Data Visualisation on Mobile

Group 3

Hussain Hussain, Lukas Burgstaller, Peter Grassberger, Nikita Lvov

706.057 Information Visualisation SS 2018
Graz University of Technology

13 May 2018

Abstract

Today the number of mobile devices growing everyday significantly. In these terms it is very important to optimize data visualizations for them. In first place comes scalability because the final graphic must fit properly on a big variety of screen sizes and resolutions. There are two main view modes on the devices – Landscape and Portrait. Therefore the end-image must scale across the screen according to the view method. Modern mobile hardware is controlled by touchscreens and most people are used to not only to see data but also interact with it. For example, the basic interaction allows users to rotate diagrams to change its type. That is why it is also important to adjust such functions as zooming, selection, etc. for comfortable usage. With the set of sensors that are available on the devices there are a lot of additional possibilities of data visualization possible. From another side here is the problem of lack of performance in comparison to desktop PCs. Hence the data visualization on mobile devices should be lightweight or created with the optimized frameworks.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

1.1 Mobile Devices

Mobile devices are computers that can be carried around, they can be hold and operated in ones hands. All following aspects that mobile devices are known by follow this definition. They need to be small enough. They have a touchscreen and do not require mouse or keyboard that would be used with PCs. In addition they can also have some physical buttons.

Mobile devices are connected to the internet via wireless connections like 4G mobile broadband or WiFi technology and to other devices via bluetooth. They are battery powered, aware of their orientation, motion and geographical position. They include smartphones, tablets and smartwatches. Wikipedia 2018

Mobile devices are also used differently that other computers, they are used on the go. The environment that the devices is used in changes together with the light situation, users might be distracted by her surroundings.

1.2 Responsive Design

As mentioned before, smaller sizes and therefore smaller screen sizes are what define mobile devices. The range of sizes between smartwatches and tablets is large. The design pattern Responsive Design (or Responsive Web Design) was established.

Responsive Design uses three techniques:

- ◇ Fluid grid: Grids are well known in graphics design for a long time, they divide the screen space up in different areas of content. Fluid grids use relative units to fluidly adopt to different screen sizes.
- ◇ Relative units: All visual elements (text, images, the grid, etc.) need sizes and units for these sizes. There are static and relative units. Relative units depend on some other size, for example a base font size or the screen width or height.
- ◇ Break points: Break points are points on a scale that mark a change in design. Points can be a certain screen width or height, when crossed some design aspects change based on this condition. Elements could be hidden, shown, scaled or rearranged differently.

With these three techniques a designer can face the challenges that screens provide, but mobile devices are more than only different screen sizes.

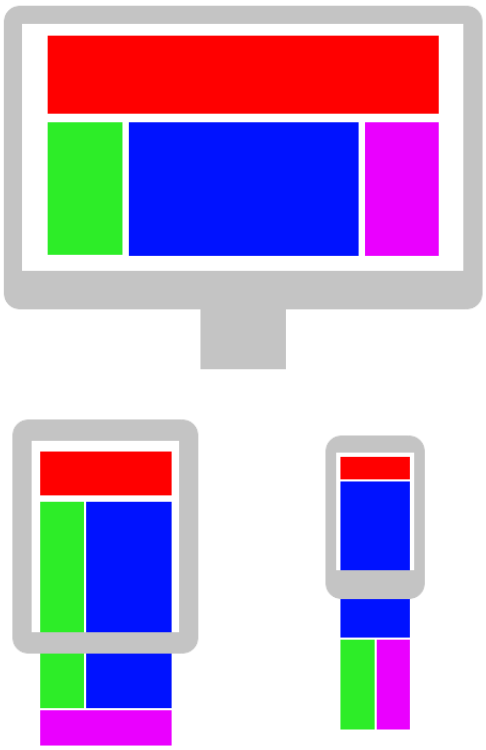


Figure 1.1: Content can be arranged differently for every device with responsive design. [Tomáš Procházka, Creative Commons CC0 1.0]

1.3 Data Visualisation on Mobile

Responsive Design does not cover all aspect of mobile devices. Therefore this paper introduces the following definition:

Data Visualisations on Mobile are data visualisations that adapt to the limitations and take advantage of additional capabilities of mobile devices and how they are used.

Limitations could be different (mostly smaller) screen sizes. Additional capabilities are more inputs (sensors) and outputs, like the touchscreen or orientation sensors. Mobiles devices are used differently than other devices, users are on the go or otherwise distracted.

Chapter 2

Technologies

2.1 Visualisation Rendering on the Web

There are two main categories of rendering visualisations on the web: Scalable Vector Graphics (SVGs) and HTML Canvas element rendering.

SVG is a vector based image file format that can be included in HTML websites like any other image format. In addition SVGs can also be used inline within an HTML website, by using the SVG element with its children elements as an HTML elements, excluding the SVG doctype. SVGs are styled with Cascading Style Sheets (CSS) as HTML is styled, only with a different set of properties.

The HTML Canvas element comes in two variants: two- and three-dimenional (3D). The 3D option is Web Graphics Library (WebGL) based and therefore allows GPU-acceleration.

Both SVG and canvas are elements within an HTML document. The HTML context can provide additional information that supports the graphical content. HTML with CSS itself (excluding SVG and canvas) is very powerful but should not be used to created visualisations.

2.1.1 Responsive Web Technology

The web technology stack is very well equipped for responsive design. There are standardized CSS selectors, properties and units that can used. More elaborate solutions can be created with javascript.

Fluid grids can be created with the CSS properties `flexbox` and `grid`. Flexbox only supports 1-dimensional (rows or columns) layouts, but multiple Flexboxes can be used to create elaborate layouts. The grid property supports 2-dimensions (rows and columns) by itself. As of the time of writing CSS grid is still not as widely supported by browsers as flexbox.

```
<div class="content">
    <section>first section</section>
    <section>secon section</section>
</div>
<style>
.content {
    display: flex;
    flex-direction: row;
}
</style>
```

Listing 2.1: flexbox example

CSS knows many different units, with the static unit of pixels `px` being widely known. Relative units are: `em`, `rem`, `vw`, `vh`, `\%`. The former two are relative to font sizes, `em` relative to the font-size of the current element and `rem` to the fonts size of the root element. The latter are relative to the screen width or height.

Break points can be created with the CSS3 Media Query selectors. In the following example the media query responds to the maximum screen width of 50em. As long as the screen width stays smaller than 50em the code within the brackets is active. Relative units are recommended in media queries.

```
@media (max-width: 50em) { /* adjust something here */ }
```

Listing 2.2: mediaquery example with relative units

Besides CSS, javascript can also be used to respond to different screen sizes. The following example shows how one would selector for screens less wide that 800 pixels. Javascript itself cannot select for relative units by itself, they would need to be calculated. For the sake of understandability absolute values are used in the example. Going beyond responsive design, a lot more can be done with javascript.

```
if (screen.width < 800) { /* adjust something here*/ }
```

Listing 2.3: javascript mediaquery example

2.2 Rendering on iOS

2.2.1 Core Graphics

The easiest way to do 2D graphics on iOS is Core Graphics. CoreGraphics is a system library that is well integrated and heavily used by the UIKit, iOS' UI framework.

To draw on the screen you need to override the draw (`_ rect: CGRect`) method of the UIView you want to draw in.

```
override func draw(_ rect: CGRect) {
  let path = UIBezierPath(ovalIn: rect)
  UIColor.green.setFill()
  path.fill()
}
```

Listing 2.4: overwrite draw Core Graphics function

2.2.2 OpenGL

The second way to generate hardware accelerated graphics. iOS provides in the GLKit framework tools, most notably the GLKView to integrate OpenGL code into the standard iOS user interface.

