Two new keywords for interactive, animated plot design: clickSelects and showSelected

Toby Dylan Hocking
Department of Human Genetics, McGill University
and
Carson Sievert
Department of Statistics, Iowa State University
and
Jun Cai
Center for Earth System Science, Tsinghua University
and
Susan VanderPlas
Department of Statistics, Iowa State University

March 21, 2016

Abstract

To facilitate the design of linked, interactive, animated statistical plots, we propose adding two new keywords to the grammar of graphics: `clickSelects` designates an element that changes the current display when clicked, and `showSelected` designates which elements change. We use several example data sets to discuss the range of visualizations that can be easily specified using these two keywords. Finally, we discuss the current limitations of `animint`, a new R package that implements these keywords: `https://github.com/tdhock/animint`.

1 Introduction

Interactive, animated data visualization is a useful tool for obtaining an intuitive understanding of patterns in multivariate data sets. In this paper, we propose a system for the design of data graphics which can be both interactive and animated. For the purposes of this paper, we define "interactive" to mean that graphical elements such as data points or legend entries may be clicked to update the data that is shown in related plots. This is closely related to the concept of "direct manipulation," which was introduced by Shneiderman [1982] and later applied to statistical data visualization [Hutchins et al., 1985, Becker

and Cleveland, 1987, Cleveland, 1993]. Our definition of an "animated" visualization is one that can be watched like a video, automatically updating over time.

The rest of this paper is organized as follows: we discuss related work in Section 2, and the usage of the `clickSelects` and `showSelected` keywords in Section 3. Then we discuss our current implementation in the animint R package in Section 4.

2 Related Work

In this section we discuss the limitations of several existing systems for interactive data graphic design, focusing on libraries with free/open-source software implementations in JavaScript and R.

2.1 D3 and other JavaScript libraries

Data-Driven Documents (D3) is a JavaScript library that can render interactive, animated data graphics in a web browser [Bostock et al., 2011]. One of the main reasons for the success and popularity of D3 is that it allows visualizations to be specified using the terminology of the Document Object Model (DOM), which makes learning D3 easy for web designers. Interactivity in D3 is specified by writing JavaScript functions that can perform arbitrary DOM manipulations in response to user events.

Dimensional Charting (DC) is another JavaScript library which can produce interactive visualizations consisting of several linked plots [DC.js Team, 2014]. DC allows data graphics to consist of one or more pre-defined chart types, which are linked and rendered in a web browser using D3. In this system it is not easy to create multi-layer graphics (e.g. a map with points for each city).

There are many other libraries which can generate web plots with limited interactivity, but are currently unable to produce interactive animations. For example, the rCharts R package is a wrapper around several JavaScript libraries [Vaidyanathan, 2013], but does not include a method for describing interactions between multiple linked plots.

2.2 Animated graphics in R

The main idea of the animation R package is to iteratively show a sequence of pre-rendered images [Xie, 2013]. In this system the only interaction possible with animation is rewinding and fast-forwarding through the animation frames.

Another R package that can produce animated graphics is googleVis [Gesmann and de Castillo, 2011]. The main limitation of this system is that it can only produce single-layer graphics using a few pre-defined plot types, only one of which can be viewed at any time.

2.3 Libraries based on the grammar of graphics

An implementation of the grammar of graphics [Wilkinson et al., 2006] is provided by the R package ggplot2 [Wickham, 2009], which makes it easy to design multi-layer non-

interactive plots. Another R package that uses the grammar of graphics to define interactive graphics is ggvis [RStudio, 2014]. It does not directly extend ggplot2, but provides a new implementation of some of the same ideas about the grammar of graphics. It relies on the shiny web server package to acheive interactivity [RStudio, 2013].

Implementations of the grammar of graphics system of Wilkinson et al. [2006] exist as the Visualization Markup Language (ViZml) and the Graphics Production Language (GPL) in IBM SPSS Statistics software, but do not support interactive animations.

