Animint: a Grammar for Interactive Animations

Category: System
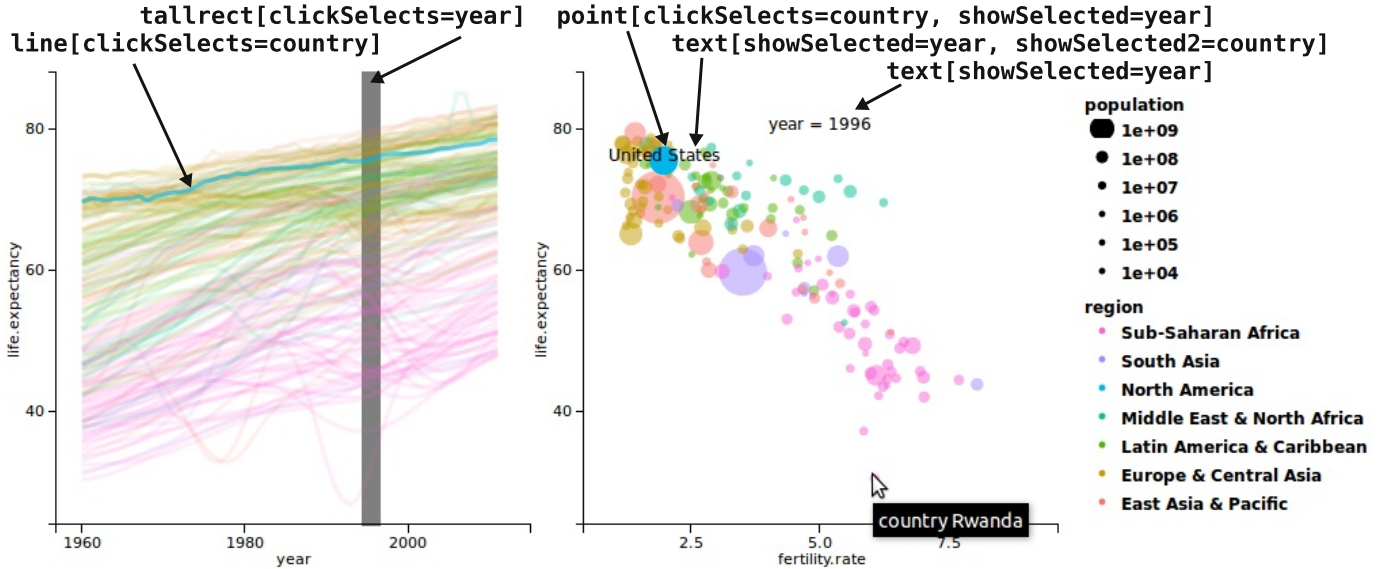

Fig. 1. This interactive animation of World Bank data was designed with only 20 lines of Animint code (`http://bit.ly/1fzfUdN`). **Top**: Five geometric elements span two linked plots: clicking a clickSelects element changes the currently selected year or country, and updates the corresponding showSelected elements. **Left**: a multiple time series from 1960 to 2012 of the life expectancy of 205 countries, with a bold line showing the selected country and a vertical grey tallrect showing the selected year. **Right**: a scatterplot of life expectancy versus fertility rate of all countries. The two text elements show the current selection: "year = 1996" and "United States."

Abstract—Animint is a novel high-level domain-specific language (DSL) for linked, interactive, animated plots. It builds on top of previous work on the grammar of graphics (ggplot2 in R) and interactive visualization using web standards (D3 in JavaScript). In other data visualization systems, interactive animations are difficult to accomplish, since they must be defined in terms of low-level operations and sometimes 100s of lines of code. In contrast, Animint's high-level DSL can be used to produce a wide variety of complex interactive visualizations with only 10s of lines of code. It works by adding 2 new aesthetics to the grammar of graphics: clickSelects, which allows users to select elements of the plot, and showSelected, which displays only elements corresponding to the current selection. After using our novel declarative DSL to define an interactive animation in R code, it is first compiled and then rendered in a web browser using D3. We discuss the design of Animint, then compare to related libraries, and show several example visualizations of high-dimensional time series data.

Index Terms—Information visualization, user interfaces, toolkits, 2D graphics, interactive, animation

---

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 introduce Animint, a new domain-specific language (DSL) for designing linked, interactive, animated data visualizations [Hocking and VanderPlas, 2014]. To explain exactly what kinds of visualizations we are interested in creating, we begin with an example.

