R3S.js – Towards Responsive Visualizations

Juliana Leclaire **Aurélien Tabard**
Université de Lyon, CNRS
Université Lyon 1, LIRIS, UMR5205,
F-69622, France
aurelien.tabard@univ-lyon1.fr

ABSTRACT
We present our preliminary work on R3S.js, a Javascript library supporting the development of Responsive Visualizations, i.e., Web visualizations adapted to the device they are displayed on. R3S.js is based on D3.js and brings the following contributions: 1. Handling of tooltips and especially their triggering on tactile devices; 2. Abstraction of input events to avoid dealing with mouse, touch or styluses separately; 3. Pre-defined media-queries to automatically control the size of graphical elements depending on the device size and resolution. And 4. Automated resizing of specific visualizations. We show how basic D3 line-chart and treemap could benefit from more responsiveness. And we conclude with a discussion on automated adaption of visualizations to devices' properties, and whether Responsive Web Design principles provide good adaptation strategies.

Author Keywords
Visualisation; adaptation; plasticity; Responsive Web Design; mobile; d3js.

ACM Classification Keywords
D.2.2. Design Tools and Techniques; H.5.2 User Interfaces.

INTRODUCTION
Since the beginning of the 1990's, digital devices of varied form factors and supporting various interaction modalities have emerged. User Interface (UI) adaptation mechanisms are an interesting strategy to avoid device-specific development [6]. Besides the devices themselves, adaption efforts started to also consider the environment and the users, i.e. the context of use, for instance with plastic UI [13].

More recently, with the commercial success of smartphones and tablets, Responsive Web Design (RWD) emerged as a simple approach to adaptation. Unlike richer adaptive approaches, RWD does not take into account the specificities of users or the environment but only devices' properties. RWD principles center mostly around fluid layout of Web pages on mobile devices, tablets, and computers screens. We can summarize the responsive approach to the following points[1]:

- Adapt the spatial layout to the screen size.
- Adapt images to the screen resolution (especially with ultra high fidelity displays).
- Simplify pages for mobile devices with low bandwidth.
- Make links and buttons clickable and touchable.

These principles are widely used for Web pages today and are starting to be adopted for images[2] and videos. Although they have been taken into account in Web-based information visualization, it is often in an ad-hoc manner. For example, the New York Times visualizations are often designed to handle touch. The approach consisting in designing first for devices with a small screen size and then extending to larger and more capable devices has been described in practioner conferences, for example by Gabriel Florit from the Boston Globe at OpenVizConf 2013[3] or Dominikus Baur at JSConfEU 2014[4].

Building on this previous work, we present our preliminary work on R3S.js, a library based on D3.js that facilitates the development of responsive visualizations. R3S.js helps developers to incorporate adaptation mechanisms in their visualizations. More specifically we bring four contributions:

1. Abstraction of input events to avoid dealing separately with mice, touch or styluses.
2. Management of tooltips, especially by providing means to have them pop-up on touch devices.
3. Predefined media-queries to automatically adjust the size of the main graphical elements (font, tooltips, and label size) to the size and resolution of the device.
4. Automated resizing of the visualization itself.

[1] Responsive Web Design Demystified, 2011, Matt Doyle, http://www.elated.com/articles/responsive-web-design-demystified

[2] https://www.w3.org/community/respimg/

[3] Gabriel Florit, 2013, On Responsive Design and Data Visualization, OpenVis Conf, https://youtu.be/BrmwjVdaxMM

[4] Dominikus Baur, 2014, Web-based data visualization on mobile devices, JSConfEU, https://youtu.be/X2ZlDrx6dAw

We discuss the challenges of making responsive visualizations, based on the case of a simple Treemap and a line-chart, and show how R3S.js could help. We conclude on whether Responsive Web Design mechanisms are suited for visualization.

RELATED WORK

While Responsive Web Design gained a lot of popularity, research on adaptation also proposed to leverage Web standards and emerging HTML standards, for instance using columns to let content flow on screens of various sizes [8], or to handle touch in a generic manner [9].

