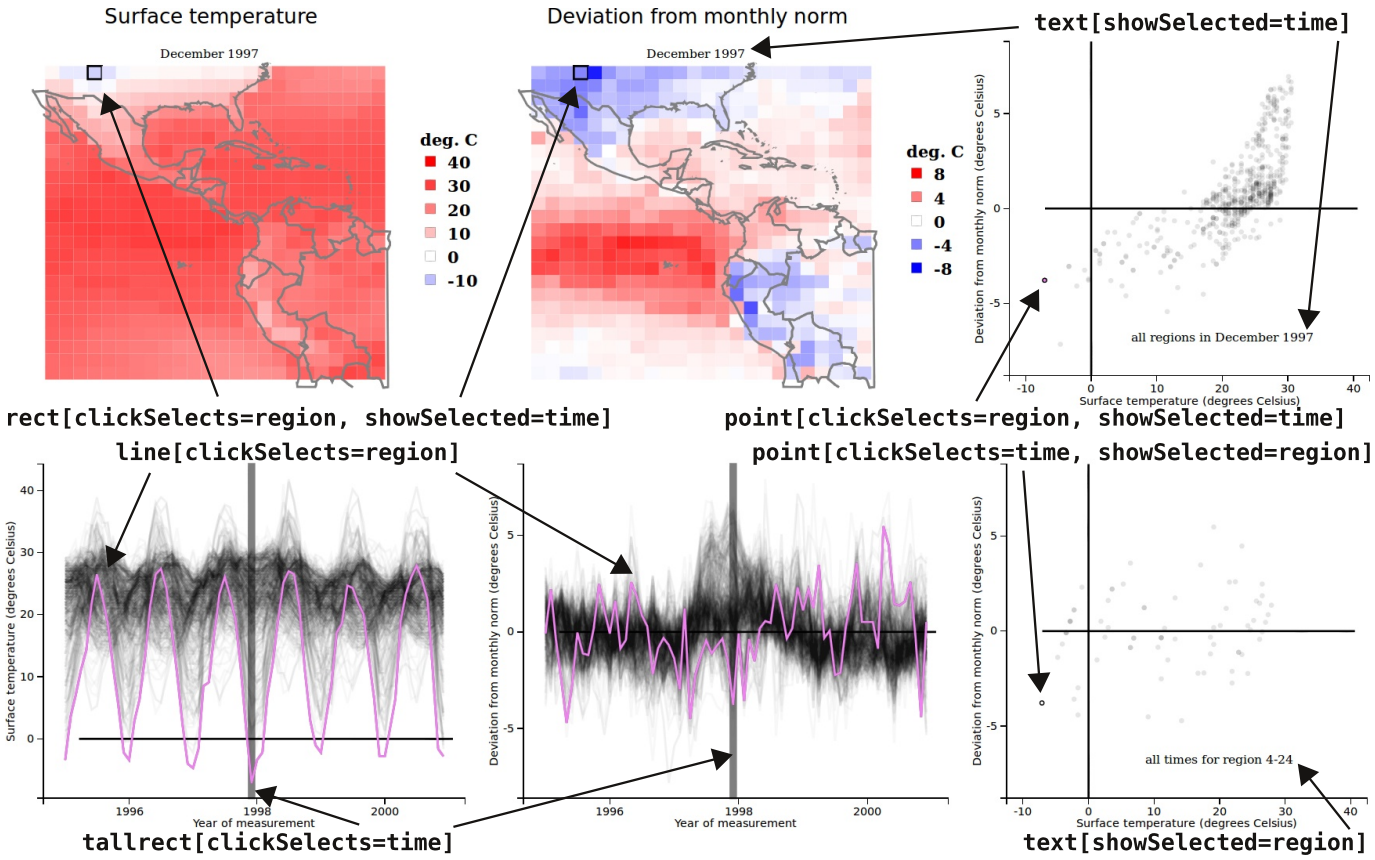

Fig. 7. Visualization containing 6 linked, interactive, animated plots of Central American climate data. **Top**: for the selected time (December 1997), maps displaying the spatial distribution of two temperature variables, and a scatterplot of these two variables. The selected region is displayed with 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.