The GLKView works very similar to the standard UIView, in that you need to override the draw(_ rect: CGRect) method, only that you can use OpenGL code to draw in this case.

```
- (void)drawRect:(CGRect)rect
{
    // Clear the framebuffer
    glClearColor(0.0f, 0.0f, 0.1f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Draw using previously configured texture, shader, uniforms, and
       vertex array
    glBindTexture(GL_TEXTURE_2D, _planetTexture);
    glUseProgram(_diffuseShading);
    glUniformMatrix4fv(_uniformModelViewProjectionMatrix, 1, 0,
       _modelViewProjectionMatrix.m);
    glBindVertexArrayOES(_planetMesh);
    glDrawElements(GL_TRIANGLE_STRIP, 256, GL_UNSIGNED_SHORT);
}
```

Listing 2.5: GLKView: draw with OpenGL

2.2.3 Content in UIWebView

The third and recommended way is to rely on a UIWebView and load HTML content into the UI. This also allows you to use more advanced libraries, since most visualisation libraries are written in Javascript.

```
private let webView = UIWebView()
webView.scrollView.isScrollEnabled = false
addSubview(webView)
webView.loadHTMLString(htmlString, baseURL: nil)
```

Listing 2.6: UIWebView: Use load HTML content to use javascript libraries

2.2.4 SVG

The final way to render custom graphics in iOS is to generate SVG code and load it into the UI, again using a web view.

2.3 Additional Native vs Web APIs

Just about every native sensor input is available via javascript. Some of the APIs mentioned above need permission from the user to be accessed. Permissions are requested and granted in different ways and times depending on platform, for mobile applications on installation and for websites on load or usage.

Web (javascript)	Native (iOS)	Native (Andriod)
Navigator.battery	UIDevice	BatteryManager
Navigator.geolocation	Core Locatoin	LocationManager
Navigator.connection	Reachability	CONNECTIVITY_CHANGE
MediaDevices.getUserMedia()	AVFoundation	MediaRecorder
Event Listeners Touch	UIKit	MotionEvent
Device Orientation API	UIDevice	WindowManager

Table 2.1: Comparison of additional Nativ vs. Web APIs

Chapter 3

Libraries

3.1 Web Libraries

There is a huge amount of web libaries for visualisations that can be freely used. This selection is limited to libraries that explicitly state that they are made for mobile devices or mobile ready.

3.1.1 graph-scroll

graph-scroll is an scrolling events framework. With it events can be set on certain scrolling heights that trigger design changes within a visualisation. It is d3.js version 4 based and MIT licensed.

3.1.2 dygraphs

dygraphs is a library that exclusively produces interactive and zoomable time series charts. It is d3.js version 4 based and MIT licensed.

3.1.3 earthjs

Earthjs is a JavaScript library for easy building orthographic globe. Framework giving 3D graphic visualization experience. Originally was inspired by planetary.js (canvas) and Faux-3d Shaded Globe (SVG) and both were created using D3-v3.

The globe itself is fully interactive. Zooming, rotating and selecting functions are available for users. The layers of the globe are represented by combination of SVG, Canvas and Threejs.

SVG for quickly prototyping the globe as it used standard SVG DOM element so event and css can be applied to each element. Canvas for more data point that need to be render and UX experience stay in good shape. Interactivity or mouse detection are available for hovering, click and double click. detect country or point of location. WebGL/Threejs is a way to go if eye catchy of globe is needed and lots of data, or want to be better CPU utilization by moving some intensive calculation to GPU. Interesting Data Visualization can be created by combining SVG, Canvas and Threejs(WebGL) like: choropleth globe using Canvas or Threejs, heatmap globe by rendering heatmap on canvas and use that canvas as a texture in Threejs, flightLine to connect two datapoint using Threejs and coloring target location (usually country) using Canvas. flashy bullet that travel along the way of flightLine is there including the mouse event using Threejs.

Internal plugins library has a fair size and includes more than 60 plugins. Framework also have an opportunity for using self-created plugins. Plugins is a function created in "earthjs.plugins" namespace, return with JavaScript object.

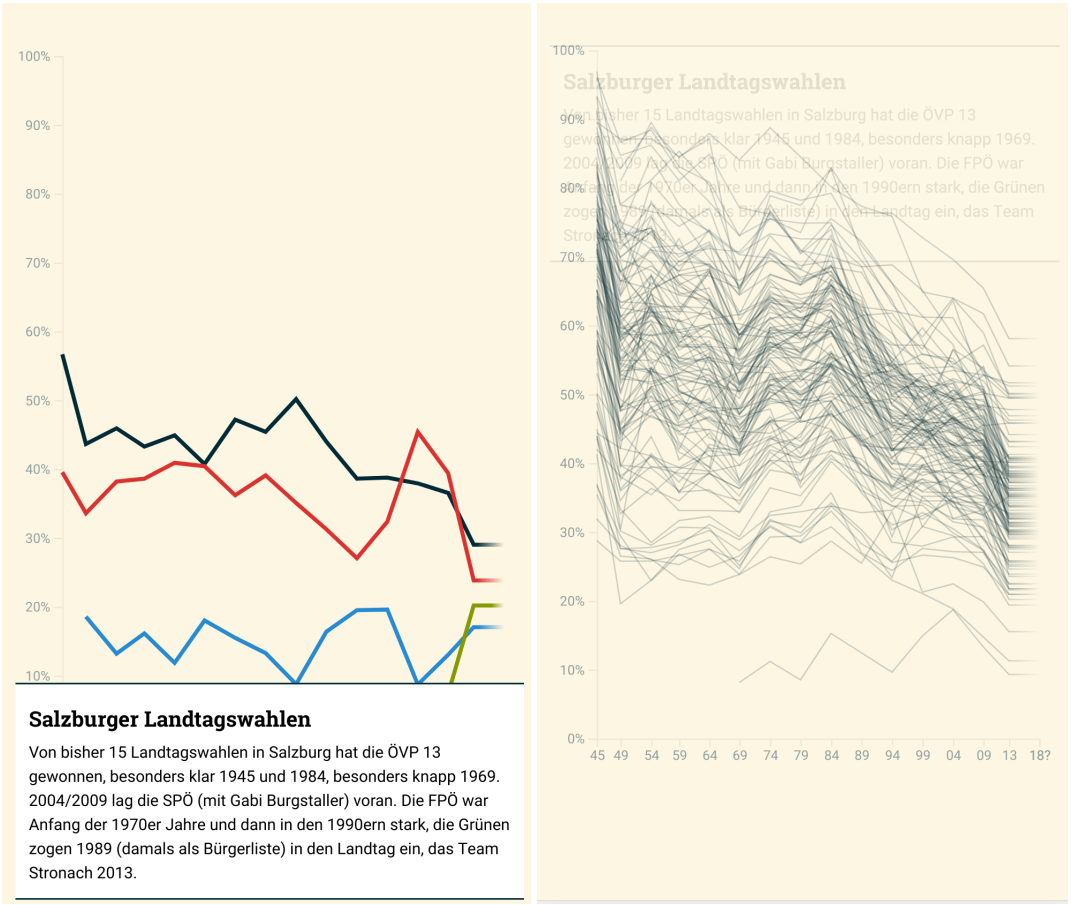


Figure 3.1: Landtagswahlen in Salzburg [Screenshot taken from website by Flooh Perlot.]