In this paper, we are concerned with data sets that can be described as a data table, with columns for each variable and rows for each observation or data point. Consider the World Bank data set, which consist of several economic variables measured for 205 countries from 1960 to 2012 [World Bank, 2012]. The visualization in Figure 1 shows six variables from this data set: year, country, life expectancy, fertility rate, region, and population. We have chosen to emphasize the country United States and the year 1996. The benefit of the interactive visualization is that the reader or user may interact with the plot to view data about another country or time period. Finally, since one of the variables in this visualization is the year of measurement, it is natural to animate the visualization to see how the data changes over time. The purpose of Animint is to make it easy to design such interactive

animations.

In this paper we will discuss three important roles that contribute to the creation of an Animint visualization: the developer, who implements the Animint library; the designer, who uses the Animint library to define a visualization; and the user, who views the visualization in a web browser. The main goal of Animint is to provide an expressive language for designers, while allowing users the freedom to interact with the plot to selectively view data subsets of interest. The designer specifies data sets and maps variables to interactive visual elements using the Animint DSL, then uses the Animint library to compile and save an interactive animation. The user writes no code, but can view and interact with an Animint visualization by clicking Scalable Vector Graphics (SVG) elements in a web browser. A final goal is that the Animint library developer should be the only role concerned with low-level plotting details such as drawing axes and legends, freeing the designer and user to focus on interpreting the data.

Our definition of an "interactive" visualization is synonymous to what Cleveland [1993] called "direct manipulation" — the user directly specifies data of interest by using the mouse on the correspond-

ing plotted elements. Our definition of an "animated" visualization is one that shows new data points over time as the user watches. The goal of the Animint DSL that we introduce in this paper is to make it easy for designers to express visualizations which are both animated and interactive.

The type of interactivity that we propose is closely related to the system described by Cleveland [1993]. In that system, the user's mouse creates a single rectangular brush which either defines the selection (transient mode), or defines data points to add to or remove from the selection (lasting mode or erasing mode). Cleveland also defines several operations on the selection, such as labeling, deleting, and enhanced linking. For example, in a scatter plot matrix, Cleveland uses enhanced linking to highlight which points are selected in each plot.

In Animint, the central concept of interactivity is a selection variable, such as year or country in Figure 1. For each selection variable, only 1 value can be selected at a time, e.g. year=1996 and country=United States. Clicking a plot element changes the value of the corresponding selection variable. Like Cleveland's system, Animint supports enhanced linking to highlight the selected value of each selection variable. In contrast to Cleveland's single rectangular brush that select points in plots of a single data table, an Animint designer may designate several selection variables in plots of several linked data tables. To declare a clickable plot element that changes the selected value of the year variable, an Animint designer writes clickSelects=year.

To achieve data-driven animations, Animint defines one other important operation involving selection variables: showing and hiding subsets of data. For example, in the scatterplot of Figure 1 we draw a point for each country, and want to move and change the size of each point as the selected year changes. To accomplish this, an Animint designer can simply declare showSelected=year which means to show only the data with the selected value of year.

Using just these two keywords, a wide variety of interactive visualizations can be defined. To make the selection automatically change over time (animation), an Animint designer may declare one variable as the time variable. Since it is perceptually advantageous to have smooth transitions in data-driven animations [Heer and Robertson, 2007], an Animint designer may also declare a duration list of selection variables which should have smooth transitions.

The Animint DSL is implemented as an extension of ggplot2 [Wickham, 2009, 2010], which is an R implementation of the grammar of graphics [Wilkinson et al., 2006]. The ggplot2 language was created as a high-level abstraction for non-interactive visualizations. One of the key strengths of ggplot2 is that it allows a designer to declare a visualization in terms of distinct geometric elements, each with a clear aesthetic mapping from data variables to geometric properties. The Animint DSL extends ggplot2 by adding two new aesthetics: clickSelects and showSelected.

The Animint library includes a compiler that converts a list of ggplots to an interactive web visualization rendered using the Data-Driven Documents (D3) library for JavaScript [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. Animint's DSL abandons the DOM standard and sacrifices some of the D3's flexibility to gain expressive power. Thus, designers familiar with ggplot2 syntax are now able to declare complex interactive animations in only a few dozen lines of code using the Animint DSL. This enables R programmers to design a large class of visualizations that were previously intractable to implement.

Finally, another key strength of ggplot2 and D3 for visualization design are the libraries' declarative syntax. Heer and Bostock [2010] proposed a declarative syntax for animated transitions, and studied the benefits of declarative languages for data visualization. Animint is another declarative DSL, but defined at a higher level. It enables designers to focus on data visualization, while the Animint library developers can work on improving the lower-level rendering details.