We designed another version of the World Bank data visualization in 115 lines of R code (`http://bit.ly/1SsJKlN`), using the ggvis graphics library combined with the recommended shiny web server package [RStudio, 2013, Chang and Wickham, 2015]. Showing and hiding data subsets was accomplished by clicking on a slider for year and a menu for country, not by clicking on the plot elements. In contrast, we designed Figure 3 using only 20 lines of R code with the animint package.

The two packages also have different features for interacting with a data visualization: ggvis uses sliders, checkboxes, and other HTML form elements, whereas animint users can directly click the SVG elements that are used to visualize the data. For example, a ggvis of the WorldBank data would animate over the years by adding a play/pause button to a slider widget which controls the selected year. In contrast, in animint we used a multiple time series plot where the year can be selected by directly clicking the data values on the plot (Figure 1). The animint plot is thus easier for the user since it has more articulatory directness [Hutchins et al., 1985], and less spatial offset [Beaudouin-Lafon, 2000].

Another difference is the amount of work required to deploy or share a visualization. A compiled animint visualization consists of static TSV, JSON, HTML, and JavaScript files which can be easily served with any web server. In contrast, ggvis+shiny requires a web server with R and special software installed, significantly complicating deployment to the web.

There are also inherent speed tradeoffs to using a client-server plotting system like ggvis+shiny rather than an entirely web client/JavaScript-based system like animint. There is one main difference between these two types of systems that affects responsiveness of a web-based interactive plotting system: client-server communication overhead. All the animint JavaScript plot rendering code is executed in the web browser, whereas ggvis executes some computations on the server. This means that after a mouse click, ggvis can not update a plot immediately, but instead must wait for the server to respond with the plot data.

We quantified speed differences between the two systems by timing web page loading using DevTools in the Chromium web browser Version 33.0.1750.152, on Ubuntu 12.04 (256984). We also used `getTime()` in JavaScript to record timings for interactive plot updates (on a desktop computer with a 2.8GHz Intel Core i7 CPU). Using ggvis with a local web server and the World Bank data resulted in a web page that loaded quickly (about 1.4s), but updated the plot with a noticeable lag after each mouse click (500–1000ms). Note that since we used a local web server, these times represent the overhead of the web server system, and would be larger with a remote web server.

When we used animint to make the World Bank data visualization, the compilation from R objects to 2.1MB of uncompressed TSV data files took 2.3s. Using a local web server, the animint JavaScript rendered the plot very quickly (100–200ms). We also observed very fast plot updates after mouse clicks in animint: 20–30ms response times for selecting the year, and 60–70ms response times for selecting the country.

The conclusion of our speed comparison is that the overhead of waiting for a web server to perform computations results in significant slowdowns for interactive animations. It is clear that for quick response times, it is preferable to use an entirely JavaScript-based system like animint.

In contrast, a web server system like ggvis+shiny would be more appropriate for performing arbitrary calculations in R/C code on the server, in response to user inputs, and then sending the result across the network for plotting in the user's web browser. This power is not always necessary for interactive animations, since the only operation needed is showing precomputed data subsets. However, the web server system would certainly be preferable when there are many more data subsets than could ever be precomputed. In that case, the web server would only compute the subsets that the user interactively specifies.

We implemented a version of the WorldBank data visualization using Tableau's GUI (`http://bit.ly/worldBank-tableau`). It was impossible to implement all features of the multiple time series plot of the data visualization, since it includes multiple layers with different data sources and variable mappings (a line for each country and a tallrect for each year). Since each mark in a Tableau plot must use the same data source, it was impossible to control the clickable multiple time series and the clickable tallrects in different ways based on the two different selection variables.

Tableau's GUI and visual query approach make it easy to design several kinds of linked plots, but it does not easily acheive the same level of flexibility that our layered grammar of graphics extension provides. More specifically, it allows for each layer of geoms in a plot to have a different data source and variable mapping, while Tableau requires each mark of a plot to be a function of a single query result. This results in a substantial conceptual difference between the selection models of Tableau and animint. In Tableau, there is a selection set for each plot, which may include several different marks. In contrast, animint keeps track of a selection set for each selection variable, each of which has geom-specific effects based on the geom's `clickSelects`/`showSelected` variables.