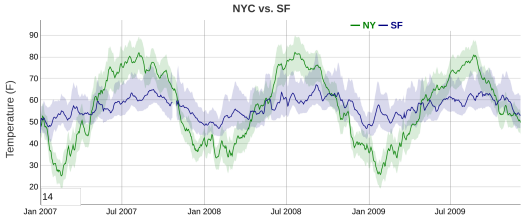


Figure 3.2: dygraphs time series chart [Screenshot taken from dygraphs website.]

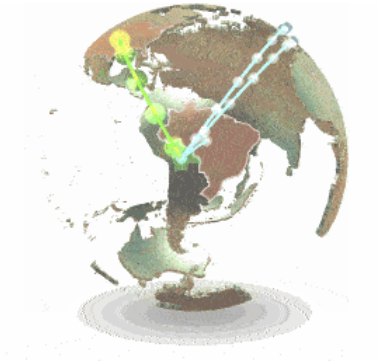


Figure 3.3: Example of 3D Flight Line showing Import/Export quantities Harsojo 2017. [Screenshot taken by the authors.]

```
earthjs.plugins.graticuleSimple = () => {
  const grat = d3.geoGraticule(), $ = {};

  function create() {
      this._.svg.selectAll('.graticule').remove();
      $.grat = this._.svg.append("path").datum(grat).attr("class", "
         graticule");
      refresh.call(this);
  }

  function refresh() {
      $.grat.attr("d", this._.path);
  }

  return {
      name: 'graticuleSimple',
      onCreate()  {create .call(this);},
      onRefresh() {refresh.call(this);}
  }
}

//... plugin in use
const g = earthjs()
.register(earthjs.plugins.graticuleSimple())
.create();
```

Listing 3.1: earthjs usage example

Requirements for running this framework are:

- D3 version 4
- Topojson version 3
- Nodejs web-server or Python simple http server.

3.2 Native Libraries

3.2.1 Core Plot

Core Plot is one of the older chart libraries on iOS. In addition to iOS it also works on macOS and tvOS. Core Plot renders the graphs using Core Graphics and supports Bar Charts, Line Charts, Candlestick Plots, OHLC Plots, Range Plots, Donut Charts, and Pie Charts. It is designed to be very accurate, but simply setting up a simple bar chart takes over 500 lines of code.

3.2.2 ScrollableGraphView

ScrollableGraphView offers nice animations and is very easy to use, but it only supports bar charts and line charts.

To setup a graph you first implement a datasource object

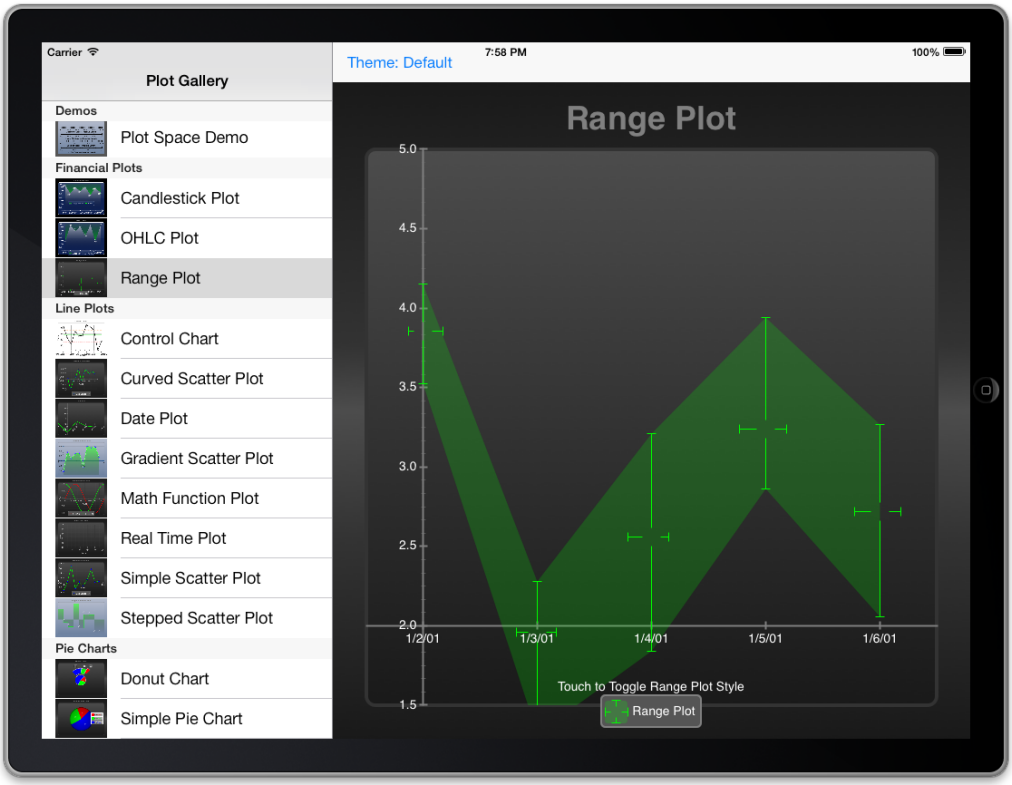


Figure 3.4: Range plot example from coreplot Skroch 2017 [Screenshot taken by the authors.]

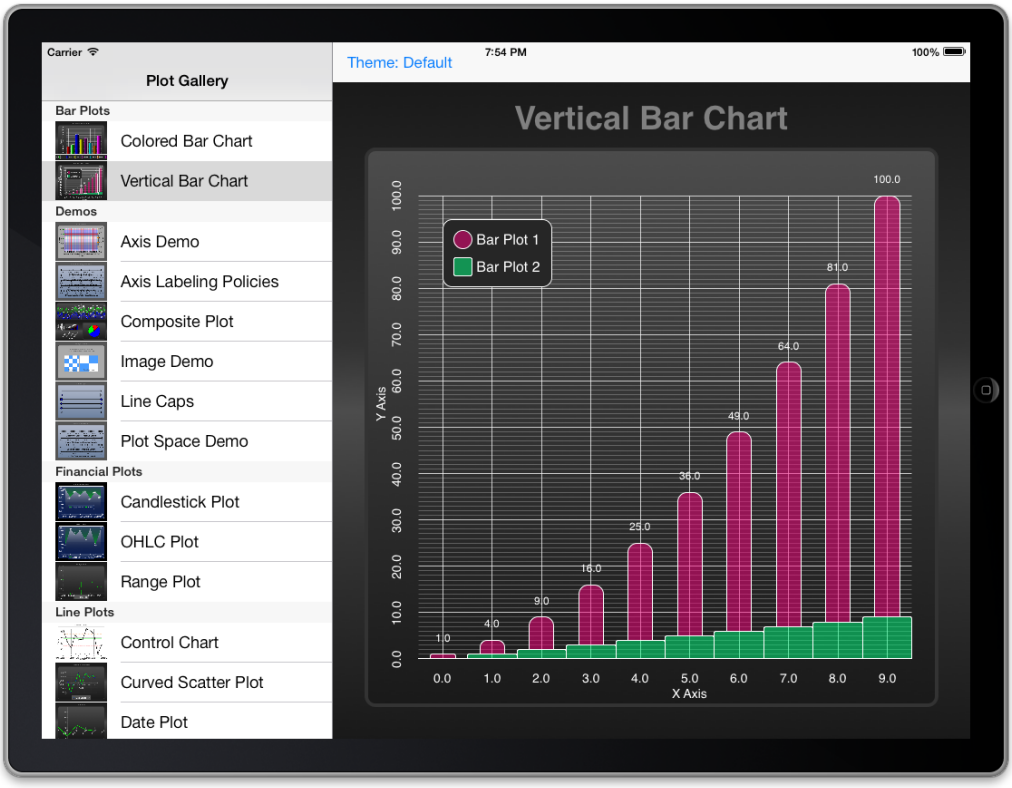


Figure 3.5: Vertical bar chart example from coreplot Skroch 2017 [Screenshot taken by the authors.]

