# Designing and Implementing an Interactive Scatterplot Visualization for a Tablet Computer

Ramik Sadana
Georgia Institute of Technology
ramik@gatech.edu

John Stasko
Georgia Institute of Technology
stasko@cc.gatech.edu

## ABSTRACT

Tablet computers now offer screen sizes and computing capabilities that are competitive with traditional desktop PCs. Their popularity has grown tremendously, but we are just beginning to see information visualization applications designed for this platform. One potential reason for this limited development is the challenge of designing and implementing a multi-touch interface for visualizations on mobile, tablet devices. In this work, we identify the primary challenges that touch screen interactions pose for information visualization applications. We explore the design space of multi-touch interactions for visualizations and present a prototype information visualization application using a specific technique, a dynamic scatterplot, for an iPad.

## Categories and Subject Descriptors

H.5.0 [**Information Interfaces and Presentation**]: General

## General Terms

Design, Algorithms, Human Factors.

## Keywords

Information visualization, multi-touch interaction, scatterplot, tablet computer, gesture.

## 1. INTRODUCTION

The popularity of tablet computers has grown tremendously in recent years. Devices such as the Apple iPad, Amazon Kindle, and Microsoft Surface constitute a significant portion of computer sales today and a wide variety of applications have been tailored to tablet platform. However, one area with just a few initial applications for tablets is information visualization. Though popular commercial systems such as Tableau and Spotfire have introduced tablet versions in the last two years, these systems still feel much like a port of a desktop application. They do not yet leverage the set of gestural interactions that touch-based interfaces provide and do not offer a rich, multi-touch interface in the style of other tablet applications.

We speculate that the limited development of information visualization applications for tablets has resulted from the challenge of designing and implementing the interface. First, most tablets still provide a smaller screen size, which is a limitation for data visualization. Perhaps more importantly, information visualization applications generally have many small visual objects to select and manipulate, and they contain many interactive widgets such as buttons, sliders, menus, and dialog boxes. Desktop information visualization applications typically make extensive use of a WIMP (window, icon, menu, pointer) interface. Translating visualization interfaces and interactive operations to a finger-directed multi-touch interface without a keyboard and mouse is simply a very challenging problem [18].

Some recent research has begun to address this challenge, however. One particular avenue of research has been the development of multi-user, multi-touch data visualization applications for tabletop computers [12], [14]. These systems have explored the design space for multi-touch interaction, but developing for tablet computers is subtly different than for large tabletop computers. Baur et al. [2] recently designed a multi-touch tablet interface for a StreamGraph-style information visualization. This application is much more aligned with what we seek to create. Our specific focus, however, is on a more general, widely used visualization technique. Rzeszotarski and Kittur's [24] tablet application is a similar technique to ours but focuses on physics-driven interactions.

Rather than viewing the movement of information visualization applications to multi-touch tablets as a simple translation or software port, we see this problem as an interesting design challenge. Initial research [25] has noted the richness of the interaction design space – for any interactive operation in an information visualization system, multiple multi-touch implementations make sense and appear totally reasonable. Furthermore, we will show that adopting current multi-touch gestures from tablet applications presents complications when applied to data visualizations.

In this work, we further explore this design space and implement a working information visualization application for a tablet PC. We chose a dynamic scatterplot visualization as the technique of our focus. Scatterplots are well-known and pervasive in information visualization; many systems include them in their suite of provided techniques. Scatterplots also have a long history in data visualization, including the pioneering Starfield-display FilmFinder system [1] that led to the development of Spotfire. Our project starts from "ground zero", that is, we are not porting an existing system. Our application was developed to take full advantage of, and to address all the limitations and constraints of, a tablet interface using only fingers and touch for interaction.

The contributions of our work are multiple and align with the sections of this article. By studying the Tableau and Spotfire commercial implementations of a dynamic scatterplot, we derive a set of tasks and interactive operations that our tablet application would support (Section 3). Next, we analyze the characteristics of the tasks and describe how those characteristics will influence design (Section 4). We ultimately explore the design space of multi-touch operations for each task, explain the nuances required for each, propose multiple potential gestures, and implement many of the options in a prototype interactive scatterplot application for the iPad (Section 5). Because the static nature of a paper may not convey all the operations well, we include a video overview summarizing the application with this submission.


*AVI' 14*, May 27 - 29 2014, Como, Italy

ACM 978-1-4503-2775-6/14/05…$15.00.

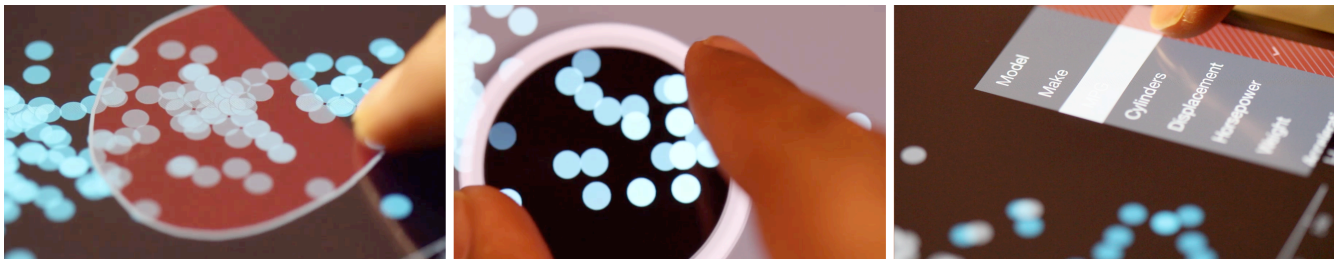


**Figure 1. Interactive Scatterplot prototype for an Apple iPad.**

## 2. RELATED WORK

Early data visualization research for computers other than desktop PCs includes systems designed for mobile devices. Buering et al. [6] presented a zoomable user interface (ZUI) based scatterplot visualization for a PDA that used a stylus for input. In a user study comparing two scatterplot applications, one displaying both a detail view and an overview and the other displaying only the detail view, participants solved search tasks on a phone display faster without the overview scatterplot. While mobile phones present similar design issues as tablets, a stylus for input is substantially different than touch.