Vega is a JavaScript visualization library that implements some concepts from the grammar of graphics [Trifacta, 2014]. Vega reads plots defined in a JSON file format, and renders them using D3. It is capable of defining a wide variety of multi-layer plots, but is unable to express all plots that can be made using pure D3. The main limitation of Vega is that it does not directly support interactions and animation. However, the recent work of Satyanarayan et al. [2014] adds some declarative Vega extensions "critical for developing interactive data visualizations."

2.4 Other systems with interactive selection

There are many different methods for interactively specifying a set of selected data points [Willett et al., 2007, Heer et al., 2008]. A common interaction involves a rectangular brush that can select several data points in a single data table [Cleveland, 1993, Becker and Cleveland, 1987], and highlights the selected data points across several plots. This type of interactive selection is implemented in Tableau [2014], which is built on top of VizQL, a visual query language which took influence from Polaris [Stolte and Hanrahan, 2002]. Another implementation is provided by the R packages cranvas and iplots. The main limitation of each of these systems is that the selection operates on only one data set.

3 The `clickSelects` and `showSelected` keywords for interactive plot design

In this section we explain the meaning of the new `clickSelects` and `showSelected` keywords that we propose to use for interactive graphic design. We propose using these keywords as new aesthetics in the ggplot2 system [Wickham, 2009]. An aesthetic is a declarative mapping from a data variable to a property of a geometric plot element. Some standard ggplot2 aesthetics are

x horizontal position,

y vertical position,

color color or fill,

size point or line thickness.

We propose adding two new aesthetics to define interactions:

clickSelects clicking changes the current selection,

showSelected show only data for the current selection.

We use a visualization based on the World Bank data set to explain the usage of these new aesthetics. The World Bank data consist of several economic variables measured for 205 countries from 1960 to 2012 [World Bank, 2012]. Figure 1 shows a visualization consisting of two linked plots based on these data. In the static PDF version of this figure, the year 1979 and the countries United States and Vietnam are selected, but readers are encouraged to interactively select other values 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.

The World Bank data visualization consists of five geometric elements, each with different interactive aesthetics (top of Figure 1). The time series on the left of Figure 1 contains two geometric elements: tallrect and line. We define the tallrects in the background based on the `years` data set, which has one row for every year between 1960 and 2010:

```
yearRects <- geom_tallrect(
  aes(xmin=year-0.5, xmax=year+0.5,
```