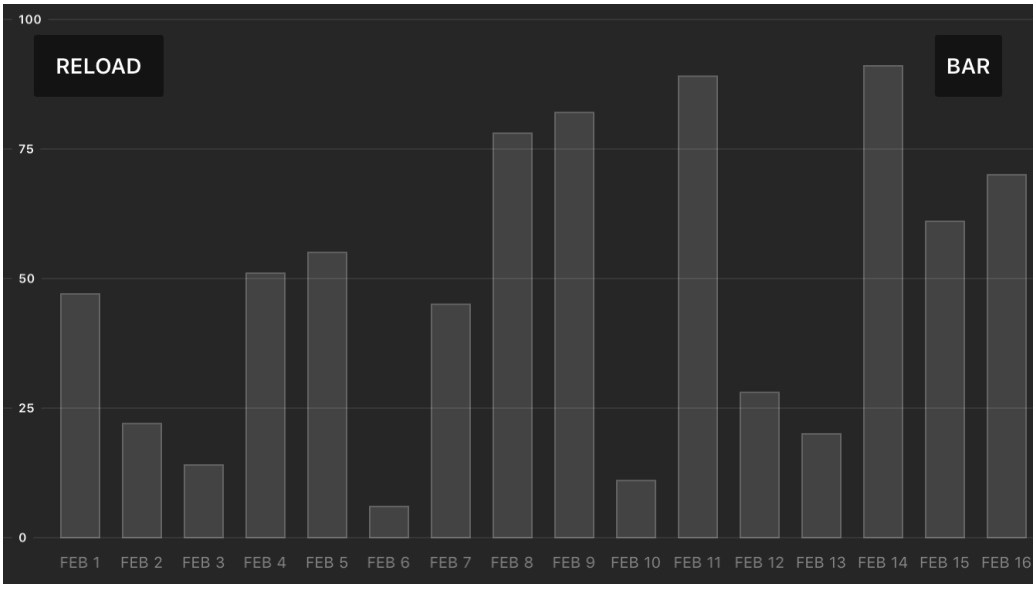


Figure 3.6: A bar chart example from ScrollableGraphView McKenna 2017. [Screenshot taken by the authors.]

```
 func value(forPlot plot: Plot, atIndex pointIndex: Int) -> Double {
    // Return the data for each plot.
    switch(plot.identifier) {
    case "line":
        return linePlotData[pointIndex]
    default:
        return 0
    }
}

func label(atIndex pointIndex: Int) -> String {
    return "FEB \(pointIndex)"
}

func numberOfPoints() -> Int {
    return numberOfDataPointsInGraph
}
```

Listing 3.2: ScrollableGraphView usage example

Once you have a datasource object, you need to create a graph view object and add the plot.

```
let graphView = ScrollableGraphView(frame: frame, dataSource: self)

let linePlot = LinePlot(identifier: "simple") // Identifier should be
   unique for each plot.
let referenceLines = ReferenceLines()

graphView.addPlot(plot: linePlot)
```

Figure 3.7: An example from FSInteractiveMap.

```
graphView.addReferenceLines(referenceLines: referenceLines)
```

Listing 3.3: ScrollableGraphView usage example

3.2.3 FSInteractiveMap

FSInteractiveMap is a simple SVG viewer that can react to touch events. It only supports parts of the SVG spec, but enough to display simple maps with only a couple lines of code

```
[map loadMap:@"world-continents-low" withData:data colorAxis:@[[UIColor
    lightGrayColor], [UIColor darkGrayColor]]];

[map setClickHandler:^(NSString* identifier, CAShapeLayer* layer) {
    // Clicked on identifier
}];
```

Listing 3.4: FSInteractiveMap usage example

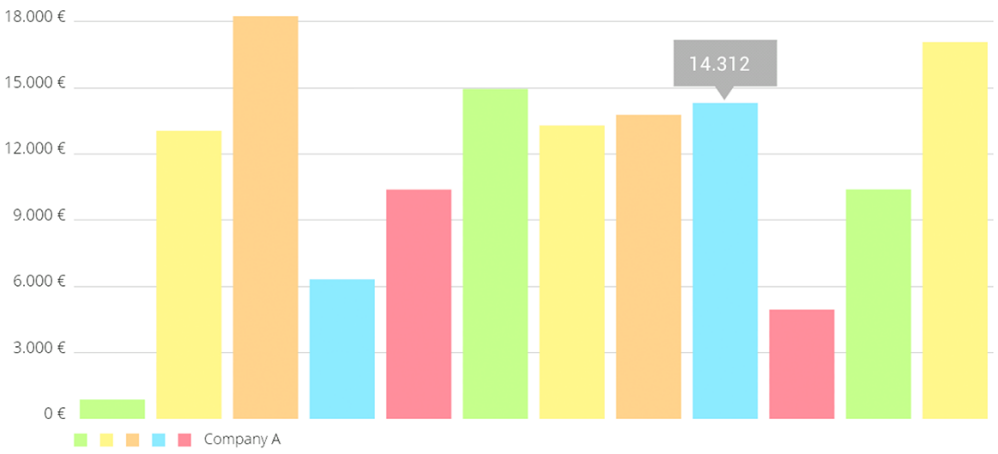


Figure 3.8: An barchart example from MPAndroidChart. [Image is included in the project readme **mpreadme2017**]

3.2.4 MPAndroidChart

It supports line charts, bar charts, pie charts, scatter plots, candle stick charts, bubble charts and radar charts.

To create a bar chart you first create the chart object and add it to the UI

```
// programmatically create a LineChart
  LineChart chart = new LineChart(Context);

  // get a layout defined in xml
  RelativeLayout rl = (RelativeLayout) findViewById(R.id.
     relativeLayout);
  rl.add(chart); // add the programmatically created chart
```

Listing 3.5: Creating and adding the chart object Jahoda 2017a.

After that you simply add the datapoints one by one

```
  for (YourData data : dataObjects) {

    // turn your data into Entry objects
    entries.add(new Entry(data.getValueX(), data.getValueY()));
}
```

Listing 3.6: Adding data to the chart object Jahoda 2017a.

3.2.5 Charts (iOS)

Charts for iOS Jahoda 2017b is a port of MPAndroidChart and uses almost the same API, just modified to work with Swift / Objective-C.

```
let entry1 = BarChartDataEntry(x: 1.0, y: Double(number1.value))
let entry2 = BarChartDataEntry(x: 2.0, y: Double(number2.value))
let entry3 = BarChartDataEntry(x: 3.0, y: Double(number3.value))
let dataSet = BarChartDataSet(values: [entry1, entry2, entry3], label:
    "Widgets Type")
let data = BarChartData(dataSets: [dataSet])
let data = PieChartData(dataSet: dataSet)
pieChart.data = data
pieChart.chartDescription?.text = "Share of Widgets by Type"
```

Listing 3.7: Charts usage example

3.3 Comparison

NAME	LICENSE	PLATFORM	EASE OF USE
Core Plot	BSD License	iOS, Mac, tvOS	Difficult
ScrollableGraphView	MIT License	iOS	Easy
FSInteractiveMap	Apache License 2.0	iOS	Easy
MPAndroidChart	Apache License 2.0	Android	Easy
Charts	Apache License 2.0	iOS	Easy

Table 3.1: Comparison among the mentioned native libraries

Chapter 4

Design Tips

4.1 Design Mobile First

When designing for multiple screen sizes, it's best to start designing for mobile and small screens first to avoid Information overload.