As multitouch devices became more common, a variety of research emerged for visualizations on touch-based tabletops that especially focused on the collaboration aspects. Cambiera [14] is a search result visualization system supporting collaborative brushing and linking of search results. Heilig et al. describe ScatterTouch [12], a two-dimensional scatterplot visualization technique that used the concept of multi-focus regions to support simultaneous, multi-user interactions. North et al. [22] compared how people manipulate node-link diagrams on a touch-based tabletop and a mouse-based desktop.

Frisch et al.'s study [9], somewhat similar to our research, uses the guessability study approach [30] to obtain user-elicited pen and touch gestures for manipulating node-link diagrams on a tabletop. The study reports that the user-defined gesture set contained many ambiguities. Our own experiments employing the guessability approach for scatterplots on tablets generated similarly ambiguous and non-novel results. Hinckley et al. [13] called this an expected behavior, citing lack of any experience as the reason why users have difficulty envisioning gestures and interactions.

SketchVis is a system that allows users to sketch visualizations on a whiteboard [5]. The user draws representative visualization constituents such as the coordinate system, labels on axes, and some initial glyphs; the system responds by filling in the chart correspondingly. Walny et al. [29] investigated the use of pen and touch for data exploration on whiteboards. Their study explored the interaction styles people employ, in particular the role of pen and touch for representations and linking. While there are some similarities between their work and ours, their major focus is on the creative aspects of visualization, while ours is on the interaction aspects of the data visualizations.

Within information visualization, only a few initial systems have focused on small-to-medium size screens with touch as the mode of interaction. For work particular to tablets, Baur et al. [2] examined the stacked graph for their TouchWave system, and developed a new set of multi-touch gestures for scaling, scrolling, providing context, extracting layers, and many other activities. They noted that developing a consistent interaction set was one of the primary challenges of their project. Rzeszotarski and Kittur [24] present TouchViz, a scatterplot visualization system for tablets that employs zoom lens and razor filter interactions. The authors enhance the playfulness of the interface using physics-based interactions and gravity. Their focus is more on exploring this metaphor rather than considering the broader design space of multi-touch interactions. It is not clear how well their interactions would map to other visualization techniques.

Recently, there has been a growing interest in understanding interaction models for visualizations on touch-screen interfaces. Lee et al. [18] survey multiple forms of post-mouse/keyboard interaction, such as touch, gesture, speech, and whole body engagement. They specify the interaction design considerations for the new modalities, eliciting four principle dimensions: the individual, the technology, social aspects of interactions between people, and the interspace between a person and the technology. They specifically identify "going beyond mouse and keyboard" as an opportunity and a topic worthy of further research.

Isenberg et al. [15] discuss the visualization space specifically for touch-based surfaces, highlighting the technical, design and social challenges in supporting visualization on touch devices. For touch interaction in particular, they outline the creation of a gesture vocabulary that is both global to various visualization types and low in complexity as a central research topic. Jansen and Dragicevic [16] describe a modification of the infovis pipeline to accommodate beyond-the-desktop visualization systems. They suggest unifying the infovis pipeline and instrumental-interaction model [3] for post-WIMP interfaces that include touch-based interfaces as well as physical, fabricated visualizations.

## 3. SCATTERPLOT VISUALIZATIONS

We began our research broadly and set out to design an intuitive, powerful, and useful suite of touch-based interactive operations for data visualization. We wanted to identify interactions that would be effective across a variety of visualization techniques. Very quickly, however, we realized that beginning with such a broad focus made the problem unwieldy. Instead, we decided to focus on a particular visualization technique and explore it in depth. We believed that the knowledge gained from this study would be valuable toward the larger, more general problem.

A multitude of visualization techniques can be used for representing data [11]. However, a few canonical examples, such as line charts, bar charts and scatterplots, are widely used across many different visualization systems. For this study, we chose to focus on scatterplots. A scatterplot primarily consists of two axes and glyphs that represent the data points. Attributes of the data encoded on the $x$ and $y$ axes typically are quantitative in nature, and nominal data attributes are often encoded using visual properties of the glyphs such as color, shape, and size.

A salient feature of a scatterplot's representation is that it displays every data item in the view individually. As a result,

scatterplots excel at highlighting clusters, outliers, and trends. Decreasing the size of the glyphs and employing opacity to manage overlap helps achieve a high data density. Small glyph sizes work reasonably well for mouse-based interaction because a precise cursor is able to address and identify individual glyphs. However, this precise resolution is not feasible in the case of finger-based interaction (the so-called "fat finger" problem [26]), which is one of the primary challenges for scatterplots on tablets. Since this problem arises for many other visualization techniques, we believe that our work will be applicable for those techniques as well.

To help construct a feature-complete implementation of scatterplots on a tablet, we examined two widely used visualization systems available on desktop computers: Spotfire and Tableau. Both systems provide a powerful, comprehensive scatterplot implementation. The purpose of this examination was to identify a set of interactive tasks/operations that support exploration with scatterplots. Analyzing the two systems generated a list of 35 tasks. We further categorized each task using the visualization interaction intent framework [31]. This categorization helped us to ensure that most aspects of interaction are covered and to minimize redundancy across the tasks.