The rest of this paper is organized as follows: we discuss related work in Section 2, and the design of the Animint system in Section 3. Then we perform a detailed comparison with other R packages in Section 4, and show some example applications of Animint in Section 5. Finally, we share some user feedback in Section 6 and then discuss future work in Section 7.

2 RELATED WORK

In this section we offer a comparison between Animint and several related systems, focusing on libraries/packages with free/open-source software implementations in JavaScript/R. The main difference is that Animint's declarative syntax allows interactive animations to be expressed in fewer lines of code than the other packages.

2.1 D3 and other JavaScript libraries

Animint uses D3 to render interactive animations in a web browser [Bostock et al., 2011]. D3 is more powerful than animint, but is also more verbose. For example, the code in Figure 2 shows that it takes a lot more D3 code than Animint code to define the scatterplot in the WorldBank visualization. Animint's high-level DSL is also able to express interactivity and animation in fewer lines of code than pure D3, but there are some specialized D3 plots which can not be created using Animint.

There are many other libraries which can generate web plots with limited interactivity, but which are unable to produce interactive animations. For example, the rCharts R package is a wrapper around several JavaScript libraries [Vaidyanathan, 2013]. The major difference is that rCharts is more expressive but less flexible than Animint. Thus, rCharts can produce several simple visualizations with only 1 line of R code, but it can not produce interactive animations. In contrast, Animint's high-level DSL can be used to specify more complex visualizations using only tens of lines of R code.

Finally, Dimensional Charting (DC) is another JavaScript library which can produce interactive visualizations consisting of several linked plots [DC.js Team, 2014]. Like Animint, DC builds on top

D3 (Javascript code)

```
svg.selectAll("circle")
  .data(one_year)
  .enter().append("circle")
.attr("cx", function(d){
  return x_scale(d.fertility_rate);
}).attr("cy", function(d){
  return y_scale(d.life_expectancy);
}).attr("r", function(d){
  return size_scale(d.population);
}.style("fill", function(d){
  return color_scale(d.region);
})
```

Vega (JSON file)

```
{ "marks": [{
  "type":"symbol",
  "from": {"data":"this_year"},
  "properties":{ "enter":{
    "x": {"field": "fertility_rate"},
    "y": {"field": "life_expectancy"},
    "size": {"field": "population"},
    "fill": {"field": "region"}
}}}]}
```

Animint (R code)

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

Fig. 2. Comparison of code used to define points on a scatterplot. Note that Animint is not as flexible as D3, but it results in shorter, more expressive code. Animint implements the clickSelects and showSelected aesthetics, but D3 and Vega do not.

of D3. Unlike Animint, DC does not use the grammar of graphics, so DC designers must use more lines of code.

2.2 Animated graphics libraries

One way to achieve animation in an iterative programming syntax is by using a for statement to loop over the time variable. This is the main idea of the animation package [Xie, 2013]. There are two main differences between this system and Animint. First, the only interaction possible with animation is rewinding and fast-forwarding through the animation frames. Second, Animint uses an declarative DSL, which enables more complex interactive animations to be defined in fewer lines of code.

Another way to produce an animated scatterplot of the World Bank data is by using a Google motion chart, available in R through the googleVis package [Gesmann and de Castillo, 2011]. Unlike Animint, googleVis has no high level DSL, so it can not be used to produce a multi-plot visualization with a linked multiple time series.

Finally, a designer can use R code and the gridSVG package to define a plot, and then use JavaScript to define an interaction [Potter, 2011]. In contrast, an Animint designer writes only R code and no JavaScript.

2.3 Libraries based on the grammar of graphics

In this section we discuss the differences between Animint and several other high-level DSLs for data visualization based on the grammar of graphics philosophy [Wilkinson et al., 2006]: ggplot2, ggvis, and Vega.

ggplot2   Animint extends the declarative DSL of ggplot2 [Wickham, 2009, 2010]. Strictly speaking, ggplot2 is for non-interactive and non-animated visualizations. In this paper, we propose the clickSelects and showSelected aesthetics for ggplot2, which extend it to accommodate interactive, animated graphics. This allows R users to create web-ready graphics using the grammar of graphics, which is familiar to many statisticians and R programmers.