4.2 The Grand Tour

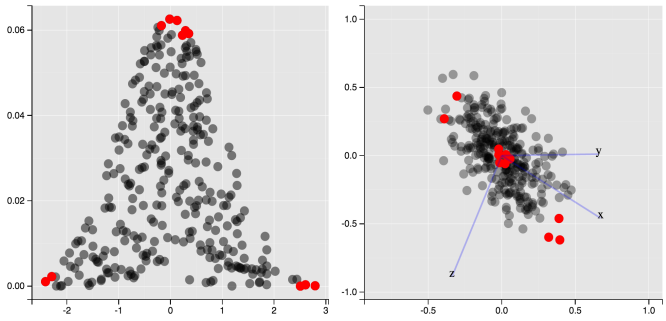

Fig. 8. Video demonstrating a grand tour with linked selection. You can view this movie by opening the pdf document with Adobe Reader and clicking on the figure above, or view it online at `https://vimeo.com/160720834`

In this section we discuss the range of examples that we have designed with animint. Table 2 shows several characteristics of 11 interactive visualizations that we have designed using animint.

We quantified the implementation difficulty of the animint examples using lines of R code, including data processing but not including comments (80 characters max per line). We counted the number of plots and variables shown to quantify the amount of information conveyed by the visualization. Using only 17 lines of code, we designed a simple visualization that shows 2 linked plots of 4 variables in the worldPop data set. In contrast, the most complex visualization required 229 lines of code, and it shows 44 variables across 5 linked plots (Figure **??**). All of the visualizations that we designed involved at least 2 interaction variables (e.g. year and country in Figure 1) and 2 plots. Indeed, animint is most appropriate for interactive visualizations of multivariate data that are not easy to view all at once in one plot.

Table 2 also shows animint system requirements for plots of various sizes. We timed the compilation step in R code ("seconds" column), and measured the size in megabytes of the compiled TSV file database ("MB" column), and found that both increase with the data set size ("rows" column). We also noticed that the time required for the interactive updates and rendering increases with the amount of data displayed at once ("onscreen" column). In particular, 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://bit.ly/O0VTS4`), but since it shows less data onscreen

	LOC	seconds	MB	rows	onscreen	variables	interactive	plots	animated?	Fig
worldPop	17	0.2	0.1	924	624	4	2	2	yes	
WorldBank	20	2.3	2.1	34132	11611	6	2	2	yes	1
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	39	1.7	6.1	103691	16642	11	2	2	no	3
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	84	12.8	19.7	253856	88980	15	2	6	yes	4
scaffolds	110	56.3	78.5	618740	9051	30	3	3	no	
ChIPseq	229	29.9	78.3	1292464	1156	44	4	5	no	5

Table 2. Characteristics of 11 interactive visualizations designed with animint. From left to right, we show the data set name, 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 (variables), the number of `clickSelects`/`showSelected` variables (interactive), the number of linked panels (plots), if the plot is animated, and the corresponding Figure number in this paper (Fig).

(about 2703 elements), it exhibits faster responses to interactivity and animation.

4.3 Animated examples

Figure 6 shows an interactive animation of tornadoes observed in the United States between 1950 and 2012. At any moment in time, the user can simultaneously view the spatial distribution of tornadoes in the selected year over all states, and see the trend over all years for the selected state. Clicking a state on the map updates the time series bars to show the tornado counts from that state. Clicking a bar on the time series updates the selected year.

Figure 7 shows an interactive animation of climate time series data observed in Central America. Two maps display the spatial distribution of two temperature variables, which are shown over time in corresponding the time series plots below. Scatterplots also show the relationships between the two temperature variables for the selected time and region. Clicking any of the plots updates all 6 of them. The `clickSelects` and `showSelected` aesthetics make it easy to design this set of 6 linked plots in only 87 lines of code.

4.4 Non-animated examples

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://bit.ly/1gGYFIV`). The user can explore different model parameters and data sets since these are encoded as animint interaction variables.

5 User feedback and observations

By working with researchers in several fields of research, we have created a wide variety of interactive visualizations using animint. Typically, the researchers have a complex data set that they wish to visualize, but they do not have the expertise or time to create an interactive data visualization. The animint DSL made it easy to collaborate with the various domain experts, who were able to provide us with annotated sketches of the desired plots, which we then translated to animint R code. In this section we share comments and constructive criticism that we have obtained from our users.