We next pruned the list of tasks to a more concise set containing what we call "primary tasks." Primary tasks are those that we consider central to data exploration with scatterplots. We included tasks fundamental to interactive visualization such as selection and filtering. We did not include others that, while useful for some analysis, clearly are not used as commonly, such as showing trend lines and highlighting clusters. We excluded other operations such as "swap variables on axes" that could be achieved using a short set of primary operations. Finally, we did not implement system-related commands such as printing, saving snapshots, changing chart background color, etc., that would be needed for a commercial tool, but are not necessary for a research prototype. The resulting set of primary tasks is shown in Table 1.

**Table 1. Primary Interaction Tasks and their categorization**

| Task | Interaction Intent [31] | Categorization |
|---|---|---|
| Assign *x* and *y* | Encode | Data-centric, Essential |
| Assign color | Encode | Data-centric, on-demand |
| Assign size | Encode | Data-centric, on-demand |
| Find detail | Abstract/Elaborate | View-driven |
| Select | Select | View-driven |
| Zoom | Abstract/Elaborate | View-driven |
| Filter on points | Filter | View-driven |
| Filter on values | Filter | Data-centric, on-demand |
| Change axis scale | Reconfigure | Data-centric, on-demand |

# 4. CLASSIFYING INTERACTIONS

In this section, we explore the design space of interactions for the set of primary tasks we identified for scatterplots. We quickly learned that this design space is much larger than one might expect. This section provides a sense of that breadth, and the subsequent section presents the ultimate design choices we made.

Previous work in the area of touch-screen interaction includes various interaction taxonomies. In our work, we primarily consider two different types of interactions: WIMP-derived and gesture-driven. One initial, natural way to develop tablet interfaces is to simply port desktop counterparts consisting of WIMP elements such as menus, buttons, and toolbars onto the touch screens. These elements leverage users' familiarity and allow for easy discovery of features. However, they occupy valuable screen space and sometimes felt less than optimal for these types of devices.

Over time, as touch screen technology evolved to provide faster response times and support multi-touch interaction, user interfaces began leveraging the strengths of gesture-driven interactions. Gesture interactions provide direct access to the objects of interest [15] and free up screen space previously utilized by WIMP elements. Conversely, gestures do not support feature discovery or learnability as well as WIMP elements.

Drucker et. al. [7] highlight the difference between gesture-driven and WIMP-driven interaction for information visualization applications. They compared two different interfaces that use a barchart to present the same data. Both interfaces provided the same functionality. The first interface (WIMP) used interface elements such as menus and buttons for interaction while the second interface (FLUID) used gestures. In their experiment, participants performed a series of tasks using both interfaces. Results showed that the FLUID interface performed better than WIMP. Participants were faster at performing tasks using gestures and also demonstrated fewer errors. A majority of participants also expressed preference for the gesture interface.

Although the study offers strong evidence for gestures being more effective for bar chart interactions than WIMP on a tablet, we have some reservations about generalizing the results for scatterplots and other techniques. First, the study employed a small number of operations. This reduced the potential for any conflict to arise between different interactions. Second, bars in a barchart are inherently easier to touch than scatterplot glyphs given their larger size. Thus, tasks such as selection and filter are considerably simpler. Finally, the primary aim of the study was to compare representative gesture and WIMP interfaces and not to find the most effective gestures for interacting with the technique.

For our application, we aim to develop a suite of effective interactions that comprehensively support data exploration on scatterplots. In the previous section, we presented a list of nine primary tasks to be supported. We believe that using only gestures for all these tasks would inherently be a weak solution. A large number of gestures would introduce issues of gesture conflicts, learnability, and discoverability. Moreover, operations such as changing the axis attributes cannot be supported effectively using only gestures. We believe that a useful solution will, instead, be one that uses a combination of gestures and WIMP elements.

To develop interactive operations for each task, we followed a two-step approach:

1. We differentiated the tasks into two categories based on their context: *data-centric* and *view-driven*.

   a. Data-centric tasks are motivated by users' need to change (aspects of) the underlying data and are independent of the visualization. Examples of data-centric tasks are filtering and changing the axis attribute. To perform these tasks, the user typically needs to pick an option from a set of options or a range. For instance, to filter data by an attribute, the user needs to view the entire range of values as well as the currently active range. Similarly, to change the attribute of an axis, the user needs to view all the attributes to pick one. Since these tasks require presentation of options, they are natural candidates for WIMP-style interactions.

   b. View-driven tasks, on the other hand, are motivated by users' desire to interact with the visualization or modify it, and do not affect the underlying data. Zoom, sort, and selection are examples of view-driven tasks. The selection task, for

instance, is independent of whether the data attributes being selected are quantitative or nominal. Since these operations do not require a presentation of options, we believe that gestures can be used to implement interactions for them.

2. We further subdivide the data-centric (1a) tasks based on frequency of use into *essential* and *on-demand* tasks. This categorization helps position the interactions and their interface elements onto either the main view or in menus & submenus. Essential tasks are frequently used and it is vital that users are able to perform these tasks in a minimum number of steps. Hence, there is value in making them available to users on the main view. Examples are selection, changing attribute on an axis, and showing data details. Conversely, on-demand tasks are applied by users infrequently and do not need interaction elements on the main view. Examples are changing the axis scale from linear to log and adding jitter to the glyphs.

The two-step categorization of tasks based on context and frequency helped to focus and clarify our development of the scatterplot application and its interactive operations. In the subsequent section, we describe both the design considerations and implementation decisions that we made.

# 5. DESIGN AND IMPLEMENTATION

We implemented the interactive scatterplot prototype application on an Apple iPad using iOS's Cocoa Touch framework. The application is optimized for a 9.7 inch screen with a 2048 x 1536 resolution. The initial view of the application presents a scatterplot using a random pair of variables from the dataset. The main view consists of data points, the two axis lines, and axis labels placed next to each line. The application reads data from a csv file where the first row contains attribute names.