ggvis   The authors of ggplot2 are currently working on a package for interactive graphics called ggvis [RStudio, 2014]. Unlike Animint, ggvis does not directly extend ggplot2, but provides a new implementation of some of the same ideas about the grammar of graphics. There

are many other differences between ggvis and Animint, but the main difference is that ggvis is more powerful and verbose when used with the shiny web server [RStudio, 2013], while Animint is more succinct for expressing interactive plots that show/hide data subsets (for a concrete example, see the comparison of World Bank data visualizations in Table 1). At the level of defining plots in R code, a ggvis object corresponds to a single plot, while an Animint object is a list of several plots which are linked using the clickSelects and showSelected aesthetics. Another difference is that sharing a ggvis visualization requires a web server that runs R and shiny, but Animint uses static files and client-side JavaScript so it does not require any special web server software. The two packages also have different interactive features: ggvis uses sliders, checkboxes, and other HTML form elements, and Animint provides clickable SVG elements. This usually results in a richer Animint data visualization using fewer lines of code. For example, in a ggvis of the WorldBank data, it is natural to use a slider to select the displayed year. In contrast, in Animint we used a clickable multiple time series plot for the same purpose, giving the viewer a more comprehensive view of these multidimensional data (Figure 1).

Vega   Like Animint, Vega is a declarative DSL that builds on top of D3 [Trifacta, 2014]. Both sacrifice some of the power of D3, while still allowing a wide variety of plots to be defined in fewer lines of code than pure D3. The main idea of Vega is to declaratively define a visualization using a special type of JavaScript Object Notation (JSON) file. As discussed in Section 3, Animint also internally uses a JSON file to store meta-data about an interactive animation.

There are two main differences between Vega and Animint. First, the Animint user writes R code, but the Vega user writes a JSON file. Second and more importantly, Vega does not currently support the complex data-driven interactive animations that Animint currently supports. However, it should be possible for Vega to support interactive animations in the future, by expanding their JSON file format definition to support mark properties based on the clickSelects and showSelected ideas introduced in this paper.

2.4 Systems which are not publicly available

The system described by Cleveland [1993] involves interaction via a rectangular brush that can select several data points. Its enhanced linking feature highlights the selected data points across several plots,
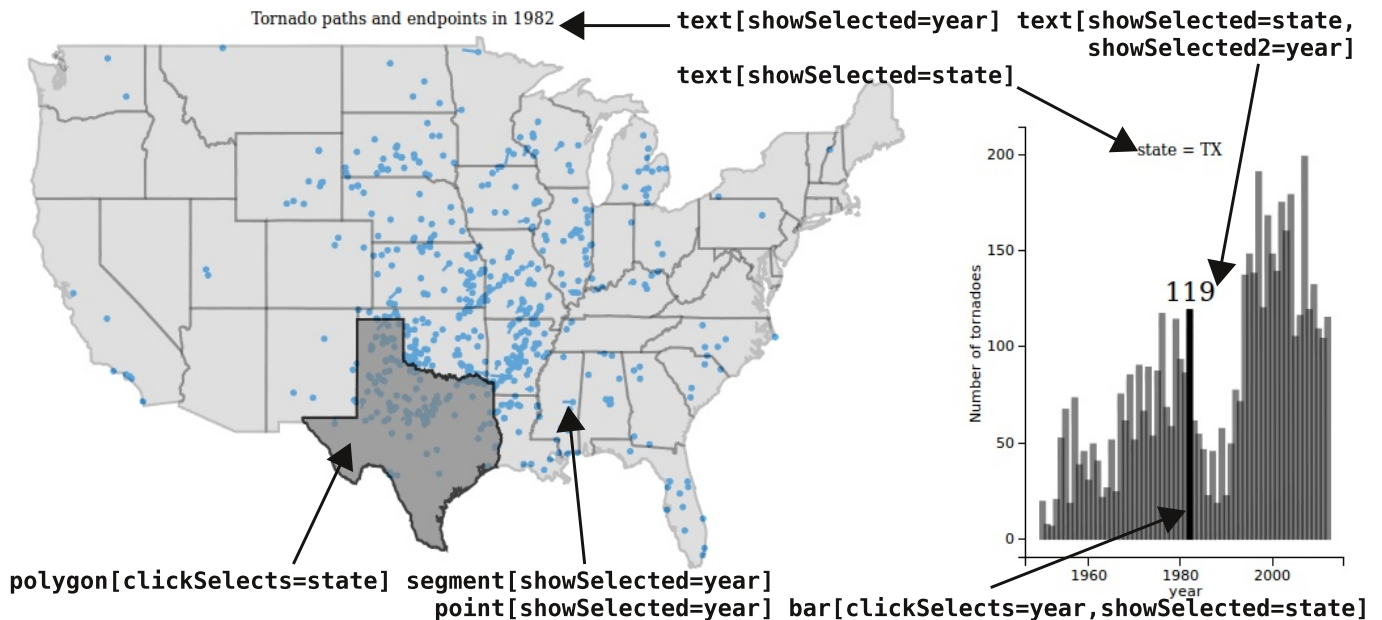