Once you have designs for small screens you can continue with adding more information for larger screens, since mobile covers a lot of different screen sizes.

4.2 Design the way people hold their mobiles

People hold their phones 99% of the time vertically (Sparkes 2016) and people have no problem with scrolling. So design vertically. Libraries like graph-scroll help a lot when designing vertical visualisations with scroll interactions.

4.3 Use gestures that are expected

Gestures are a great way to add additional features to touch UIs. However most gestures are hard to find, and people generally do not search to find gestures. Two exceptions are the swipe gesture to scroll content, especially if the content is cut off, to indicate scrollability, and the pinch to zoom gesture.

4.4 Hover States

Touch Screen do not support hover states, and you can not expect people to tap too much. If you have important information do not hide it in tooltips, instead think about what information is important and always show the important information without any extra actions.

4.5 Mobile Context

Remember that mobile is not only about the device, but also about the context it's used in. People may use the app on the go, while having to watch out for traffic, so don't expect a long attention span. They might use the app outside, with the sun (or other light sources) reflecting in the screen, so don't expect them to notice small differences in colors and use higher contrasts.

Chapter 5

Design Examples

5.1 Responsive Example

5.1.1 New York Times Map

The New York times and other newspapers use SVGs for their information graphics. For the following example they use a toolkit called ai2html that turn Adobe Illustrator files into a combination of SVG and HTML code, that work for different screen sizes. Cedric 2018

Figure 5.1: New York Times Map Example: top side Desktop, bottom side mobile. [Screenshot taken from New York Times website.]

5.2 Vizable

Visable is a tool for iOS platform developed by Tableau that allow users to transform data into interactive charts and graphs by importing from files, services or cloud. One of the other features is the sharing the data stories within static image or interactive VIZ file. Interface itself is pretty simple - there are no configuration screens, wizards, or chart builders. Visable provides a lot of interactive functions, optimized for touchscreen, which allow user fully control the process of building data visualizations:

- Adding columns by pinching out to add or pinching in to remove
- Changing calculations by flick
- Selecting the intervals by tapping and dragging
- Changing required fields by swiping them left and right
- Filtering categories by swiping to left to remove and right to focus

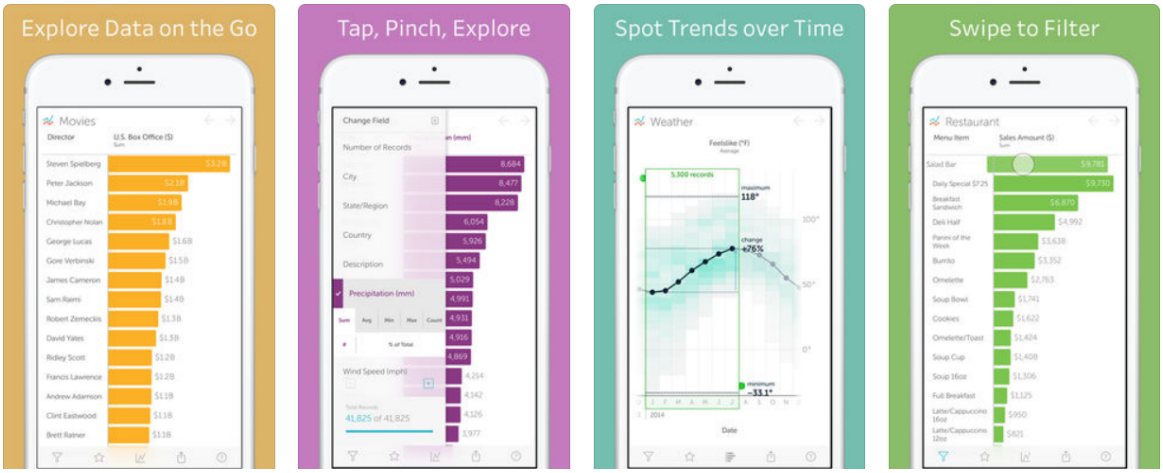


Figure 5.2: Examples of possible data visualizations and controls. [Screenshot taken from by the authors.]

5.3 Hierarchy Visualization

Hierarchy visualization (or tree visualization) targets the representation of connected, acyclic graphs Schulz 2011. Examples for such visualizations include the visualization of disk space usage, file systems, etc. Hierarchies are mostly represented as lists in mobile devices where the view corresponds to a certain node and the options of navigation are limited to going back to the parent node view or selecting one child, that means no direct connection to the siblings, and no appearance of descendants rather than the direct children, e.g. files explorers, media navigators.

5.3.1 Voronoi tree map

Voronoi tree maps idea is the space-filling recursive subdivision of a given area without producing holes or overlappings. It offers low aspect ratios, better interpretability of hierarchical structures, and flexible adaptability regarding the enclosing shape. Balzer and Deussen 2005 The hierarchy is assumed to be attributed, which means that each node in the hierarchy has a value represented by its size on the tree map. Balzer and Deussen 2005 We assume that this attribute needs to be aggregated from the sub-tree of the corresponding node.

A practical example comes from FoamTree, a JavaScript tree map visualization with innovative layout algorithms and animations. It works both on desktop and mobile and it supports the touch interactions (tapping, scaling, panning). Carrot Search 2012

The advantage of such a design is that it uses up the whole screen space and allows the user to navigate conveniently between nodes using the intuitive gestures such as pinching for zooming in or out and tapping to focus on a certain node or to dive a step down the tree.

Figure 5.3 shows the FoamTree mobile demo in a web browser on a mobile phone presenting a hierarchy for some data mining topics. The figure to the left shows an overall view of the data, while the figure to the right shows the visualization after focusing on 'Data Mining Tools' node where the node takes up as much from the screen size as possible, gets a bit magnified with respect to the other nodes and its children become more obvious.

5.3.2 Sunburst

According to The Data Visualisation Catalogue no date , sunburst visualisation shows hierarchy through a series of rings, that are sliced for each category node. Each ring corresponds to a level in the hierarchy, with the central circle representing the root node and the hierarchy moving outwards from it.