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

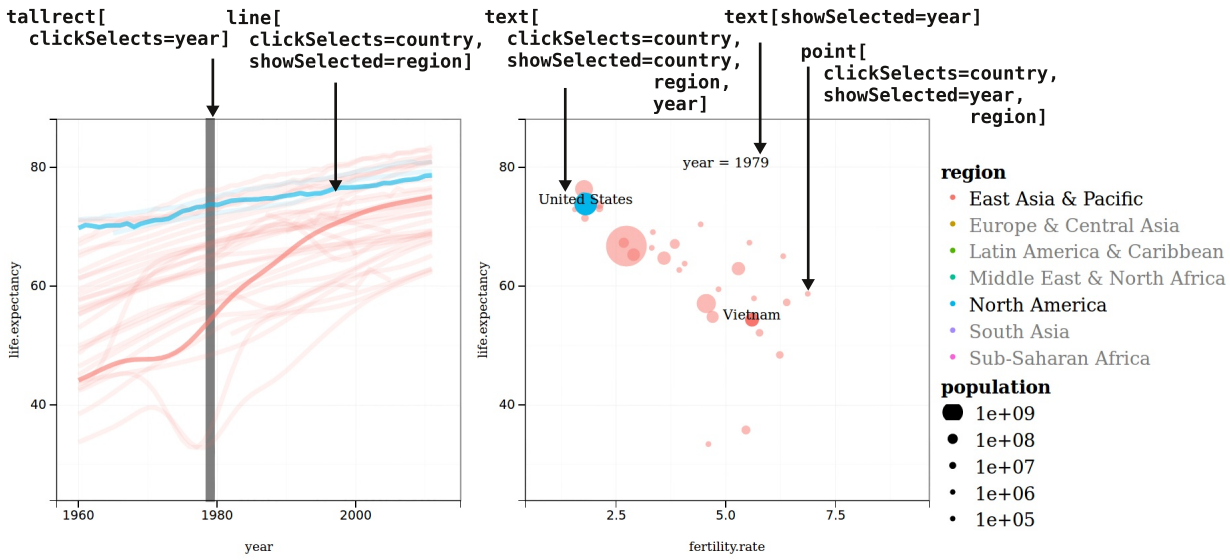

Figure 1: An interactive animation of World Bank demographic data of several countries, designed using `clickSelects` and `showSelected` keywords (top). **Left**: a multiple time series from 1960 to 2010 of life expectancy, with bold lines showing the selected countries and a vertical grey tallrect showing 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}

```
      clickSelects=year),
  data=years)
```

The tallrect plots a rectangle that spans the entire vertical range, for each row of the `years` data set. The horizontal range of each rectangle is specified by the `xmin` and `xmax` aesthetics. The `clickSelects=year` aesthetic means that clicking on a tallrect should change the selected year. For example, if the rectangle for 1980 is clicked, the selected year should change to 1980.

To create the time series plot, we create a ggplot that combines the tallrects above with lines based on the entire World Bank data set:

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

To define the lines we used both interactive aesthetics. The `clickSelects=country` definition means that clicking a line should change the selected country. The `showSelected=region` definition means to only plot the subset of data that corresponds to the current selection of the `region` variable.

Note that the tallrect used `data=years` whereas the line used `data=WorldBank`. The multi-layer ggplot2 system allows a different data set for each layer/geom. Interactivity is accomplished by defining interactive aesthetics using common variable names in the different data sets. For example, the `year` variable is common to both the `WorldBank` and `years` data sets.

Having completed the definition of the time series plot on the left of Figure 1, now let us consider the scatterplot on the right. The following R code defines the points in the scatterplot:

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

The code creates a point for every row in the World Bank data table. The visual characteristics of each point are defined by the values of the corresponding data: the (x,y) position encodes fertility rate and life expectancy, point color encodes the region,

and point size encodes the population. Note that scales are automatically constructed for the x and y axes, and legends are automatically constructed for color and size. The `clickSelects=country` aesthetic means that clicking a point should change the selected country. The `showSelected=year` and `showSelected2=region` aesthetics mean to only plot the points that match both the selected year and region.

In order to remind the plot user which subset of data are selected, we will draw text labels for the selected year and countries. First, we create the year labels using

```
yearText <- geom_text(
  aes(label=sprintf("year = %d", year),
      showSelected=year),
  x=5, y=80,
  data=years)
```

The `label` aesthetic is used to define the text from the year variable, and only the selected year is shown due to the `showSelected=year` aesthetic. Note that since the label `x` and `y` positions are constant, they are not defined as aesthetics.

We can also add another label to show the selected countries:

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

A geom/layer may contain any number of showSelected aesthetics. In this example, the three `showSelected` aesthetics mean to only show the subset of labels corresponding to the selected country, year, and region.

Having defined these three geometric elements, we combine them in a single ggplot:

```
scatterPlot <- ggplot()+
  countryPoints+
  yearText+
  countryText
```

This completes the definition of the scatterplot. Note that both the `timeSeries` and `scatterPlot` objects are valid ggplots, even though they use the new `clickSelects` and `showSelected` aesthetics. However, plotting these objects using ggplot2 will render non-interactive plots with all geometric elements, including data for all years and countries. In the next section, we propose the `animint` R package, which implements a new interactive renderer for ggplots with the `clickSelects` and `showSelected` aesthetics.

4 Current implementation in the animint R package

Since 2013 we have been developing the animint R package,[2] which provides an interactive renderer for ggplots with clickSelects and showSelected aesthetics. We first explain how a graphic designer can use animint to render a set of ggplots, then explain the graphical user interface (GUI) for selecting data subsets. Finally, we discuss some implementation details of the compiler and renderer, which only the animint package developers need to be concerned about.