In the following subsections, we highlight potential options that we considered for interactions for the primary tasks. The options are a mix of some that we obtained from past research and others that we crafted ourselves. We implemented most of the options in order to gain a "feel" of what they are like in practice. We relate these experiences and discuss the merits and limitations of these options. Finally, we highlight one or more options that we consider best for each task. Figure 1 shows images from the application and an accompanying video provides a more interactive look at the design choices described below and some of the operations that we implemented.

## 5.1 Selection

Selection is a core task in any visualization system. In traditional desktop systems, selection can be realized both through hover and click-to-select actions. A selection task predominantly constitutes the following use cases:

1. A user identifies a data point and wants to learn its details.
2. A user wants to highlight a point and track it across views, such as a change in axis attributes.
3. A user identifies a point and wants to include it in or exclude it from subsequent operations.

Each of these tasks is applicable to a single point or a set of points. Moreover, the set of points can be located close together or at diverse locations in the visualization. In the case of closely located points, the points could be non-overlapping, partially overlapping, or completely overlapping with each other.

To support the above tasks, the chosen interaction must be able to select each individual data point. On a touch screen, this causes considerable usability problems because large finger sizes do not match well to the typically small size of scatterplot glyphs. As a consequence, a glyph's minimum size may be limited by the size of a finger. The glyph size in turn then limits the number of data points that effectively can be shown. In our prototype, although we do not limit the size of the input, we found the interface to be most effective when number of data items is less than 1000.

Similar issues of selection on touch screens have been carefully studied in the past. Moscovich [21] presents precise selection techniques for closely placed widgets by enhancing the pointer activation areas. Although quite useful, the technique fails to address situations where the underlying objects have overlaps. Benko et al. [4] highlight a host of other options for precise selection on touch. All these options suggest that there is no ideal selection technique for touchscreen interaction as there is for cursor-based interactions. Here we highlight some potential options that we considered suitable for scatterplots:

A. Lasso: User draws a path to enclose a region containing the point(s) of interest.
B. Rubber band: User drags on the view to draw a rectangular area containing the point(s) of interest. This is likely faster but less precise than lasso selection.
C. Zoom view: User taps-and-holds near the data points. This reveals a zoom lens (similar to the iOS text correction view). User taps to select the data point inside the lens and hides the lens by tapping outside.
D. Swipe+Lens: User swipes on the intended point to select it. If system detects multiple points with the swipe, a lens opens with a magnified view to assist in precise selection [19].
E. Off-centered pointer: User taps-and-holds the screen to reveal a cursor positioned n-pixels above and to the left of the touch location. The user drags the cursor over the intended point. Dragging over the point reveals details and lifting the finger performs selection [28].
F. Axis Pan: User slides one finger on both axes simultaneously. This creates a horizontal and vertical reference line. The data point under the intersection of the two lines is selected.

Various studies [4], [10], [20] have compared a combination of the above selection techniques. However, they all consider interfaces with a relatively low density of selectable elements. In our experimentation with these options, we identified issues with a number of them. *Off-centered pointer* (E) has the limitation that certain positions on the view, such as the bottom edge, are inaccessible because of the way the cursor is placed. *Axis pan* (F) necessitates bimanual input that, though feasible on tablets, requires the user to first place the tablet on some surface. *Zoom view* (C) and *swipe+lens* (D) use a zoomed-in lens view. Selection inside a lens creates issues for scatterplots since the shapes and colors of the glyphs are often the same. As a result, switching between the lens and non-lens modes causes a loss of target.

From among the list of possibilities, we found *lasso* (A, Figure 2a) and *rubber-band* (B) to be most effective for selection. We support both these options in our prototype. *Rubber-band* builds on user's familiarity from use in visualization systems on desktops. Alternately, *lasso* gives users finer control during selection. To provide the user with feedback while drawing a path, we highlight the area formed by completing the path between the start and the current point.

Another approach to selection is by means of zooming [4], [23]. We can break down selection of a data point in a densely packed region into three steps: zooming in, selecting, and zooming out. The steps ensure high precision, as there is no limit to how much the view can be zoomed. On the other hand, zooming increases the time cost of selection and also leads to a loss in context similar to lens-based solutions. However, in some situations, the location of data points makes performing selection without zooming near impossible. In our prototype, we support selection of points at all states of zooming (discussed in the next section).

## 5.2 Zoom

The zoom operation is another vital interaction for visualizations. The operation modifies the viewport to increase clarity of data points that lie too close to each other. Below we highlight some potential ways to perform zooming:

A. Pinch-to-zoom: User performs a pinch operation using two fingers. The visualization scales depending on finger movement.

   A1. Fixed aspect ratio: The visualization scales up or down uniformly in both directions.

   A2. Flexible aspect ratio: The visualization scales up or down independently in each direction.

   A3. Critical angle: If the angle between the *x*-axis and the line that connects the two fingers is less than 45 degrees, the visualization scales on X. If the angle is greater than 45 degrees, the visualization scales on Y.

B. Axis-based zoom: Instead of performing a gesture on the view, the user performs the pinch operation directly on the axis that s/he wants to scale.

C. Select + zoom: The user highlights a region on the axis or a set of points on the view. The user then double-taps to scales the view to the selected points.

D. Zoom lens: The user performs a pinch operation to reveal a lens containing the magnified view of the region between fingers. The user can select the data within the lens and modify its magnification. [17]

E. Automatic zoom: The user double taps on the view to magnify the region around the touch location. The view is magnified by an amount that minimizes the number of overlapping points in the region.

