Supplementary materials

0.1 Additional examples

Summary statistics describing complexity and performance of a variety of examples are displayed in Table 1. The climate data visualization has noticeably slow animations, since it displays about 88,980 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. This performance could be improved by leveraging canvas-based (rather than vector-based) rendering, but at the time of writing, **animint** does not support canvas-based rendering. Another **animint** visualization with a similar amount of total rows is based on the evolution data, but since it shows less data onscreen (about 2,703 elements), it exhibits faster responses to interactivity and animation.

0.2 Additional implementation details

As shown in Figure 1, 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.

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 (Heer 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 **??**.

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.

Our current implementation of **animint** depends on a fork of **ggplot2**[1] that contains some minor modifications which are needed to support interactive rendering on web pages.

0.3 Caption of Tornadoes figure

Particular queries may also be stored and shared via a URL, for example: https://bl.ocks.org/faizan-khan-iit/raw/b3912f21ec2750f96e8d1bd4b66463b2/#year=%7B1982%7Dstate=%7BTX%7D. In this link to the interactive version, we also demonstrate the ability to dynamically rescale axes when a new query is triggered. The middle and right panel display the same data, but use different scaling: the middle panel reflects the

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

Table 1: Characteristics of 11 interactive visualizations designed with **animint**. The interactive version of these visualizations can be accessed via http://members.cbio.ensmp.fr/~thocking/animint/. From left to right, we show the data set name, the lines of R code (LOC) including data processing but not including comments (80 characters max per line), the amount of time it takes to compile the visualization (seconds), the total size of the uncompressed TSV files in megabytes (MB), the total number of data points (rows), the median number of data points shown at once (onscreen), the number of data columns visualized (vars), the number of `clickSelects`/`showSelected` variables (int), the number of linked panels (plots), if the plot is animated.

Figure	LOC	seconds	MB	rows	onscreen	vars	int	plots	animated?
worldPop	17	0.2	0.1	924	624	4	2	2	yes
WorldBank	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

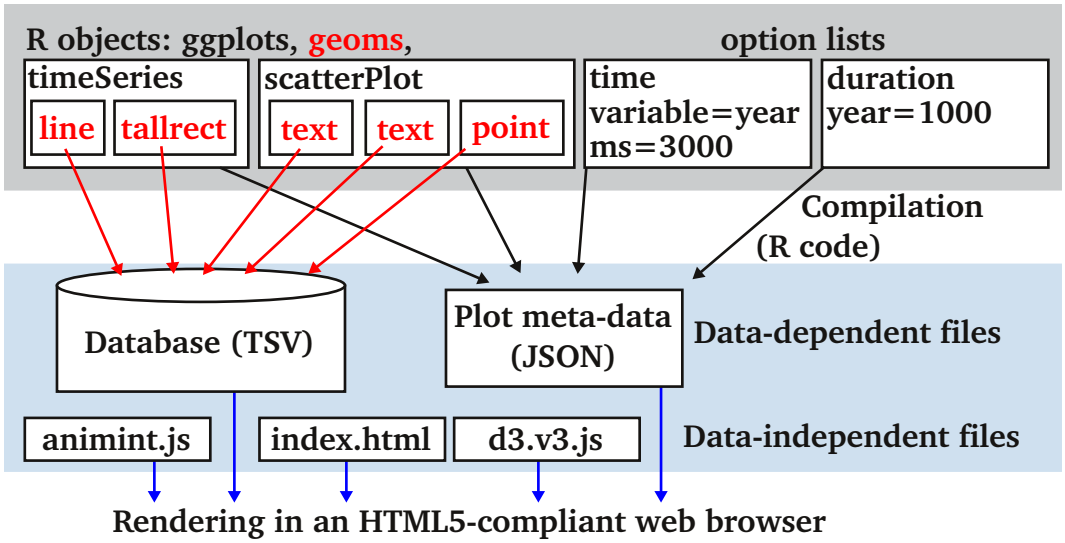

Figure 1: A schematic explanation of compilation and rendering in the World Bank visualization. Top: the interactive animation is a list of 4 R objects: 2 ggplots and 2 option lists. Center: **animint** R code compiles data in ggplot geoms to a database of TSV files (⇾). It also compiles plot meta-data 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 (⇾).