4.1 Animint renderer for a list of ggplots

We will again use the World Bank data visualization of Figure 1 as an example. Recall that in Section 3 we defined two ggplot objects, scatterPlot and timeSeries. To plot them with animint, we first put them in a named list:

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

The viz list contains two ggplots and three options: time, duration, and selector.types. The time option specifies that in absence of user interaction, we want the plots to animate over time, progressing at a rate of one year every 3 seconds (ms = milliseconds). The duration option specifies that when updating the selected year, there should be a smooth transition with a duration of 1 second. Smooth transitions are useful to emphasize continuity between plotted objects before and after the transition [Heer and Robertson, 2007]. Finally, the selector.types option is used to define either single or multiple selection for each variable. For single selection, only one value is selected at any time (e.g. year=1990). For multiple selection, a set of values is selected (e.g. country={United States, Vietnam}).

The viz list contains all of the information that we need to create the linked plots of Figure 1. To render the plots, we call the animint2dir function:

```
animint2dir(viz, out.dir="figure-WorldBank")
```

The animint2dir function saves some data files and a web page in the figure-WorldBank directory, then it opens the interactive plot in a web browser.

[2] freely available at https://github.com/tdhock/animint

4.2 User interaction

In this section we explain how a user can view and interact with a data visualization produced by the `animint` system. As explained in the previous section, the `animint` system outputs a set of data files along with a web page. Opening the web page in a web browser will render the data visualization and allow the user to select different subsets of data to plot.

When viewing an `animint` data visualization in a web browser, interactions are accomplished mainly via direct manipulation. Beaudouin-Lafon [2000] discussed the advantages of direct manipulation in graphical user interfaces. `Animint` data visualizations are inspired by principle 2, "Physical actions on objects vs. complex syntax." In particular, the user can update the selection by directly clicking on the geoms with `clickSelects` aesthetics. Also, `animint` creates a multiple selection variable for each discrete legend (e.g. region in Figure 1). Clicking on an entry in a discrete legend will update the corresponding selection variable. This use of direct manipulation contrasts other systems which use menus and widgets, and thus suffer from less articulatory directness [Hutchins et al., 1985].

For single selection variables such as year, clicking sets the value of the corresponding selection variable. For multiple selection variables such as country, clicking adds or removes values from the set of selected values.

While it is often best to interact by directly clicking on the plotted data, there are situations where indirect manipulation is preferable. For example, when there are many different values that could be selected, it is convenient to have a menu that contains all possible values. More concretely, suppose that we would like to add Thailand to the selected set of countries in Figure 1. The `animint` renderer provides a selection menu which shows the full list of countries, and allows entering text to search for a particular country (Figure 2).

4.3 Implementation details

As shown in Figure 3, 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 3).

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 4.

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.

5 Results and comparison study on World Bank data

To show the advantages that the `animint` grammar brings for creating interactive and animated data visualizations, we implemented the World Bank visualization of Figure 1 using two other R packages and Tableau. The main result of our comparison (Table 1) is that `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 (Figure **??**), but would require significantly more code.

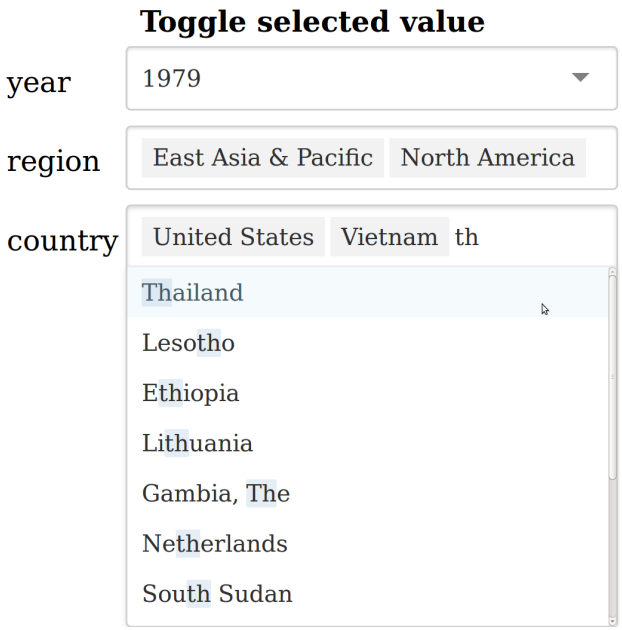

Figure 2: Animint provides a menu to update each selection variable. In this example, after typing "th" the country menu shows the subset of matching countries.