On touch-based devices, the pinch-to-zoom (P2Z) gesture (A1) has been employed extensively to perform zoom across applications, such as maps, documents, and so on. However, we identified a number of problems in using P2Z for scatterplots. The primary problem arose as a result of the fixed aspect ratio constraint of P2Z. When scaling content such as images, maps or documents, the gesture typically maintains the aspect ratio of the underlying content and does not permit scaling in only one direction. For these content types, maintaining a constant aspect ratio is appropriate since the two dimensions of the view are dependent. However, in a scatterplot, the two dimensions, i.e. axes, are largely independent. It is fairly common for the data to be densely packed in such a manner that zooming in only one direction suffices. As a result, although P2Z was effective in some situations, we decided against using it because of restrictions it caused in many other situations.

We modified the P2Z gesture by relaxing the fixed-aspect ratio constraint (A2). However, in our implementation, we found this modified gesture to be fairly difficult to use. In particular, when

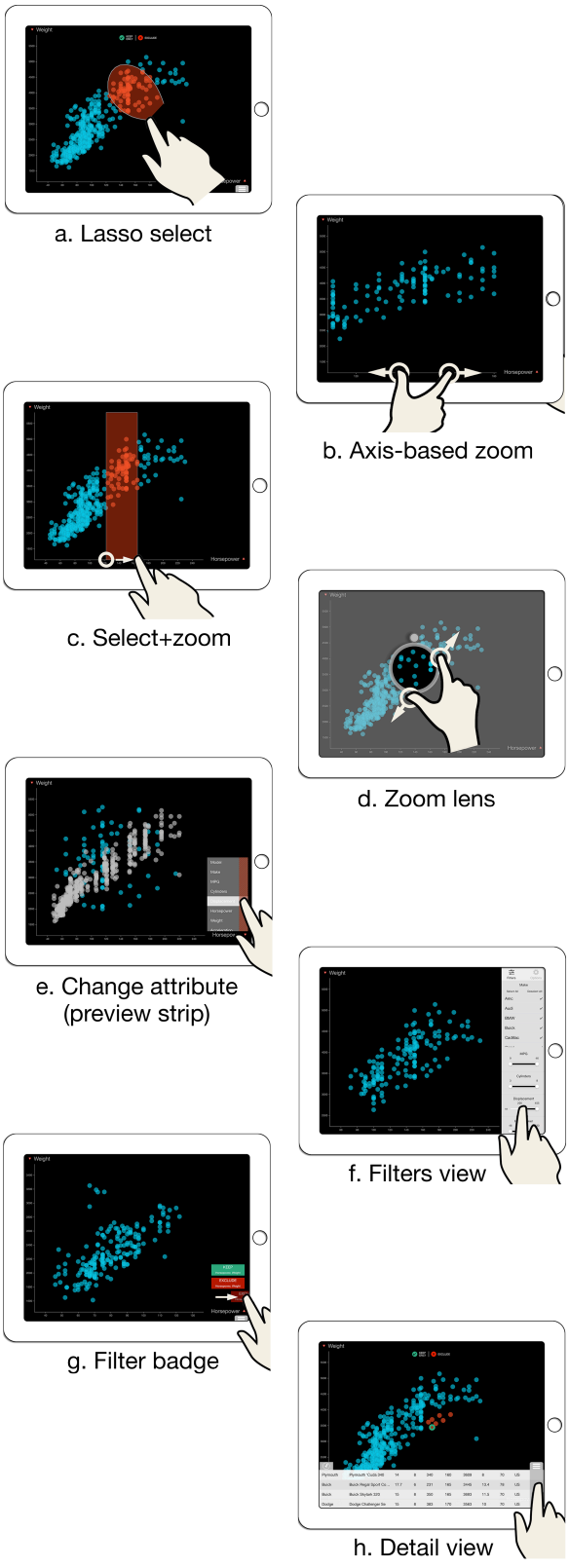


**Figure 2. Interaction techniques used for primary tasks in the prototype running on an Apple iPad. Illustrations provided using GestureWorks® (www.gestureworks.com)**

we intended to scale the view in only one direction, we needed to move fingers very precisely only in that direction so as not to perform any movement in the other direction. We subsequently also considered other variations to P2Z (A3), but they lead to a less desirable situation where the default behavior of pinch-to-zoom that users have learned and expect is modified.

In all these variations of P2Z, the primary issues arise as a result of the gesture operating on both dimensions simultaneously. We can manage this by adding context information (position) to the gesture. *Axis-based zoom* (B) provides one way to do that (Figure 2b). The user scales each dimension individually by performing the pinch gesture directly on the corresponding axis. Alternately, the user can also tap-and-drag to highlight a range on the axis and select the zoom action (*select+zoom*: C, Figure 2c). These options have the advantage that users can be precise about zooming into a region on the view or zooming to a range of values on the axis.

We found both *axis-based zoom* and *select+zoom* to be useful for zooming. Since the interactions do not conflict, we currently support both in our prototype. However, both of these are performed on the axes, not the data portion of the view. As a result, the user will not receive feedback when s/he performs P2Z in the data area. Given users' expectations and familiarity with P2Z on touchscreens, we believe that it is essential for the system to support the gesture. Thus, we utilize *zoom lens* (D) as an additional mechanism for zooming in the view (Figure 2d). The *zoom lens* uses the P2Z gesture and is particularly useful when users want to view data density around a point without losing the overall context. The lens supports repositioning, changing zoom level and selection of points inside. The lens module is also extensible to support filters [27].