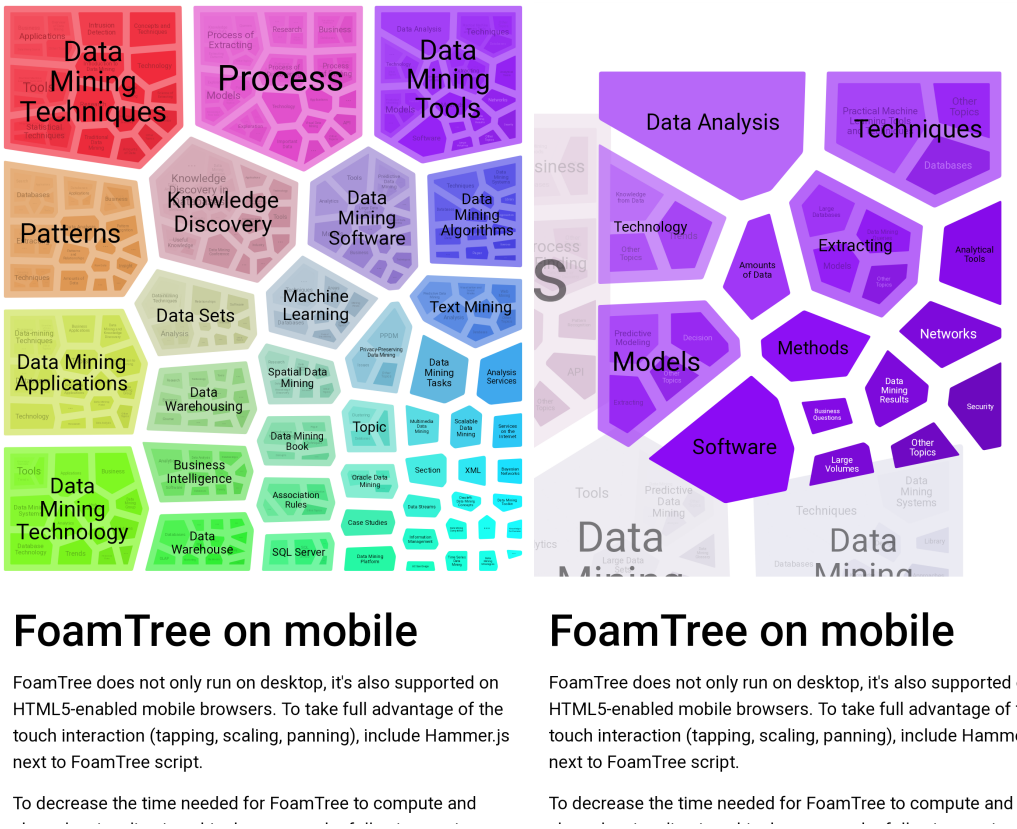


Figure 5.3: FoamTree Voronoi treemap visualization example on mobile Carrot Search 2012. [Screenshot taken from the demo, FoamTree on mobile.]

For sunburst chart, we include an demo from Highcharts Highsoft 2009, a SVG-based, multi-platform charting library used to add interactive, mobile-optimized charts to web and mobile projects. The example in figure 5.4 shows a sunburst visualization of the world population in 2017 divided level by level into continents, regions and countries.

The navigation options here include drilling down a node by tapping on it, so the sunburst view now corresponds to its subtree and the title of the chart will be the label of this node. Tapping on that title or on the root node in the center rolls up the view and climbs one level up the tree.

The problem with this type of visualization is that it leaves holes since the whole shape is circular on a rectangular screen. Another problem is the labelling of each node given that the screen size is small and so are the nodes.

A workaround would be including only a limited number of levels in the view, e.g. 3 levels would be convenient enough (the parent node in the center, the siblings in the second level, and their children in the third one). Given the navigation options, whenever we drill down a node, it becomes the parent (in the center) and the same mentioned pattern applies for it. This can provide good local views of the tree, and enable better labelling. Figure 5.5 shows the same visualization after drilling down to the node 'Europe'. It shows the title as mentioned 'Europe' and the labelling seems less of a problem because of the bigger sizes for nodes. Still there is so much space that should be used.

5.3.3 Enhanced Radial Edgeless Tree (ERELT)

Introduced in Chhetri et al. 2015, ERELT offers a clear visualization of any tree structure with intuitive interaction. It is another space-filling hierarchy browser which has no holes or overlaps between nodes.

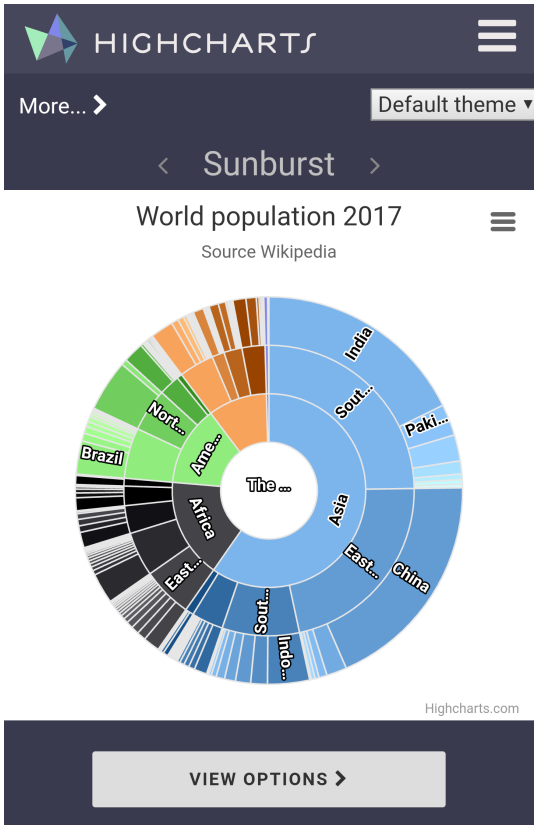


Figure 5.4: Highcharts sunburst visualization example on mobile showing the world population Highsoft 2009. [Screenshot taken from Highcharts sunburst demo.]

The screen is divided into rectangles (one inside the other). The innermost rectangle lies in the top-left corner and represents the root of the subtree presented at the moment. The next level in the subtree lies in the area between the innermost rectangle and the one containing it, which is also aligned to the top-left corner and so on. Each level is divided with radial lines starting from top-left corner. Further design details were discussed in Chhetri et al. 2015 in order to make the view more optimized and convenient.

An example is shown in figure 5.6, where we can see how the screen is divided and how each level is divided. We may notice the that the text orientation is aligned with the radial lines emitted from the top-left corner, which gives an optimized space for labelling and also a feeling that the contents originate from the root node.

The number of levels appearing on the screen adapts to the screen size. Besides, the number of nodes appearing in a single level adapts to the screen size. Moreover, a branching limit is set so that no node shows more than a certain number of direct descendants, e.g. 3 children. However, this may leave some nodes of a shown level hidden although their siblings are not. Whenever this happens, arrows appear on the corresponding nodes, indicating that we can scroll in the direction of the arrow to continue viewing the hidden nodes which, by this gestures, replace the shown nodes. This is in addition to the drill-down and roll-up operations, that were mentioned in 5.3.2 and were applied in the same manner in this visualization.

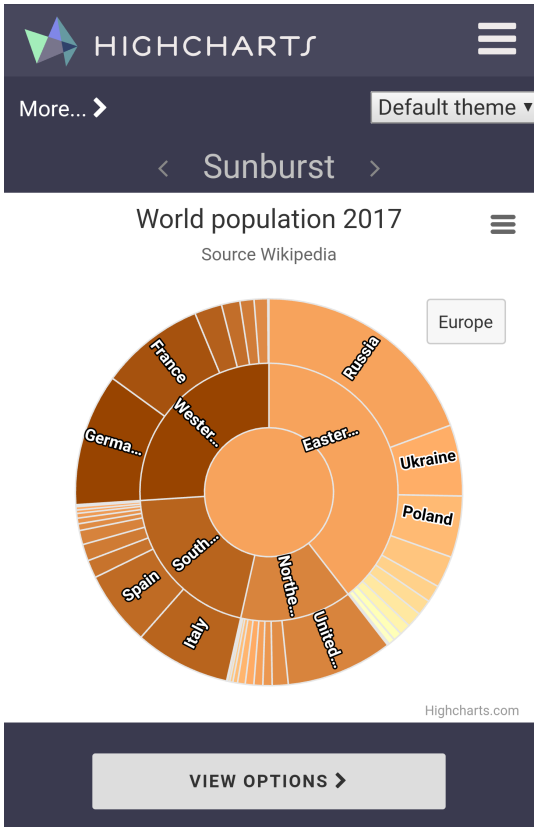


Figure 5.5: Highcharts sunburst visualization example on mobile showing the population of Europe in the context of the world population Highsoft 2009. [Screenshot taken from Highcharts sunburst demo.]