Mobile visualizations

Several examples of visualizations for phones [2] or large interactive displays [7] have demonstrated the relevance of touch devices to visualize and explore data. Regarding interaction, recent work such as TouchWave [1] or Kinetica [10], demonstrated that touch input could support understanding and create engaging experiences of data exploration. However, most visualization toolkits are still geared towards interaction with a mouse and keyboard. Zoomable visualizations lend themselves particularly well to adaption [5], but they nonetheless require specific adjustments.

In their survey of mobile visualizations, Sadowski and Heidmann [11] note that "*Tablets and smartphones are not only varying in size but are also providing new interaction methods or sensors which enable new design possibilities*", which suggests that adaptation could be about retargeting input and output modalities to other ones depending of their availability on different devices.

D3.js

D3.js[5] is a Javascript library sometimes referred to as a visualization kernel [4], in the sense that it provides the core functionalities to create novel Web based visualizations. D3 is based on the browser's Document Object Model (DOM), which enables developers to apply transformations to data Since D3 is based on Scalable Vector Graphics (SVG) and the DOM, scaling to different screen size is relatively straightforward. And the use of Web standards such as CSS makes it possible to modify graphical properties of visualizations. Nonetheless adaptation mechanisms are not offered by D3.js. This can be explained by D3 focus on offering rich control on the basic elements of interactive visualizations, rather than offering a library of ready to use visualizations. This motivates our work to offer alongside D3.js a library supporting the development of adaptive visualizations.

[5] http://d3js.org/

CHALLENGES IN DESIGNING RESPONSIVE VISUALIZATIONS

Based on the related work and our experience adapting a line-graph and a Treemap to various devices, we have identified the following challenges.

Variations in input modalities across devices

Touch devices have different input modalities than laptop or desktop computers. For instance, while a mouse scroll often controls zoom levels on computers, a pinch gesture is generally preferred on touch devices. Although the correspondence is well accepted for zooming, there is rarely a generic correspondence between a touch-based and a mouse-based interaction technique. For example, hovering with a mouse, is difficult to translate to touch devices. Different applications and operating systems handle this differently, either through a long touch, or quick tap or a gesture.

Besides interaction modalities, the form of the devices has an impact on possible interactions. Finger size is rarely a problem on large screens, but can become one on small mobile devices [3], where the hand or even a single finger can easily hide the points of interests.

Variations in displays

The main motivation behind RWD is to manage screens of various sizes and resolutions, i.e., the available display space. We separate display sizes into five broad categories: Large screens with a diagonal size larger than 27", desktop computers with a size of 19" to 27", laptops with screens ranging from 11" to 17", tablets between 8" and 11", and smartphone between 5" and 8".

Screen resolution can vary a lot and smartphones may have more pixels than a 55" touch screens. So relying on either screen resolution or screen size, or even a mix of the two such as pixels per inch (ppi), may not be satisfying. Text at a small size on a large screen with low resolution may become sharp but unreadable of a small screen with high resolution if proportions are preserved. For a given resolution, the smaller the screen, the smaller graphical elements will become. Some high-resolution devices (e.g. Retina devices) already offer a lower "virtual" resolution to simplify display management.

Finally, besides screen size and resolution, the width-height ratio of a smartphone, a tablet and a computer screen are often different.