5.1 User perspective

For the `prior` data visualization (`http://bit.ly/1peIT7t`), the animint user is a machine learning researcher who developed an algorithm and applied it to 4 benchmark data sets. He wanted to explore how his algorithm performed, in comparison to a baseline learning algorithm. He appreciated the intuition about his algorithm's performance that he learned from the interactive plots: "Interactive plotting allows us to explore all relationships of our high-dimensional dataset and gives us an intuitive understanding of the performance of our proposed algorithm. An intuitive understanding of the results is important since it shows under which conditions our proposed method works well and provides avenues for further research."

Another user from a machine learning background found the interactive plots useful for presenting his work: "the 'regularization path' is a difficult concept to demonstrate in my research. The animint (`http://bit.ly/1gVb8To`) helped greatly by rendering an interactive plot of regularization path, likelihood, and graph at the same time and illustrating their connections. It also reveals an interesting phenomenon that maximizing the testing likelihood actually gives many false positives."

In another application, the animint user was a genomics researcher: "viewing and exploring my complex intestinal microbiome dataset in animint allowed me to grasp the patterns and relationships between samples at an almost intuitive level. The interactive aspect of it was very helpful for browsing through the dataset."

Finally, users also appreciated the simple web interface, and the detail that is possible to show in interactive plots, but impossible to show in publications: "... the web interface is simple and easy to use. It also enables us to publish more detailed interactive results on our website to accompany the results presented in publications."

5.2 Developer perspective

R users, and in particular ggplot2 users, have found that animint is easy to learn and use. One statistics Ph.D. student writes, "animint is a fantastic framework for creating interactive graphics for someone familiar with R and ggplot2's grammar of graphics implementation. The API is very intuitive and allows one to quickly bring their static graphics to life in a way that facilitates exploratory data analysis."

6 Discussion

Finally, another key strength of ggplot2 and D3 for visualization design are the libraries' declarative syntax, which allows a visualization designer to specify *what* they want to render rather than *how* to render it. Heer and Bostock [2010] proposed a declarative syntax for animated transanitions, and studied the benefits of declarative languages for data visualization. animint is another declarative DSL, but defined at a higher level of abstraction than D3. It enables designers to focus on data visualization, while the animint library developers can work on improving the lower-level rendering details.

7 Limitations and future work

There are several limitations to the animint system, which suggest avenues for future work. Some limitations are specific to the current implementation as research software, and some limitations are inherent in the `clickSelects`/`showSelected` keywords.

7.1 Limitations of current implementation

animint implements several linked plots, one of the hallmarks of interactive visual analysis [Konyha et al., 2012]. However, one limitation to the current implementation is that a selection is defined as a set of distinct elements (e.g. year=$\{1991, 1992\}$) rather than a logical expression (e.g. year > 1990). Also, animint does not yet implement a rectangular brush for specifying values of multiple selection variables. Importantly, these are drawbacks of the current implementation, not the animint DSL.

A number of limitations derive from the fact that some plot elements are computed once during the compilation step and remain static on a rendered plot. For example, users are unable to change variable mappings since these are specified by the designer at compile time. Also, when different data subsets have very different ranges of values, it may be preferable to recompute scales when `clickSelects` selection(s) change. A workaround is shown in Figure **??**, which omits the x axis on the bottom plot, since in fact the x values are all normalized to [0,1]. A future implementation of animint would benefit from changes to the compiler and renderer that allow scales to be updated after each click.

Some animint limitations can be resolved by animint designers who are familiar with the shiny web server R package [RStudio, 2013]. animint provides "shiny bindings" which enables a designer to embed an animint plot within a shiny app without writing any HTML or JavaScript, which allows a user to re-compile an animint from a web browser. For example, we implemented a shiny app in which users can redefine variable mappings (`http://bit.ly/animint-shiny`).

As discussed in Section 3.3 and illustrated in Figure 5, the compiler is written in R, and the renderer is written in JavaScript. animint designers define interactive animations using only R code, and no knowledge of JavaScript is necessary. This is convenient for useRs from a statistical background, but presents a barrier for web developers who are more familiar with JavaScript than R. For these web developers, it would be advantageous in the future to implement a compiler and renderer in pure JavaScript, by possibly building `clickSelects` and `showSelected` extensions into Vega [Trifacta, 2014].