Finally, we also incorporate *automatic zooming* (E). Users primarily zoom to distinguish data points located in a dense area. To facilitate this, the system can automatically calculate an appropriate zooming amount in order to minimize overlaps and close contacts among the glyphs in that area. Images and maps on smartphones support a similar style of zooming, called smart zoom, where a user double-taps on the screen to zoom the content by a fixed amount (for maps, the amount is the subsequent tile level). We extend smart zoom by utilizing the same gesture – double tap – thus utilizing users' familiarity.

## 5.3 Change attributes

In section 4, we defined changing the axis attribute as a *data-centric* task that entails presenting the user with a set of options. We believe that a list is ideal for such tasks. We also classified the task as *essential* due to its frequency of use. It is desirable for the task to be achievable in fewest steps possible. As a result, we incorporate the task in the main scatterplot view. We made the labels representing the axis attributes interactive so that tapping them reveals a list of attributes the user can choose. Once the user selects an option, the data points animate to their new positions and the list closes. Alternately, the user can dismiss the list by tapping outside it. The position of the axis labels on the view suits both the feedback on the currently selected attribute and easy access to the change action.

One extension to this attribute switching is *preview* mode [8]. The main purpose of this mode is to show users a preview of the effects of an action before they commit it. They then choose to either commit the action or revert to the original state. In a visualization system, preview mode serves an additional purpose. Consider the situation where a user selects two data points and wants to compare them on different attributes. Normally, the user would have to then sequentially switch to every attribute. With the preview mode, the user can simply enable preview states serially on each attribute without committing selection to any one. Further, this allows the user to compare the position of points in the previewed state and the current state. Preview mode is also useful for tracking changes in color, shape, and size.

On mouse-based interfaces, the preview mode is typically engaged using mouse-hover. On touch interfaces, the equivalent of mouse-hover, i.e. finger panning, conflicts with the default scrolling operation of a list view. We experimented with modifications to a list view to support preview mode, such as:

A. Two finger drag: User performs a two-finger-pan on the list of options. The system previews the data attribute corresponding to the row at the centroid of the fingers.

B. Hold for preview: Tap-and-hold on the list engages the preview mode that allows the user to pan on the attribute rows to preview.

C. Preview Handle: The pop-up list contains a circular handle on the right edge. The handle can be dragged up and down. The preview mode is engaged for the row the handle is currently over.

D. Preview Strip: The list contains a vertical strip on one end. Panning on the strip previews the row corresponding to the position of the finger.

E. Fisheye: The height of each row in the list view is compressed so that all rows are visible. Panning on the list magnifies the row below the finger and shows the preview for that attribute. Lifting the finger selects the row.

In our trials, *two-finger drag* (A) caused considerable occlusion since multiple fingers were placed on a single row of the list. *Hold for preview* (B) had the problem of lag as the user had to wait to first engage the preview mode. *Fisheye* (E) was feasible for small sized lists (~10 items) but for longer lists, the text size became very small. Although we could still access each option using a fisheye, we could not get an overview of all the options simply by looking. More importantly however, similar to modifications of P2Z, these three options have the disadvantage that they modify the default scroll behavior of lists that users are familiar with.

*Preview handle* (C) and *preview strip* (D), on the other hand, do not affect the default scroll behavior. *Preview handle* provides a circular handle that the user drags over each row to preview the row. *Preview strip* provides a rectangular strip that extends through the height of the list. Whereas panning the finger on the list scrolls it vertically, panning on the strip previews the row corresponding to the finger. We found both these options to be effective, and support both in our prototype. However, given its size, it is considerably easier to touch the strip than the handle, particularly with fingers. As a result, we use the strip as the default mode (Figure 2e). The strip contains a texture that helps separate the region from the main list and also affords panning on the strip. To select a row (attribute) while previewing, the user can simply lift his/her finger. Alternately, a user can drag the finger outside the strip to cancel the preview.

## 5.4 Filter

Filter interaction techniques enable users to change the set of data items being presented based on some specific conditions. Users typically specify a data range or condition, and the system shows only those data items that meet the criteria. Data items outside of the range or not satisfying the condition are either hidden from the display or shown differently.

In a scatterplot, the filter operation is similar to zoom in that the same visual representation can often be achieved using either operation. They differ in how the effect is achieved, however. Zoom operations are view-based and change the range of values being used to display glyphs. Conversely, filter operations operate on the data space; particular values or ranges of values can be selected, and only data matching those criteria are shown. Filtering out data removes it from every subsequent visual operation (including zoom) until the filter is explicitly removed.

### *5.4.1 Data-centric filter*

A user's intention to filter data is primarily motivated by two use cases. In the first use case, the user wants to filter points based on values of some attributes. For instance, in a scatterplot showing data for average salaries for people in a country, the user wants to view data for only those with age above 50 or those with a PhD degree. This is the *data-centric, on-demand* filter we present in Table 1. To support this task, the system needs to extract the entity details from the data model, find the range of values for the various attributes, and display filter options. And since the user needs to view the options, a reasonable solution for tablets is to show interface widgets such as checkboxes and sliders, similar to desktops. These widgets have proved to be very effective [21] and tablet operating systems such as iOS and Android use these extensively. Apart from supporting filtering, the widgets also present the currently active range of values. Since we classify data-centric filters as on-demand, these filters are placed in a menu positioned off-screen to the right (Figure 2f). The menu can be dragged on the screen by swiping inwards from the right edge.

### *5.4.2 View-driven filter*

In the second use case for filtering, the user identifies a set of points in the view that s/he wants to either focus on or filter out of the data. For instance, the user might want to concentrate only on a set of outlying data points. This is the *view-driven* filter from in Table 1. To support this operation, each time the user selects a few data points, the interface provides the user with buttons to 'keep only' or 'exclude' the selected data. Users can select the data to filter using a *lasso* (5.1.A) or *rubber band* (5.1.B) interaction. Alternately, the user can also *tap-and-pan* (5.2.C) on an axis to select a range of values. Tap-and-pan has the added advantage of enabling both view-driven and data-centric filtering.