5.1 R package animation

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.

5.2 Client-server systems like ggvis/shiny

We designed another version of the World Bank data visualization in 84 lines of R code (`http://bit.ly/1diUYsg`), using the ggvis graphics library combined with the recommended shiny web server package [RStudio, 2013, 2014]. Showing and hiding data subsets was accomplished by clicking on a slider for year and a menu for country, not by clicking

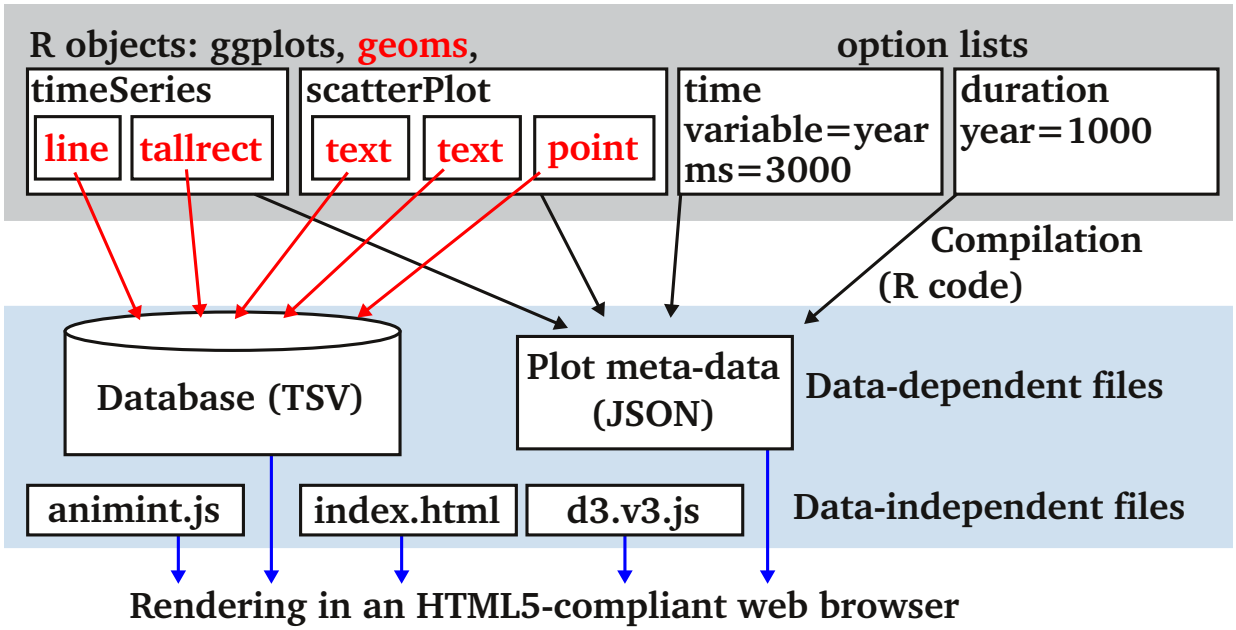

Figure 3: Schematic explanation of compilation and rendering the World Bank visualization shown in Figure 1. **Top**: the interactive animation is a list of 4 R objects: 2 ggplots and 2 option lists. **Center**: `animint` R code compiles data in ggplot geoms to a database of TSV files (⇾). It also compiles 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	http://bit.ly/
`animint`	20	plotted data	several	
`animation`	38	play/pause	1 = time	
`ggvis/shiny`	84	widgets	several	
Tableau		widgets/plot	several	

on the plot elements. In contrast, we designed Figure 1 using only 20 lines of R code with the `animint` package. Implementation is significantly simpler using `animint` because `animint`'s DSL is designed specifically for this type of interactive animation.