Fig. 3. Interactive animation of tornadoes recorded from 1950 to 2012 in the United States (`http://bit.ly/1hWvYo0`). **Left**: map of the lower 48 United States with tornado paths in 1982. The text shows the selected year, and clicking the map changes the selected state, currently Texas. **Right**: 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).

which is similar to the selection functionality implemented in Animint. There are four important differences between Animint and this system. First, Animint is based on a high-level DSL, but Cleveland's system proposes no such language. Second, Animint supports a brush for every selection variable, but Cleveland's system only supports a single brush. Third, the current implementation of Animint only supports selecting single values, but Cleveland's system supports selecting several values. Importantly, this is a limitation of the current implementation, not the proposed Animint DSL. Finally, Animint is publicly available on GitHub [Hocking and VanderPlas, 2014], but Cleveland's system is not publicly available.

Another system with interactive facilities similar to Animint is Tableau [2014]. Like Animint, it is based on a declarative DSL. The Tableau DSL is called VizQL, which is based on another DSL called Polaris [Stolte and Hanrahan, 2002]. Unlike Animint, there is no free/open-source software implementation of Tableau, so we will not discuss it further.

Finally, 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.

In conclusion, other systems that implement the grammar of graphics are limited to non-interactive plots, and other systems that implement interactive animations do not exploit the powerful grammar of graphics. Animint is the first system that implements both.

3 DESIGN OF THE ANIMINT SYSTEM

In this section we explain the main idea of the Animint grammar for interactive animations. We first explain the grammar using the World Bank example of Figure 1, and then give implementation details of the compiler and renderer.

3.1 The Animint grammar for interactive animations

The main idea of Animint is that a large class of interactive plots can be specified using just two interactive keywords: clickSelects and showSelected. For example, after clicking a clickSelects=year element, the plot is updated to show only the showSelected=year elements corresponding to the selected year.

The first step in the design of an Animint is usually to make a sketch of the desired interactive plot on paper or a whiteboard. First sketch the axes, legends, a few geometric elements, and note which variables will be shown in the linked plots. Also, note the clickSelects and showSelected variables 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.

Animint implements the clickSelects and showSelected interactive keywords as aesthetics in ggplot2. An aesthetic is a declarative mapping from a data variable to a characteristic of a geometric plot element. Some standard ggplot2 aesthetics are