Each view-driven filter operation updates the filter widgets appropriately. Additionally, a *badge* representing the filter also appears in a stack on the scatterplot view as additional feedback to the user (Figure 2g). This is essential because a view-driven filter operation might generate a situation where the data filtered does not produce any change in the widgets (e.g. if the data filtered out lies in the middle of a slider). Additionally, the badge allows the user to individually disable or remove the filter.

## 5.5 Find Data Details

Viewing details of data points (all their attributes) is crucial to any data exploration task using visualizations. A table view is an ideal widget to present these details. Hence, we use a table in our prototype (Figure 2h).

Each time the user selects data points on the scatterplot, a table view is populated with the details of these points. This table is initially placed off-screen at the bottom. The user can drag the table in using a handle that animates into view after selection. The table and the handle hide automatically when the user deselects the data points. Alternately, the user can hide the table manually with the handle. The user also can snap the table to always stay in view using the snap button. If snapped, the visualization compresses vertically to fit the remaining area. The table also has a preview strip similar to attribute popup. Panning on the strip highlights the glyph corresponding to the touched row.

For the table view, instead of using a handle to drag the table, we considered using a bottom edge swipe-up gesture. However, the gesture conflicts with iPad's system-wide gesture to bring up the control center. More importantly, the table view is not available when no data is selected. A swipe-up gesture, however, does not provide feedback of unavailability of the table in the manner the absence (or disappearance) of the handle does.

In situations when the user wants to only view the data values for attributes on the axes, instead of revealing the table view, the user can pan on any axis to reveal a reference line. When the reference line intersects a data point, a perpendicular reference line becomes visible, highlighting the exact value on the other axis.

## 5.6 Modify Visual Mappings

The data-centric on-demand features (ODF) from Table 1 (map color and size, change axis scale) are assembled and placed in the same menu as the data-dependent filters. The menu has separate tabs for filters and ODF. The ODF tab offers options to assign color and size to the glyphs as well as options to change the axis scale from linear to log for both the $x$ and $y$ axis. The axis scale options are disabled in case of nominal attributes on the two axes.

# 6. CONCLUSION

In this article, we explored the key challenges in designing an information visualization system for multi-touch tablet computers. More specifically, we explored the design space of interactions for an interactive scatterplot visualization. By investigating two widely used, commercial desktop scatterplot applications, we derived a set of tasks that our system should support. To make the construction of a prototype more feasible, we chose to implement only a subset of primary tasks. For these tasks, we then developed a suite of potential multi-touch interactions, based either on prior research or our own designs. We iterated on most of these options, explored their usability, and developed a prototype using the best alternatives we identified.

A natural follow up to this work is a user evaluation of our prototype system. Multiple approaches make sense. First, user testing on the various options for each task will provide helpful feedback about their usability. We could then take the interactions that emerge and bundle them into a complete prototype. The resulting application could be compared with a current commercial system for tablets or a WIMP-style system in the manner done by [7]. Finally, a longer-term usage study would be the best evaluation of an application like this. Clearly, people's prior experience with multi-touch applications will have a strong influence. Issues such as discoverability, learnability, and longer-term satisfaction emerge better when testing is conducted in a realistic setting for an extended period of time.

Because visualization systems seldom provide only a single representation for viewing data, we would also like to examine the applicability of the developed interactions for other types of visualizations. An obvious extension to our application would be to incorporate other visual representations such as linecharts, barcharts, treemaps, and parallel coordinates. However, once different visual representations appear together in an application, one confronts issues relating to gestures and interactive operations

transferring across different representations. This clearly presents a further challenge – developing a suite of compatible multi-touch gestures across a variety of visualizations.

Finally, we would also like to analyze the feasibility of porting our prototype to other operating systems such as Android and Windows. We speculate that a number of issues will emerge since some components of our interaction design, such as swiping in from the right and bottom, are available for developers to use only on iOS. On Android and Windows, the OS uses these gestures for system level tasks and an application cannot override their default use. Conversely, gestures such as swipe up from bottom are available on both Android and Windows, but are in use by iOS. Discovering a combination of interactions that work independently of the operation system remains a challenge.

# 7. REFERENCES


[1] Ahlberg, C. and Shneiderman, B. Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays. *Proc. of ACM CHI,* Apr 1994, 313–317.

[2] Baur, D., Lee, B. and Carpendale, S. TouchWave: Kinetic Multi-touch Manipulation for Hierarchical Stacked Graphs. *Proc. of ACM ITS,* Nov. 2012, 255–264.

[3] Beaudouin-Lafon, M. Instrumental Interaction: An Interaction Model for Designing post-WIMP User Interfaces. *Proc. of SIG CHI*, Apr. 2000, 446–453.

[4] Benko, H., Wilson, A.D. and Baudisch, P. Precise Selection Techniques for Multi-touch Screens. *Proc. of ACM CHI,* Apr. 2006, 1263–1272.

[5] Browne, J., Lee, B., Carpendale, S., Riche, N. and Sherwood, T. Data Analysis on Interactive Whiteboards Through Sketch-based Interaction. *Proc. of ACM ITS,* Nov. 2011, 154–157.

[6] Buering, T., Gerken, J. and Reiterer, H. User Interaction with Scatterplots on Small Screens - A Comparative Evaluation of Geometric-Semantic Zoom and Fisheye Distortion. *IEEE Trans. on Visualization and Computer Graphics*. 12, 5 (2006), 829–836.