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.

	LOC	seconds	MB	rows	onscreen	variables	interactive	plots	animated?	F
worldPop	17	0.2	0.1	924	624	4	2	2	yes	
WorldBank	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	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	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	

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).

5.3 Tableau

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 is limited to a single 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. In conclusion, although Tableau is useful for many simple interactive data visualizations, it is inherently limited in ways that `animint` is not.

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 `animint`/ggplot2's layered grammar of graphics provides. In particular, ggplot2 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.

6 Example applications

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

6.1 Animated examples

Animation is very useful for data sets which have a time variable, as in the World Bank data of Figure 1.

Figure 4 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 5 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.

6.2 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.

Another non-animated example is Figure 6, which was used to explain a complex machine learning model for predicting differences between samples. The interactive visualization allows the user to explore how the predictions change as a function of the model complexity parameter, in several train and test samples. It also uses facets, a feature from ggplot2 that allows multi-panel plots with aligned axes. Finally, it includes hyperlinks which open related web pages in new windows.

Overall, we have found that `animint` is useful for exploring relationships in many different kinds of multivariate data. By using `clickSelects` and `showSelected`, it is easy to design interactive plots that reveal patterns in complex data.

7 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.

7.1 Designer perspective

R users have found that `animint` is easy to learn. 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."

TODO: expand to address reviewer comments.

7.2 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."

8 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.

8.1 Value of sketching

The first step in the design of any data visualization is usually to make a sketch of the desired interactive plot on paper or a whiteboard. In any plot a designer would sketch the axes, legends, a few geometric elements, and note which variables will be shown in the linked plots. An `animint` designer needs only add the `clickSelects` and `showSelected` mappings for each geometric element, as shown in Figures 1, 3, and 4. These notes can be directly translated to ggplot2 aesthetics in R code, as explained below.

9 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.

9.1 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 6, 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 and illustrated in Figure 3, 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.

9.2 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 5 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

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.

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

W. S. Cleveland. *Visualizing Data*. Hobart Press, New Jersey, 1993.

DC.js Team. Dimensional charting JavaScript library, 2014. URL `http://dc-js.github.io/dc.js/`.

M. Gesmann and D. de Castillo. googleVis: Interface between R and the Google Visualisation API. *The R Journal*, 3(2):40–44, December 2011.

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

J. Heer and G. Robertson. Animated transitions in statistical data graphics. *IEEE Transaction on Visualization and Computer Graphics*, 13(6):1240–1247, 2007.

J. Heer, M. Agrawala, and W. Willett. Generalized selection via interactive query relaxation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 959–968. ACM, 2008.

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.

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`.

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

RStudio. ggvis: interactive grammar of graphics for R, Mar 2014. URL `http://ggvis.rstudio.com`.

A. Satyanarayan, K. Wongsuphasawat, and J. Heer. Declarative interaction design for data visualization. In *ACM User Interface Software & Technology (UIST)*, 2014. URL `http://idl.cs.washington.edu/papers/reactive-vega`.

B. Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behaviour & Information Technology*, 1(3):237–256, 1982.

C. Stolte and P. Hanrahan. Polaris: A system for query, analysis and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8:52–65, 2002.

Tableau. VizQL: Visualization Query Language, 2014. URL `http://www.tableausoftware.com/`.

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

R. Vaidyanathan. rCharts: Interactive JS Charts from R, 2013. URL `https://github.com/ramnathv/rCharts`.

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, 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.

W. Willett, J. Heer, and M. Agrawala. Scented widgets: Improving navigation cues with embedded visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1129–1136, Nov. 2007. ISSN 1077-2626. doi: 10.1109/TVCG.2007.70589. URL http://dx.doi.org/10.1109/TVCG.2007.70589.

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/.