x horizontal position,

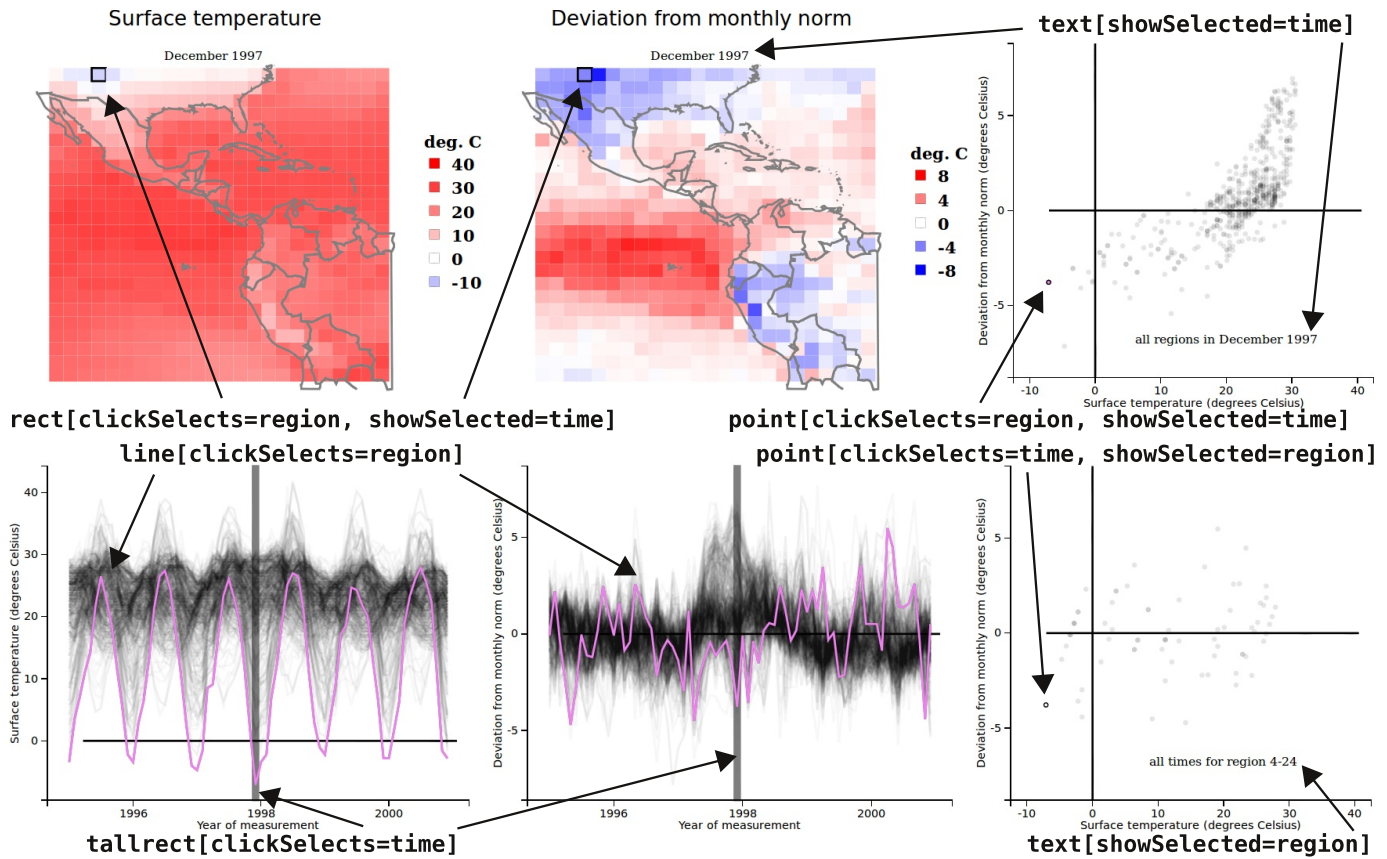**y** vertical position,

color color or fill,

Fig. 4. Visualization containing 6 linked, interactive, animated plots of Central American climate data (http://bit.ly/QcUrhn). **Top**: for the selected time (December 1997), maps displaying the spatial distribution of two temperature variables, and a scatterplot of these two variables. The selected region is displayed with a black outline, and can be changed by clicking a rect on the map or a point on the scatterplot. **Bottom**: time series of the two temperature variables with the selected region shown in violet, and a scatterplot of all times for that region. The selected time can be changed by clicking a background tallrect on a time series or a point on the scatterplot. The selected region can be changed by clicking a line on a time series.

4

size point or line thickness.

For example, consider the following R code which defines the points in the World Bank data scatterplot on the right of Figure 1:

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

The code defines a `geom_point`, which means to create a point for every row in the `WorldBank` 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 interactivity is also defined as a simple variable mapping: `clickSelects=country` means that clicking a point changes the selected country, and `showSelected=year` means to only plot the points for the selected year.

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

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

Note that `data=years` specifies another data table, with 1 row for each year. Animint does not require that linked plots originate from the same data table; the clickSelects and showSelected aesthetics will work as long as the different data tables have common variable names (e.g. the `year` variable is present in both `WorldBank` and `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. Finally, 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 country:

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

Note that Animint allows any number of showSelected aesthetics. In this example, `showSelected=country` combined with `showSelected2=year` means to only show the subset of labels corresponding to both the selected year and country. Since there is only 1 row for each (country,year) combination in the WorldBank data, this has the effect of drawing the selected country's label at the location of the selected year.

Having defined these 3 geometric elements, we combine them in a single ggplot:

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

This completes the definition of the scatterplot. Now, we discuss the time series on the left of Figure 1. First, the tallrects in the background are used to select the year:

```
yearRects <- geom_tallrect(
  aes(xmin=year-1/2, xmax=year+1/2,
      clickSelects=year),
  data=years, alpha=1/2)
```

The tallrect is an Animint extension useful for selecting the variable which is plotted on the x axis, such as `year` in this example. The tallrect plots a rectangle for every row of the `years` data table. The rectangle spans the entire y region, and the `xmin` and `xmax` aesthetics define the left and right limits. Since the designer specified `clickSelects=year`, users can click on a tallrect to select a year. Finally, `alpha=1/2` specifies that the selected tallrect should have an opacity of 50%. Since color is black by default, this results in a selected semi-transparent tallrect which appears grey. Since nonselected clickSelects plot elements have 1/2 less opacity by default, this results in non-selected rectangles which are completely transparent, blending in with the white background.