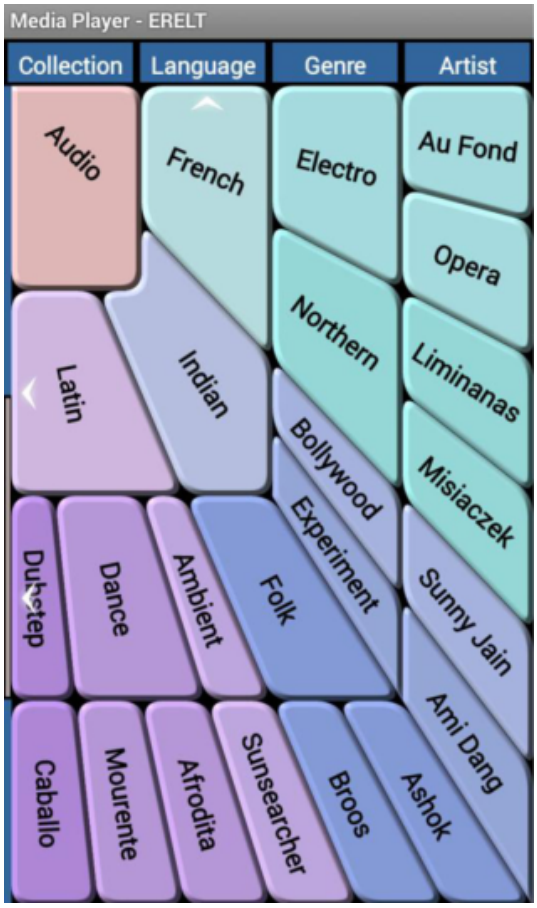


Figure 5.6: ERELT visualization of a sample music library segment (language, genre and artist levels) Chhetri et al. 2015. The white arrows indicate the existence of hidden nodes in the corresponding levels. Sliding such levels result in viewing the hidden nodes. [Screenshot taken by the authors.]

5.4 Parallel Coordinates

Parallel coordinates visualization is a method to visualize multidimensional numerical data. It uses equidistant parallel vertical lines which represent the axes of a multidimensional space (one vertical line for each dimension). Each object is plotted as a polyline defined by values along each dimension. **InfoVisNotes**

With these plots, we can recognize patterns in data, and analyze the behavior of a subset of data that satisfy a certain predicate.

When showing such visualization on a small screen, only a small number of axes can be easily mapped and navigated, so an idea would be to adapt the number of shown axes to the screen size and try to show the most relevant dimensions by doing so while giving the user the ability to choose the axes and their number at the same time.

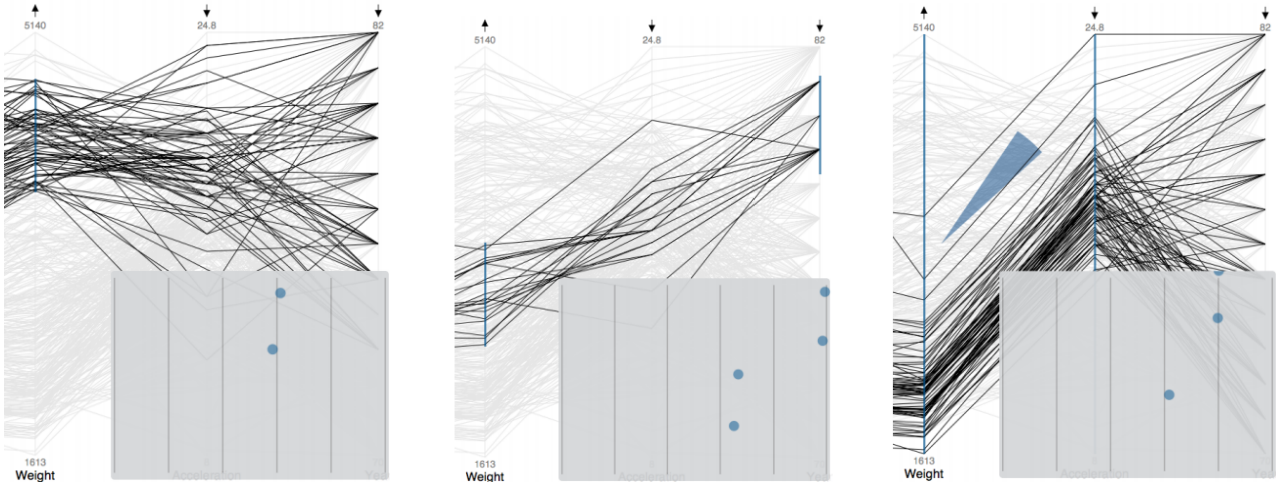
We include an example for parallel coordinates visualization on mobile devices, which makes use of the multi-touch feature in most modern touch devices. Kosara 2011 It deals specifically with the brushing operation, which usually means selecting a range of values on one or more axes Inselberg 2009. In addition, there is the angular brushing operation in which the user selects a sub-set of slopes between two axes and the focus then turns to the objects whose slope between the corresponding axes belongs to the selected subset Hauser et al. 2002.

Next, we will discuss the three types of brushing included in Kosara 2011. The authors use a separated multi-touch trackpad rather than the screen itself so that the fingers don't cover the view. However, we may assume that such design option can be applied only with the touch screen without the trackpad. These examples makes more sense to hold the device in a horizontal orientation.

Brush on One Axis After selecting a certain axis, one can use two fingers to specify a range on that axis, and the brush operation will be interactively conducted upon that. The example for that is shown in

Brush on Two Axes This can be viewed as an extension to applying brush on one axis. Selecting two ranges on two different axes requires two fingers for each axis (four in total). The intersection of the two brushing operations on both axis is the result. The selected axis do not have to be adjacent.

Angular Brushing This operation requires three fingers, one finger on a certain axis, and the two others on an **adjacent** one. The absolute positions of the fingers does not matter here. What specifies the selection is their relative position, namely the range of slopes that lies between the two slopes from the single fingertip on one axis to both of the other two fingertips on the other axis. This allows to brush different kinds of correlations since it relies on brushing by direction rather than location Kosara 2011.



(a) One-axis brushing with two fingers.

(b) Two-axis brushing with four fingers.

(c) Angular brushing with three fingers.

Figure 5.7: Parallel coordinates brushing examples. The pad in the bottom-right corner represents the fingertips on the trackpad Kosara 2011. [Screenshots are taken by the authors]

5.5 Interactive scatter plot

A scatter plot shows the distribution of data points on a two-dimensional Cartesian plane based on two numeric attributes. It helps in recognizing correlations between data objects with respect to the corresponding attributes. The idea can be generalized to include not only one variable. For example the colors and the shapes of the points may represent other (probably non-numeric attributes).

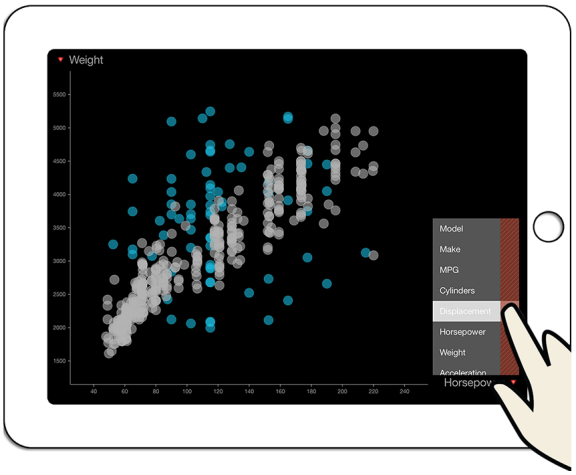
The usual interaction for such plots on touch devices is to make additional information pop up upon tapping on a certain point. This sort of interaction could be seen as inconvenient since it covers other data points until the user chooses to defocus or after a certain amount of time. The reason for this solution could be the attempt to find an alternative for the hoover status which is not supported by touch devices in general.