Use cases

We studied two simple visualizations offered alongside the D3.js library to better understand the challenges of developing responsive visualizations. First, a line chart, presented the following challenges (see figure 1):

- The size of the line and the axes are not updated when the window is resized.
- The quantity of information displayed does not depend of the available screen space and its resolution. A line with lots of variations can

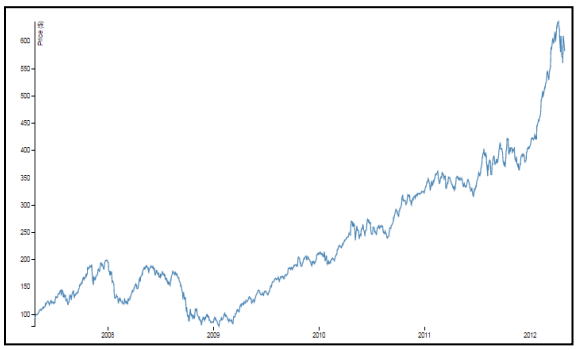

Figure 1. Timeline visualization offered with d3.js

become unreadable as they become squeezed horizontally.

- It is difficult to explore specific zones or points on the line.
- When adding interaction capabilities to the visualization, the "fat finger" problems appears.

Second, a Treemap visualization revealed several problems when displayed on different devices :

- Labels were displayed as the visualization loaded, and when no space was available they were not displayed. This is not a problem on large displays but quickly becomes one when most labels are not displayed.
- The layout of the rectangles is not adapted to the screen size.

R3S.JS

Based on the challenges and issues identified above, we have started to develop R3S.js[6] (figure 2), a library to ease the development of responsive visualizations. We present here our preliminary work on the library.

Event management

R3S.js offers a `ResponsiveEvent` class to bind callback functions to objects when an event is triggered. By default, `ResponsiveEvent` establishes a correspondence between mouse events and touch events. Since the default correspondence may not always be the most appropriate, it can be changed by extending the object. Depending on the use case, it can be better to use specific events rather than others, e.g. mouseover event can be associated to a touchenter in some cases, a touchmove in other cases or even a simple tap depending on the action triggered by the event.

Tooltip Management

Tooltips are a classical method to display extra information about points of interests while keeping the context visible. Mouse hovering often triggers Tooltips. But very few touch devices have the ability to detect finger moving over the surface. The `Tooltip` object makes tooltip use more

Figure 2. Adaptation of a Treemap visualization with R3S.js

straightforward by removing the need to handle different input event listeners. At the moment, developers still have to handle callbacks and dynamically assign the content of the tooltip related to the hovered object. This could probably be improved in future versions of R3S.js, so that the content of tooltips is defined with the object.

Media queries

Media queries enable developers to specify rules that change the CSS style of a page based on some conditions. Although media queries were originally designed to link a specific style to a specific medium (e.g. printouts or screens), media queries now support the activation of styles when some criteria are met, for example a device or window having a given width, this is called a breakpoint.

Breakpoints

We have defined a series of breakpoints adapted to visualizations, especially on small devices, while taking into account portrait and landscape orientation (see Table 1). Besides size and orientation R3S media queries also consider the type of devices. Further work would involve dealing with "real" displayed sizes using ppi instead of pixels.

Min-width	Max-width	Orientation	Device
X	320	X	Phone
321	768	X	Phone-landscape
1024	X	X	Desktop
1824	X	X	Large screen
768	1024	X	Tablet
321	768	landscape	Phone-landscape-strict
321	768	portrait	Phone-portrait-strict
768	1024	landscape	Tablet-landscape-strict
768	1024	portrait	Tablet-portrait-strict

Table 1. Media queries.

Media queries configuration with Less

Less[7] is a CSS pre-processor. It enables developers to generate style sheets using variables, functions or inheritance. Default values for media queries breakpoints

[6] http://juliana23.github.io/responsiveVisualisations/

[7] http://lesscss.org/

and textual elements such as fonts, labels and tooltips size are defined in a less file. A JavaScript utility class is dedicated to setting less variables and adjust them if needed. It generates getters and setters for all the variables defined in less files. Developers can then change dynamically the style of their visualizations.

Axes management

Finally, R3S.js offers an `Axis` objects that handles visualization resizing. Whenever a resizing happens (on load or later on), `Axis` will recompute and redraw its axis automatically. Developers only set an initial container size; the object will then compute the initial ratio and marks and will maintain the ratio and adjust the marks whenever the object is redrawn.