To create the time series plot, we combine the tallrects above with lines. To declare that clicking a line should change the selected country, we use the `clickSelects=country` aesthetic:

```
timeSeries <- ggplot()+
  yearRects+
  geom_line(
    aes(x=year, y=life.expectancy,
        group=country, color=region,
        clickSelects=country),
    data=WorldBank, size=3, alpha=3/5)
```

Note that both the `timeSeries` and `scatterPlot` objects are valid ggplots. However, plotting them using the standard ggplot2 library will show a non-interactive plot will all geometric elements, including data for all years and countries. To plot them with Animint, we define a list of ggplots and options, then call the `gg2animint` compiler:

```
viz <-
  list(scatterPlot=scatterPlot,
       timeSeries=timeSeries,
       time=list(variable="year", ms=3000),
       duration=list(year=1000))
gg2animint(viz, out.dir="WorldBank")
```

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. We also use the `duration` option to specify a smooth transition over 1 second for the year variable. The `gg2animint` compiler saves some data files and a web page in the `WorldBank` directory, then it opens the interactive plot in a web browser.

3.2 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 1000 lines of R code that converts a list of ggplots and options to a comma-separated values (CSV) file database and a JSON plot meta-data file. The compiler scans the aesthetics in all of the ggplots to determine how many selection variables are present, and which plots to update after a selection variable is clicked. It also uses ggplot2 to automatically calculate the axes scales, legends, and labels. It outputs this information to the JSON plot meta-data file. It also uses ggplot2 to convert data variables (e.g. life expectancy and region) to visual properties (e.g. y position and color), saving this information in the CSV file database. It also copies the rendering engine (`index.html`, `d3.v3.js`, and `animint.js` files) to the plot directory. Since the compiled plot is just a directory of files, the designer can easily upload interactive plots to the web for sharing visualizations with users.

The `animint.js` renderer is implemented in about 1000 lines of JavaScript/D3 code that renders the CSV and JSON plot data files as SVG in a web browser. The main idea for implementing interactivity is that clicking a `clickSelects=year` geom calls the `update_selector` function, which stores the newly selected value of year, then calls `update_geom` to redraw every geom with `clickSelects=year` or `showSelected=year`. Importantly, animation is achieved by using the JavaScript `setInterval` function, which executes `update_selector` with a new value, every few seconds.

4 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 (Table 1). The main result of our comparison is that Animint requires significantly fewer lines of code, since it uses a declarative grammar to specify interactive animations. Note that is it possible to implement the World Bank visualization in pure D3, but would require significantly more code (Figure 2).

4.1 R package animation

We designed a version of the WorldBank visualization with limited interactivity, using 38 lines of R code and the `animation` package (`http://bit.ly/1hnUnkE`). 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 a country to emphasize, and the user does not have that choice. 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.

4.2 Client-server systems like ggvis/shiny

We designed another version of the World Bank data visualization in 84 lines of R code, using the ggvis graphics library combined with the recommended shiny web server package [RStudio, 2013, 2014] (`http://bit.ly/1diUYsg`). 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 1 using only 20 lines of R code with the Animint package. The
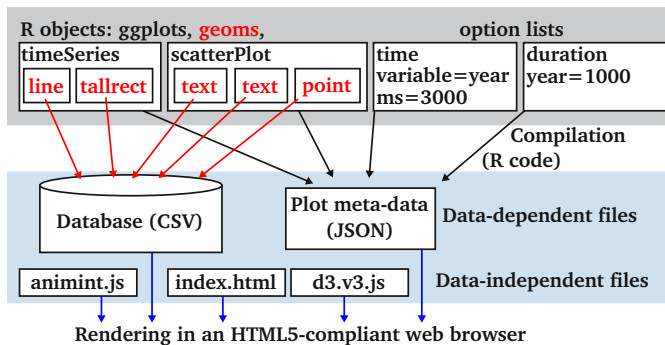

Fig. 5. Schematic explanation of compilation and rendering the World Bank visualization shown in Figure 1. **Top**: the interactive animation is a list of 4 R objects: 2 ggplots and 2 option lists. **Center**: Animint R code compiles data in ggplot geoms to a database of CSV files (⇢). It also compiles 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), programming style, number of interaction variables, and URL of the interactive version.

library	LOC	style	interaction vars	http://bit.ly/
animint	20	declarative	several	1fzfUdN
animation	38	imperative	1 = time	1hnUnkE
ggvis/shiny	84	reactive	several	1diUYsg

reason why the implementation is significantly simpler using Animint is because its DSL is designed specifically for this type of interactive animation.