However, many other design and interaction alternatives can be applied to make scatter plots easily explorable on touch devices of different sizes. Here we look at an example implemented in Sadana and Stasko 2014 which includes a variety of ways to manipulate scatter plots on a touch device.

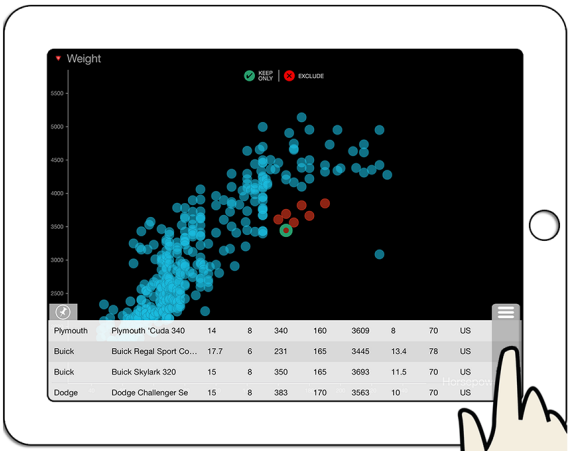
Choosing the attributes is up to the user where they can even see both plots (with two different choices of a certain attribute) at the same time giving two different colors for the two plots while choosing. Figure 5.8a shows the process of attribute selection.

Selecting a subset of the data points can be also touch friendly whether it was free drawing of a closed shape to specify the region of interest or using orthogonal selection. The selected data can then be filtered out or retained. There is also a choice to give more detailed information about the selected subset i.e. with respect to the attributes not included in the plot. This is shown as tabular information; see figure 5.8b.

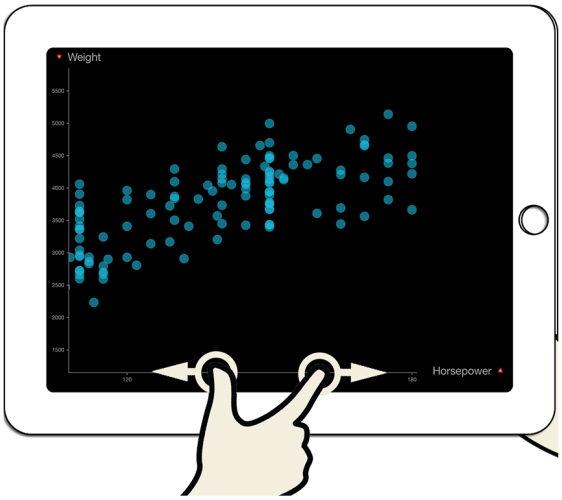
Zooming can be done on a single axis pinching on it (figure 5.8c), which could be viewed as more accurate, as well as using intuitive gestures such as double tapping somewhere on the plane. Besides, one can use zooming lens to gain local zooming by pinching on the corresponding region and can control how much it should be magnified (figure 5.8d).



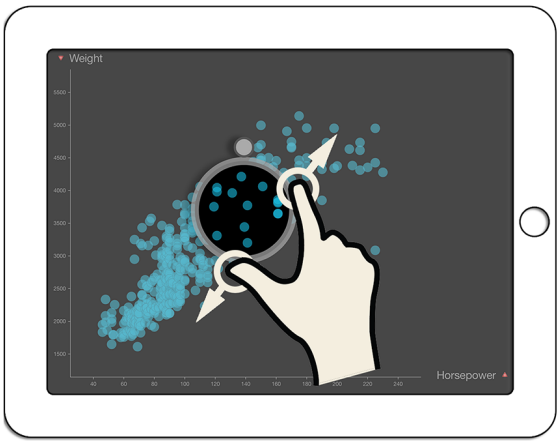
(a) Choosing the attributes.



(b) Detailed tabular information.



(c) Zooming on a single axis.



(d) Zoom lens.

Figure 5.8: Touch interactions with a scatter plot Sadana and Stasko 2014. [Pictures are designed by the authors.]

Bibliography

Balzer, Michael and Oliver Deussen [2005]. "Voronoi Treemaps". *Proceedings - IEEE Symposium on Information Visualization, INFO VIS* InfoVis 05 [2005], pages 49–56. ISSN 1522404X. doi:10.1109/INFVIS.2005.1532128 (cited on page 19).

Carrot Search [2012]. *FoamTree 3.4.5*. 2012. https://carrotsearch.com/foamtree/ (cited on pages 19–20).

Cedric, Sam [2018]. "Ai2html and Its Impact on the News Graphics Industry". In: (Montréal, Canada). Apr 2018. https://mobilevis.github.io/assets/mobilevis2018_paper_20.pdf (cited on page 17).

Chhetri, Abhishek P., Kang Zhang and Eakta Jain [2015]. "A mobile interface for navigating hierarchical information space". *Journal of Visual Languages and Computing* 31 [2015], pages 48–69. ISSN 1045926X. doi:10.1016/j.jvlc.2015.10.002. http://dx.doi.org/10.1016/j.jvlc.2015.10.002 (cited on pages 20–21, 23).

Harsojo, Widi [2017]. *Earth JS*. 2017. https://github.com/earthjs/earthjs (cited on page 8).

Hauser, Helwig, Florian Ledermann and Helmut Doleisch [2002]. "Angular brushing of extended parallel coordinates". In: *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*. IEEE. 2002, pages 127–130 (cited on page 24).

Highsoft [2009]. *Highcharts*. 2009. https://www.highcharts.com/products/highcharts/ (cited on pages 20–22).

Inselberg, Alfred [2009]. "Parallel coordinates". In: *Encyclopedia of Database Systems*. Springer, 2009, pages 2018–2024 (cited on page 24).

Jahoda, Philipp [2017a]. *MPAndroidChart Getting Started*. 2017. https://github.com/PhilJay/MPAndroidChart/wiki/Getting-Started (cited on pages vii, 13).

Jahoda, Philipp [2017b]. *MPAndroidChart readme*. 2017. https://github.com/PhilJay/MPAndroidChart (cited on page 13).

Kosara, Robert [2011]. "Indirect multi-touch interaction for brushing in parallel coordinates". *Visualization and Data Analysis 2011* 7868 [2011], pages 786809–786809–7. ISSN 0277-786X. doi:10.1117/12.872645. http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.872645 (cited on pages 24–25).

McKenna, Phillip [2017]. *ScrollableGraphView*. 2017. https://github.com/philackm/ScrollableGraphView (cited on page 11).

Sadana, Ramik and John Stasko [2014]. "Designing and implementing an interactive scatterplot visualization for a tablet computer". *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces - AVI '14* [2014], pages 265–272. doi:10.1145/2598153.2598163. http://dl.acm.org/citation.cfm?id=2598153.2598163 (cited on pages 25–26).

Schulz, Hans Jörg [2011]. "Treevis.net: A tree visualization reference". *IEEE Computer Graphics and Applications* 31.6 [2011], pages 11–15. ISSN 02721716. doi:10.1109/MCG.2011.103 (cited on page 19).

Skroch, Eric [2017]. *CorePlot Example Graphs*. 2017. https://github.com/core-plot/core-plot/wiki/Example-Graphs (cited on page 10).

Sparkes, Sophie [2016]. *My top 5 mobile design tips*. 2016. http://www.datasparkes.com/data-sparkes/2016/8/22/my-top-mobile-design-tips (cited on page 15).

The Data Visualisation Catalogue. *Sunburst Diagram*. https://datavizcatalogue.com/methods/sunburst%7B%5C_%7Ddiagram.html (cited on page 19).

Wikipedia [2018]. *Responsive web design*. 1st Jun 2018. https://en.wikipedia.org/wiki/Responsive_web_design (cited on page 1).