[7] Drucker, S.M., Fisher, D., Sadana, R., Herron, J. and schraefel, m. c. TouchViz: A Case Study Comparing Two Interfaces for Data Analytics on Tablets. *Proc. of ACM CHI,* Apr. 2013, 2301–2310.

[8] Forlines, C., Shen, C. and Buxton, B. Glimpse: A Novel Input Model for Multi-level Devices. *CHI'05 Extended Abstracts*, Apr. 2005, 1375–1378.

[9] Frisch, M., Heydekorn, J. and Dachselt, R. Investigating Multi-touch and Pen Gestures for Diagram Editing on Interactive Surfaces. *Proc. of ACM ITS*, Nov. 2009, 149–156.

[10] Gunn, T.J., Zhang, H., Mak, E. and Irani, P. An Evaluation of One-handed Techniques for Multiple-target Selection. *CHI'09 Extended Abstracts*, Apr. 2009, 4189–4194.

[11] Heer, J., Bostock, M. and Ogievetsky, V. A tour through the visualization zoo. *Commun. ACM*. 53, 6, Jun. 2010, 59–67.

[12] Heilig, M., Huber, S., Demarmels, M. and Reiterer, H. ScatterTouch: A Multi Touch Rubber Sheet Scatter Plot Visualization for Co-located Data Exploration. *Proc. of ACM ITS,* Nov. 2010, 263–264.

[13] Hinckley, K., Yatani, K., Pahud, M., Coddington, N., Rodenhouse, J., Wilson, A., Benko, H. and Buxton, B. Manual Deskterity: An Exploration of Simultaneous Pen + Touch Direct Input. *CHI'10 Extended Abstracts,* Apr. 2010, 2793–2802.

[14] Isenberg, P., Fisher, D., Morris, M.R., Inkpen, K. and Czerwinski, M. An exploratory study of co-located collaborative visual analytics around a tabletop display. *Proc. of IEEE VAST*, Oct. 2010, 179–186.

[15] Isenberg, P., Isenberg, T., Hesselmann, T., Lee, B., von Zadow, U. and Tang, A. Data Visualization on Interactive Surfaces: A Research Agenda. *IEEE Computer Graphics and Applications*. 33, 2, (2013), 16–24.

[16] Jansen, Y. and Dragicevic, P. An Interaction Model for Visualizations Beyond The Desktop. *IEEE Trans. on Visualization and Computer Graphics*. 19, 12 (2013), 2396–2405.

[17] Käser, D.P., Agrawala, M. and Pauly, M. FingerGlass: Efficient Multiscale Interaction on Multitouch Screens. *Proc. of ACM CHI*, May 2011, 1601–1610.

[18] Lee, B., Isenberg, P., Riche, N.H. and Carpendale, S. Beyond Mouse and Keyboard: Expanding Design Considerations for Information Visualization Interactions. *IEEE Trans. on Visualization and Computer Graphics*. 18, 12 (2012), 2689–2698.

[19] Mankoff, J., Hudson, S.E. and Abowd, G.D. Interaction Techniques for Ambiguity Resolution in Recognition-based Interfaces. *Proc. of ACM UIST,* Nov. 2000, 11–20.

[20] Mizobuchi, S. and Yasumura, M. Tapping vs. Circling Selections on Pen-based Devices: Evidence for Different Performance-shaping Factors. *Proc. of ACM CHI,* Apr. 2004, 607–614.

[21] Moscovich, T. Contact Area Interaction with Sliding Widgets. *Proc. of ACM UIST*, Oct. 2009, 13–22.

[22] North, C., Dwyer, T., Lee, B., Fisher, D., Isenberg, P., Robertson, G. and Inkpen, K. Understanding Multi-touch Manipulation for Surface Computing. *Proc. of INTERACT,* Aug. 2009, 236–249.

[23] Olwal, A., Feiner, S. and Heyman, S. Rubbing and Tapping for Precise and Rapid Selection on Touch-screen Displays. *Proc. of ACM CHI*, Apr. 2008, 295–304.

[24] Rzeszotarski, J.M. and Kittur, A. TouchViz: (Multi)Touching Multivariate Data. *CHI '13 Extended Abstracts,* Apr. 2013, 1779–1784.

[25] Sadana, R. and Stasko, J. Interacting with Data Visualizations on Tablets and Phones: Developing Effective Touch-based Gestures and Operations (Poster). IEEE Vis, Atlanta, GA, Oct 2013.

[26] Siek, K.A., Rogers, Y. and Connelly, K.H. Fat Finger Worries: How Older and Younger Users Physically Interact with PDAs. *Proc. of INTERACT,* Sep 2005, 267–280.

[27] Stone, M.C., Fishkin, K. and Bier, E.A. The Movable Filter As a User Interface Tool. In *Proc. of ACM CHI,* Apr. 1994, 306–312.

[28] Vogel, D. and Baudisch, P. Shift: A Technique for Operating Pen-based Interfaces Using Touch. *Proc. of ACM CHI*, Apr. 2007, 657–666.

[29] Walny, J., Lee, B., Johns, P., Riche, N.H. and Carpendale, S. Understanding Pen and Touch Interaction for Data Exploration on Interactive Whiteboards. *IEEE Trans. on Visualization and Computer Graphics*. 18, 12 (2012), 2779–2788.

[30] Wobbrock, J.O., Morris, M.R. and Wilson, A.D. User-defined Gestures for Surface Computing. *Proc. of ACM CHI*, Apr. 2009, 1083–1092.

[31] Yi, J.S., Kang, Y., Stasko, J. and Jacko, J. Toward a Deeper Understanding of the Role of Interaction in Information Visualization. *IEEE Trans. on Visualization and Computer Graphics*. 13, 6 (Nov. 2007), 1224–1231.