Another difference is the amount of work required to deploy or share a visualization. A compiled Animint visualization consists of static CSV, JSON, HTML, and JavaScript files which can be easily served with any web server. In contrast, ggvis+shiny requires a web server with R 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 plotting code 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 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 2.3MB of uncompressed CSV data files took 2.5s. 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 communicating with a web server 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.

5 EXAMPLE APPLICATIONS

In this section we discuss the range of examples that we have designed with Animint. Table 2 shows several characteristics of nine different 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 87 lines of code, and it shows 15 variables across 6 linked plots (Figure 4). 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 multi-variate 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 CSV 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 88980 geometric elements at once. 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.

5.1 Animated examples

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

Figure 3 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 time 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 4 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. Animint's DSL allows for this level of flexibility while using minimal lines of code to define the plots and the relationship between them.

5.2 Non-animated examples

When the data to visualize do not contain a time variable, we have found that Animint is still useful for creating interactive but non-animated plots. In fact, five of the nine examples in Table 2 are non-

animated. For example, these 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.

Also, we have found that Animint is useful for exploring pairwise relationships in multivariate data (e.g. using several scatter plots). By using clickSelects, it is easy to explore outliers and other extreme values in the dataset and to see how those relationships translate to other variables. With traditional non-interactive graphics, it would be much more difficult (if not impossible) to explore individual points within a larger multivariate context.

6 USER FEEDBACK AND OBSERVATIONS

We and our users have created interactive visualizations using Animint that would be too complex to implement in other systems. In this section we share comments and constructive criticism that we have obtained from our users.

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."

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 — especially when compared to scrolling through huge spreadsheets! Constructive criticism:

	lines of R code	seconds	MB	rows	onscreen	variables	interactive	plots	animated?	Fig
worldPop	17	0.2	0.1	924	624	4	2	2	yes	
WorldBank	20	2.5	4.0	34132	11611	6	2	2	yes	1
evolution	25	23.7	12.5	240600	2703	5	2	2	yes	
change	36	3.4	2.7	36238	25607	12	2	3	no	
tornado	39	2.1	6.2	103691	16642	11	2	2	no	3
prior	54	0.8	0.2	1960	142	12	3	4	no	
breakpoints	66	0.5	0.3	4242	667	13	2	3	no	
compare	66	11.8	7.8	133958	2140	20	2	5	no	
climate	87	14.3	21.5	253856	88980	15	2	6	yes	4

Table 2. Characteristics of nine interactive visualizations designed with Animint. From left to right, we show the data set name, the lines of R code 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 CSV 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).

finding ways to speed up data loading would help maximize the software's utility for the visualization of large datasets."

Finally, users also appreciated the web-based output format, 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."

7 LIMITATIONS AND FUTURE WORK

There are several limitations to the current implementation of Animint, which suggest avenues for future work.

As discussed in Section 3 and illustrated in Figure 5, the compiler is written in R, and the renderer is written in JavaScript. This means that Animint developers must be proficient in both R and JavaScript. This represents a significant barrier for source code contributions from developers who are proficient with one language but not the other.

Animint users 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 types of interactivity without changing the Animint DSL. Examples include zooming, multiple selection, increase/decrease animation speed, and plot resizing. However, some kinds of interactivity would require additions to the Animint grammer. For example, a tooltip aesthetic could be used to indicate the text the user views when they hover the mouse over a data point.

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 definitely be made even more efficient and responsive. For example, the compilation step currently saves geom-specific CSV files and could be improved by saving CSV files for each showSelected subset of data. This would also allow for faster rendering for large data sets, since then the plots could be displayed as soon as the first selection is downloaded. In contrast, the current code downloads all of the data before displaying the geom.

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. It is free/open-source software available on GitHub [Hocking and VanderPlas, 2014].

REFERENCES

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.

T. D. Hocking and S. VanderPlas. animint R package, 2014. URL https://github.com/tdhock/animint.

S. J. Potter. *Web-based interactive graphics with gridSVG*. PhD thesis, University of Auckland, 2011.

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.

C. Stolte and P. Hanrahan. Polaris: A system for query, analysis and visualization of multi-dimensional 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. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.

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/.