The current animint implementation is limited to two specific types of interactivity: highlighting the selected `clickSelects` element, and showing/hiding `showSelected` elements. In the future, we could implement several other types of interactivity without changing the animint DSL. Examples include zooming, panning, and plot resizing. However, some kinds of interactivity would require extensions to the animint grammar. For example, a `hoverSelects` aesthetic could be used to change the selection when hovering over a data point.

7.2 Limitations of `clickSelects`/`showSelected` keywords

TODO. discuss that grand tours are awkward in animint since each animation frame must be pre-computed, and there are many more possible. It's not that they can't be computed beforehand. But if you have a large amount of projections that you want to view, that could be a bottleneck [Wickham et al., 2011]. The grand tour picks random projections, so I'm pretty sure the number of projections is infinite in the mathematical sense. But there are also guided tours that pick "interesting" projections.

TODO: discuss two main failure modes: 1. you really want to compute something on the fly (like a random projection) and 2. with multiple selection variables, there are too many items in the power set so they can't all be computed in advance.

TODO: discuss conditioning on quantitative variables? any concrete example plots where this would be useful?

TODO: distinction between the `clickSelects`/`showSelected` keywords and the animint system which pre-computes everything? Could there be a `clickSelects`/`showSelected` system which does NOT pre-compute everything?

Since animint does not perform any computations other than showing and hiding data subsets, there is a limitation to what can be displayed with multiple selection variables. The limitation is that it is not feasible to precompute something to display for each of the combinatorial number of possible selections of a multiple selection variable. For instance, in the WorldBank visualization of Figure 1, it would not be feasible to display a single smoothing line computed from all the selected countries. This is because `showSelected=country` means to show one thing for each selected country (not one thing computed based on the set of selected countries). Supporting this kind of interaction would require substantial modifications to the animint system, including adding the ability to perform computations on multiple selection variable sets.

TODO: revise paragraph. animint's performance can be measured using speed, memory, and disk space requirements in the compilation and rendering steps. Although we showed in Section 4 that animint provides smoother interactivity than client-server systems, future versions of animint could be made even more efficient and responsive. For example, of the plots in Table 2, the longest compilation step took 56.3 seconds, which may be reduced by optimizing the R code compiler.

This highlights one of the main motivations for using a declarative DSL like animint: none of the designer's R code needs to be changed to implement improvements like this. Instead, the animint developers just need to work on a better compiler and rendering engine. Indeed, Heer and Bostock [2010] noted that this is one of the main benefits of declarative language design: "By decoupling specification from implementation, developers can implement language optimizations without interfering with the work of designers."

While several optimizations remain to be implemented, the current animint library already provides an efficient syntax for the design of interactive, animated data visualizations.

In the future, I'd be interesting in trying to "solve" (1) for some class of problems where you want some elements to have smooth transitions when new data arrives.

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

C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *ACM CHI '92 Conference Proceedings*, volume 21, pages 619–626, 1991.

C. H. Andreas Buja, Daniel Asimov and J. A. McDonald. Elements of a viewing pipeline for data analysis. In W. S. Cleveland and M. E. McGill, editors, *Dynamic Graphics for Statistics*. Wadsworth, Inc., Belmont, California, 1988.

M. Beaudouin-Lafon. Instrumental interaction: An interaction model for designing post-wimp user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '00, pages 446–453, New York, NY, USA, 2000. ACM. ISBN 1-58113-216-6. doi: 10.1145/332040.332473. URL `http://doi.acm.org/10.1145/332040.332473`.

R. Becker and W. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, May 1987.

R. A. Becker, W. S. Cleveland, and M.-J. Shyu. The visual design and control of trellis displays. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.

M. Bostock, V. Oglevetsky, and J. Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17 (12):2301–2309, December 2011.

A. Buja, J. A. McDonald, J. Michalak, and W. Stuetzle. Interactive data visualization using focusing and linking. *IEEE Proceedings of Visualization*, pages 1–8, Feb. 1991.

Cairo. Cairo: A vector graphics library, 2016. URL http://cairographics.org/.

W. Chang and H. Wickham. *ggvis: Interactive Grammar of Graphics*, 2015. URL https://CRAN.R-project.org/package=ggvis. R package version 0.4.2.