CONCLUSION AND FUTURE WORK

We have presented our preliminary work on R3S.js a library for responsive visualizations. Further work is required to make the library more in line with D3 philosophy and to better work alongside it. D3 being low level, it also means that we only tackled a very limited set of visualizations in our work, and that more efforts are needed to make a library like R3S.js really generic and reusable.

An alternative to working with D3, would be to explore if toolkits of more ready to use visualizations wouldn't be a better place to offer responsive facilities in a totally transparent manner. Or also in the spirit of reducing the amount of code required, another possibility would be to incorporate responsive elements in a declarative visualization format such as Vega[8] [12].

Our work only touches on one aspect of adaptation; we mostly ignored how visualizations would be explored differently on a smartphone and a computer. We can imagine that in many cases the questions asked while interacting with a tablet on a sofa would be different from the ones asked while sitting on a desk in font of a dual-display.

In this perspective, it would be interesting to explore how devices could complement each other. For instance, how one could explore datasets using both a tablet and a large screen. Each device supporting interaction that is most suited, e.g., focused exploration on a tablet and context on a large screen, and how adaptive methods could be used to split relevant visualization elements to the right devices.

REFERENCES

1. Baur, D., Lee, B., & Carpendale, S. (2012). TouchWave: kinetic multi-touch manipulation for hierarchical stacked graphs. In Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces (pp. 255-264). ACM.
2. Bederson, B. B., Clamage, A., Czerwinski, M. P., & Robertson, G. G. (2004). DateLens: A fisheye calendar interface for PDAs. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *11*(1), 90-119.
3. Boring, S., Ledo, D., Chen, X. A., Marquardt, N., Tang, A., & Greenberg, S. (2012). The fat thumb: using the thumb's contact size for single-handed mobile interaction. In Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services (pp. 39-48). ACM.
4. Bostock, M., Ogievetsky, V., & Heer, J. (2011). D^3 data-driven documents. Visualization and Computer Graphics, IEEE Transactions on, 17(12), 2301-2309.
5. Blanch, R., & Lecolinet, E. (2007). Browsing zoomable treemaps: Structure-aware multi-scale navigation techniques. Visualization and Computer Graphics, IEEE Transactions on, 13(6), 1248-1253.
6. Browne, D., Totterdell, P. & Norman, M. (eds.) (1990), Adaptive User Interfaces, Computer and People Series, Academic Press.
7. Jakobsen, M. R., & Hornbæk, K. (2013). Interactive visualizations on large and small displays: The interrelation of display size, information space, and scale. *Visualization and Computer Graphics, IEEE Transactions on*, *19*(12), 2336-2345.
8. Nebeling, M., Matulic, F., Streit, L., and Norrie. M. C., (2011). Adaptive layout template for effective web content presentation in large-screen contexts. In *Proceedings of the 11th ACM symposium on Document engineering* (DocEng '11). ACM, New York, NY, USA, 219-228.
9. Nebeling, M., & Norrie, M. (2012). jQMultiTouch: lightweight toolkit and development framework for multi-touch/multi-device web interfaces. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems* (pp. 61-70). ACM.
10. Rzeszotarski, J. M., & Kittur, A. (2014). Kinetica: Naturalistic multi-touch data visualization. In Proceedings of the 32nd annual ACM conference on Human factors in computing systems (pp. 897-906). ACM.
11. Sadowski, S., & Heidmann, F. A Visual Survey of Information Visualizations on Smartphones.
12. Satyanarayan, A., Wongsuphasawat, K., & Heer, J. (2014). Declarative interaction design for data visualization. In *Proceedings of the 27th annual ACM symposium on User interface software and technology* (pp. 669-678). ACM.
13. Thevenin, D., & Coutaz, J. (1999). Plasticity of user interfaces: Framework and research agenda. In *Proceedings of INTERACT* (Vol. 99, pp. 110-117).

[8] http://vega.github.io/vega/