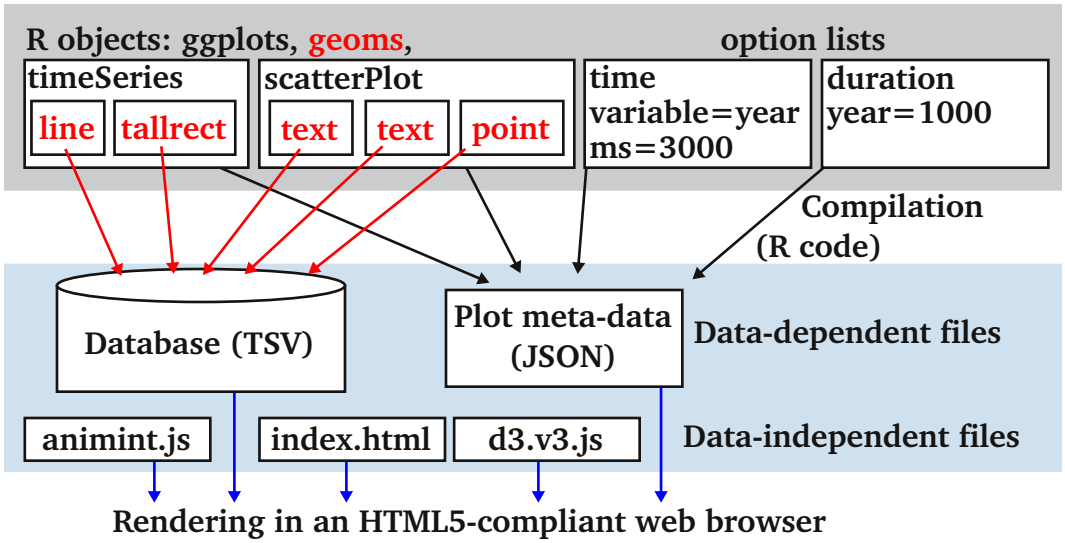

Figure 2: A schematic explanation of compilation and rendering in the World Bank visualization. Top: the interactive animation is a list of 4 R objects: 2 ggplots and 2 option lists. Center: **animint** R code compiles data in ggplot geoms to a database of TSV files (⇾). It also compiles plot meta-data 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 (⇾).

'global' range (US) while the right panel reflects the 'local' range (Arkansas). Futhermore, when an axis update is triggered, it smoothly transitions from one state to next (preserving object constancy in the axis ticks). This helps the viewer better perceive/understand how the range has changed from one state to the next.

1 Comparison study

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

1.1 The Grand Tour

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

When touring data, it is generally useful to link low-dimensional displays with the tour itself. The video in Figure 3 was generated with our current **animint** implementation, and points are selected via mouse click which reveals that points with high marginal density are located in the ellipsoid center while points with a low marginal density appear near the ellipsoid border. In this case, it would be convenient to also have brush selection, as we demonstrate in Figure 4 which implements the same touring example using the R packages

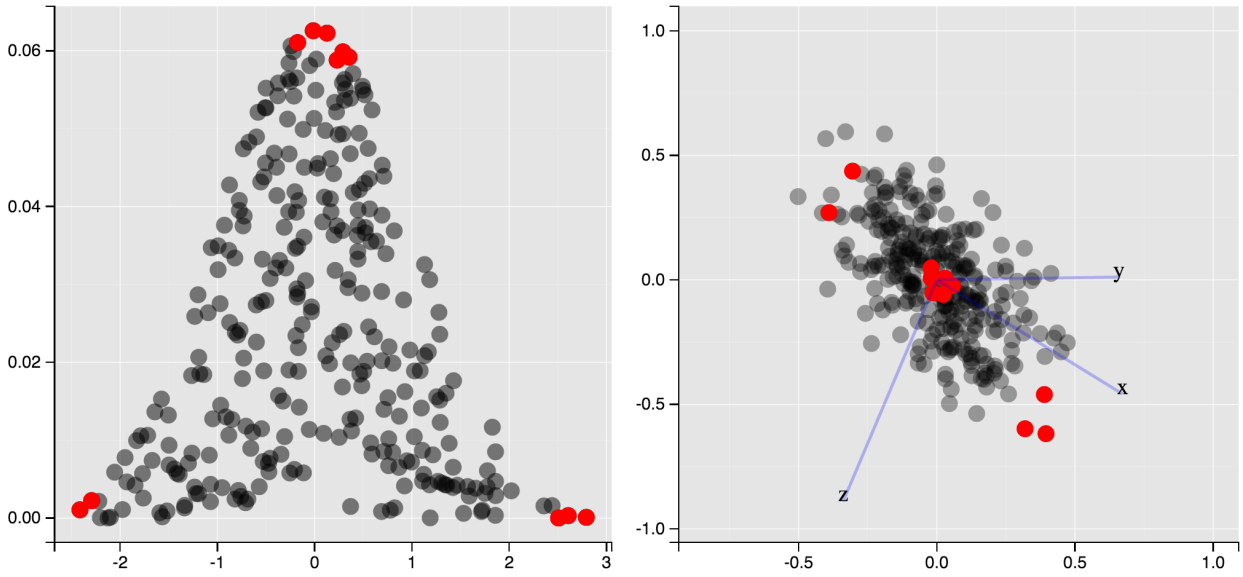

Figure 3: Linked selection in a grand tour with **animint**. A video demonstration can be viewed online at https://vimeo.com/160720834

ggvis and **shiny**. The brush in Figure 4 is implemented with **shiny**'s support for brushing static images, which currently does not support multiple brushes, making it difficult to select non-contiguous regions.