D. Cook and D. F. Swayne. *Interactive and dynamic graphics for data analysis : with R and GGobi*. Use R ! Springer, New York, 2007. ISBN 978-0-387-71761-6. URL http://www.ggobi.org/book/.

D. Donoho. 50 years of Data Science. Oct. 2015. URL https://dl.dropboxusercontent.com/u/23421017/50YearsDataScience.pdf.

R. Gentleman and D. T. Lang. Statistical Analyses and Reproducible Research . *Bioconductor Project Working Papers*, pages 1–38, Nov. 2004.

J. Heer and M. Bostock. Declarative language design for interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1149–1156, 2010.

E. L. Hutchins, J. D. Hollan, and D. A. Norman. Direct manipulation interfaces. *Hum.-Comput. Interact.*, 1(4):311–338, Dec. 1985. ISSN 0737-0024. doi: 10.1207/s15327051hci0104_2. URL http://dx.doi.org/10.1207/s15327051hci0104_2.

Z. Konyha, A. Lež, K. Matković, M. Jelović, and H. Hauser. Interactive visual analysis of families of curves using data aggregation and derivation. In *Proceedings of the 12th International Conference on Knowledge Management and Knowledge Technologies*, page 24. ACM, 2012.

M. Lawrence and D. Sarkar. *Interface Between R and Qt*, 2016a. URL https://github.com/ggobi/qtbase. R package version 1.1.1.

M. Lawrence and D. Sarkar. *Qt-Based Painting Infrastructure*, 2016b. URL https://github.com/ggobi/qtpaint. R package version 0.10.0.

M. Lawrence and D. Temple Lang. RGtk2: A graphical user interface toolkit for R. *Journal of Statistical Software*, 37(8):1–52, 2010. URL http://www.jstatsoft.org/v37/i08/.

Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. *Computer Graphics Forum (Proc. EuroVis)*, 32, 2013. URL http://vis.stanford.edu/papers/immens.

P. Murrell and S. Potter. *gridSVG: Export 'grid' Graphics as SVG*, 2015. URL https://CRAN.R-project.org/package=gridSVG. R package version 1.5-0.

D. Nolan and D. T. Lang. Interactive and animated scalable vector graphics and R data displays. *Journal of Statistical Software*, 46 (1):1–88, 2012. URL http://www.jstatsoft.org/v46/i01/.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. URL http://www.R-project.org/.

D. Robinson. *gganimate: Create easy animations with ggplot2*, 2016. URL http://github.com/dgrtwo/gganimate. R package version 0.1.

RStudio. shiny: easy web applications in R, 2013. URL http://www.rstudio.com/shiny/.

Trifacta. Vega: a declarative visualization grammar, Mar 2014. URL http://trifacta.github.io/vega/.

S. Urbanek. *rJava: Low-Level R to Java Interface*, 2016. URL https://CRAN.R-project.org/package=rJava. R package version 0.9-8.

H. Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009. ISBN 978-0-387-98140-6. URL http://had.co.nz/ggplot2/book.

H. Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.

H. Wickham, M. Lawrence, D. Temple Lang, and D. F. Swayne. An introduction to rggobi. *R-news*, 8(2):37, October 2008. URL http://CRAN.R-project.org/doc/Rnews/Rnews_2008-2.pdf.

H. Wickham, M. Lawrence, D. Cook, A. Buja, H. Hofmann, and D. F. Swayne. The Plumbing of Interactive Graphics. *Computational Statistics*, pages 1–7, Apr. 2010.

H. Wickham, D. Cook, H. Hofmann, and A. Buja. tourr: An R Package for Exploring Multivariate Data with Projections . pages 1–18, Apr. 2011.

L. Wilkinson, D. Wills, D. Rope, A. Norton, and R. Dubbs. *The grammar of graphics*. Springer, 2006.

World Bank. World development indicators, 2012. URL http://data.worldbank.org/data-catalog/world-development-indicators.

Y. Xie. animation: An R package for creating animations and demonstrating statistical methods. *Journal of Statistical Software*, 53 (1):1–27, 2013. URL http://www.jstatsoft.org/v53/i01/.

Y. Xie, H. Hofmann, D. Cook, X. Cheng, B. Schloerke, M. Vendettuoli, T. Yin, H. Wickham, and M. Lawrence. *Interactive statistical graphics based on Qt*, 2013. R package version 0.8.5.