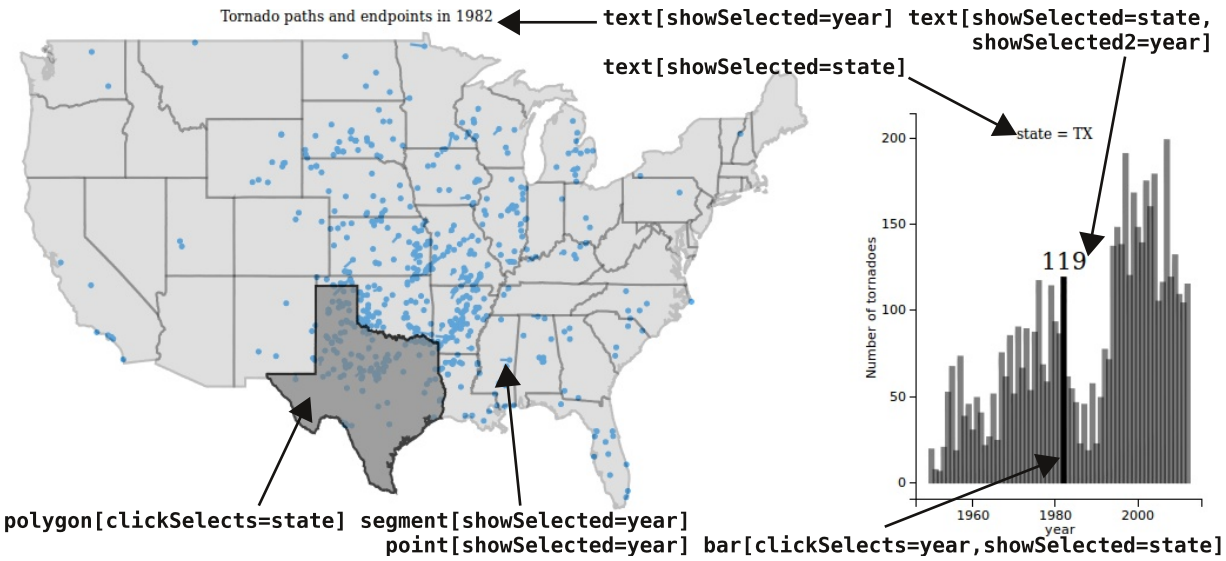

Figure 4: Interactive animation of tornadoes recorded from 1950 to 2012 in the United States. **Left**: map of the lower 48 United States with tornado paths in 1982. The text shows the selected year, and clicking the map changes the selected state, currently Texas. **Right**: time series of tornado counts 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).