This example helps point out a few other important differences in using **animint** versus **ggvis**+**shiny** to implement "multiple linked and dynamic views" as described in Ahlberg, Williamson, and Shneiderman (1991) and Buja et al. (1991). Maintaining state of the linked brush in Figure 4 requires both knowledge and clever use of some sophicated programming techniques such as closures and reactivity. It also requires knowledge of the **shiny** web application framework and a new approach to the grammar of graphics. On the other hand, maintaining state in Figure 3 requires a few different `clickSelects`/`showSelected` mappings. As a result, we believe **animint** provides a more elegant user interface for this application.

The touring example also helps point out important consequences of the design and implementation of these two different systems. As mentioned in Section 0.2, our current **animint** implementation requires every subset of data to be precomputed before render time. For visualizations such as tours, where it is more efficient to perform statistical computations on-the-fly, this can be a harsh restriction, but this is a restriction of our current implementation (not a restriction of the framework itself). As a result, when touring a large high-dimensional space, where many projections are needed, **ggvis**+**shiny** may be desirable since the projections are computed on the server and sent to the browser in real-time. This works fine when the application is running and viewed on the same host machine, but viewing such an application hosted on a remote machine can produce staggered animations since client-server requests must be performed, processed, and rendered roughly 30 times a second. Also, generally speaking, the **animint** system results a more pleasant experience when it comes to hosting and sharing applications since it doesn't require a Web Server with R and special software already installed.

1.2 World Bank example

Even as experienced **ggvis**+**shiny** users, we found it quite difficult to replicate the World Bank example using **ggvis**+**shiny** and Tableau, and were not able to completely replicate it due to a lack of a mechanism for coordinating indirect and direct manipulations. Overall the visualization is pretty similar, but lacks a few important features. In particular, there is no way to control the selected year using both the slider (indirect) and clicking on the ggvis plot (direct). It also lacks the ability to click on a country time series and label

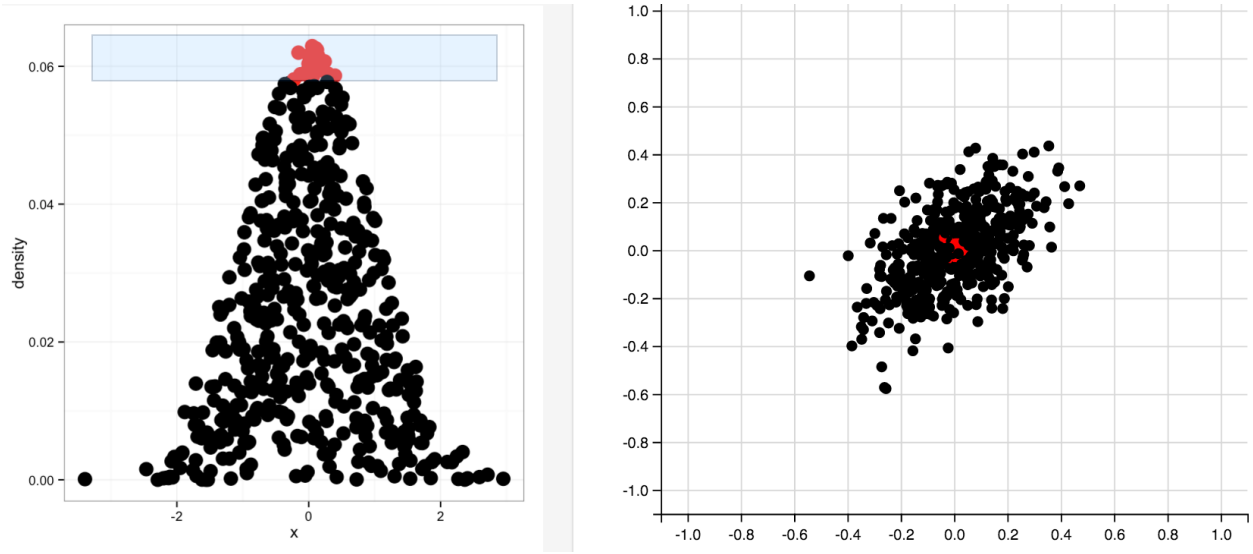

Figure 4: Linked selection in a grand tour with **ggvis** and **shiny**. A video demonstration can be viewed online at https://vimeo.com/160825528

the corresponding point on the scatterplot. This might be possible, but we could not find a way to update a plot based on a click event on a different plot. Even with this lack of functionality, the **ggvis**+**shiny** is significantly more complicated and requires more code (about 100 lines of code compared to 30).

It was also impossible to completely replicate Figure **??** using Tableau essentially because the example requires a *layered* approach to the grammar of graphics. In particular, since graphical marks and interaction source/target(s) must derive from the same table in Tableau, it was impossible to control the clickable multiple time series and the clickable tallrects in different ways based on the two different selection variables. In other words, in Tableau, selections are managed on the plot level, but in **animint**, selections are specific to each graphical layer.

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

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

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

Heer, Zhicheng Liu AND Biye Jiang AND Jeffrey. 2013. "ImMens: Real-Time Visual Querying of Big Data." *Computer Graphics Forum (Proc. EuroVis)* 32 (3). http://vis.stanford.edu/papers/immens.

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