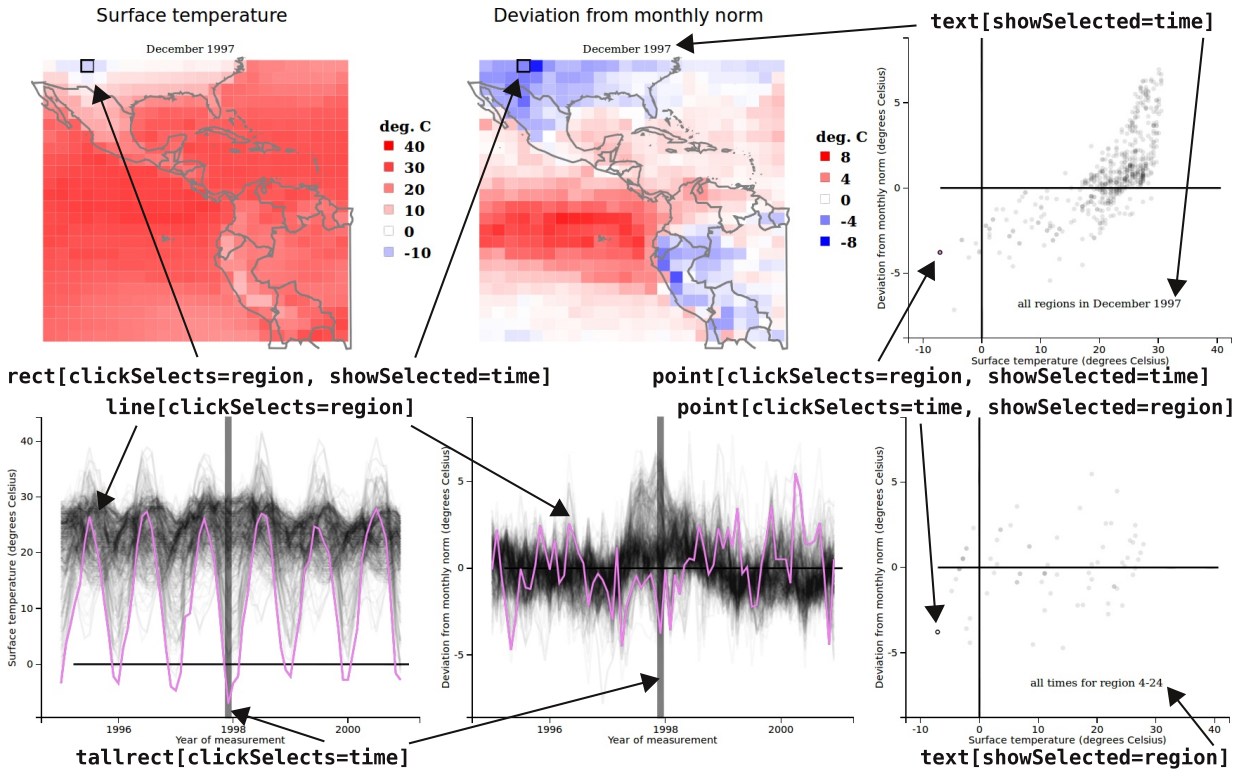

Figure 5: Visualization containing 6 linked, interactive, animated plots of Central American climate data. **Top**: for the selected time (December 1997), maps displaying the spatial distribution of two temperature variables, and a scatterplot of these two variables. The selected region is displayed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

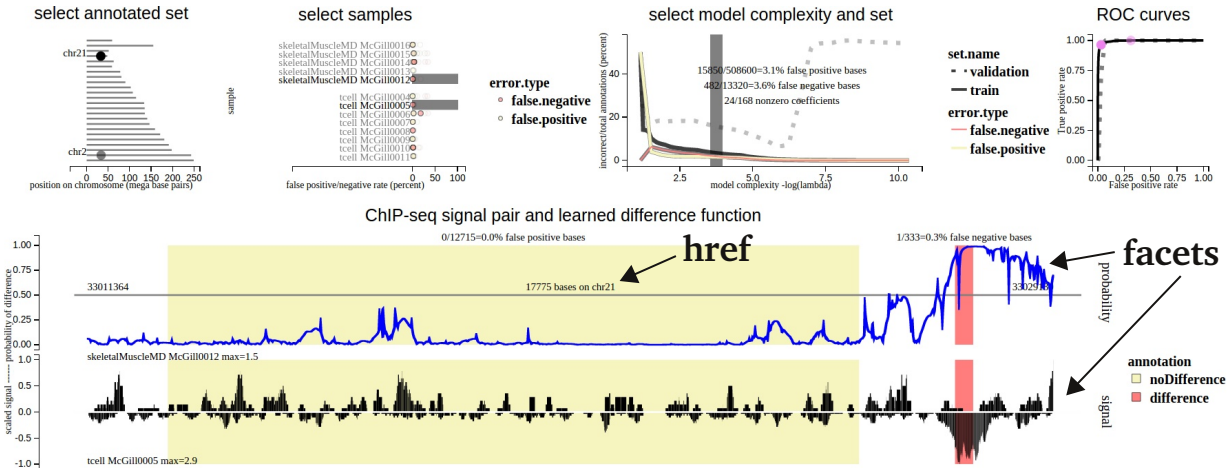

Figure 6: Visualization with 4 selection variables used to navigate through 1,292,464 rows of data in 5 linked interactive plots. The bottom plot shows a facets plot with aligned x axes, used to emphasize that the blue probability function is defined at the same positions as the black signals below. It also contains an href tag (web link), which opens a new genome browser web page zoomed to the same region as the selected data.