

Supporting Asynchronous Interactive Visualization for Exploratory Data Analysis

Larry Xu



Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2017-97

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-97.html>

May 12, 2017

Copyright © 2017, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

**Supporting Asynchronous Interactive Visualization
for Exploratory Data Analysis**

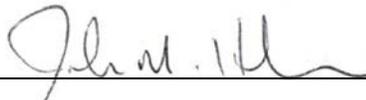
by Larry Xu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:



Professor Joseph M. Hellerstein
Research Advisor

5/9/17

(Date)

* * * * *



Professor Eugene Wu
Second Reader

5/9/17

(Date)

Abstract

Supporting Asynchronous Interactive Visualization for Exploratory Data Analysis

by

Larry Xu

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Joseph M. Hellerstein, Chair

Interactive data visualizations have become a powerful tool for exploratory data analysis, allowing users to quickly explore different subsets of their data and visually test hypotheses. While new design tools have made creating such visualizations easier, these tools often do not scale well to large datasets that do not fit on a user's machine. This becomes increasingly more common as data collection grows and storage costs become cheaper. In such cases, the visualization must request new data across a network for each interaction. When interactive latencies are introduced to a visualization, the user experience can deteriorate to the point of being inconsistent and unusable. To address these issues we present a new interaction design technique, *Chronicle Interactions*, which allows for creating asynchronous interactive visualizations that are resilient to latency. Through online user studies, we validate that users are able to complete visual analysis tasks quicker with this new design compared to a traditional interface under conditions of latency.

Contents

Contents	i
1 Introduction	1
2 Modeling Interactive Visualizations	5
2.1 Event streams	5
2.2 Human-in-the-loop distributed systems	6
2.3 Visual encodings	8
2.4 Cognition	9
3 Chronicled Interactions	12
3.1 Data layer: history of interactions	13
3.2 Visual layer: visualizing history	16
3.3 Correspondence: mapping interactions to results	18
3.4 Summary	19
4 User Experiments	21
4.1 Spatial vs. temporal designs	21
4.2 Experiment design	22
4.3 Experiment analysis	26
5 Conclusion	35
Bibliography	36

Acknowledgments

This work would not have been possible without the help of so many individuals I have had the honor of collaborating with. I would like to thank my research advisor Joseph Hellerstein for providing enormous support throughout both my undergraduate and graduate years at Berkeley. He has been a great source of advice in research, teaching, and graduate school, and I am constantly impressed by his wide breadth of expertise and ability to balance academic, professional, and personal life. Eugene Wu, who I like to consider my second advisor at Columbia University, thank you for taking a chance on a young Berkeley undergrad during your short time here at Berkeley two years ago, and for guiding and mentoring me through research and academia ever since then. Remco Chang at Tufts University, thank you for contributing your wealth of HCI wisdom, and always keeping things down-to-earth in all of our discussions. And to my research partner in crime, Yifan Wu, none of this work would have been possible without you and I am grateful for having been able to work together in the research trenches this past year.

Many others have helped me throughout my academic research journey at Berkeley that I would like to thank as well. Thank you to Arnab Nandi of Ohio State University and his group there including Meraj Ahmed Khan and Lilong Jiang for the opportunity to work on research together during my time as an undergrad. Thank you to Eugene Wu's group at Columbia University including Hamed Nilforoshan and James Sands for contributing their time and energy on supporting our experiments. Thank you to members of the Berkeley AMPLab & RISELab community including Daniel Haas and Sanjay Krishnan for their sage advice both in previous research and in the work presented here. And to my fellow lab students Michael Whittaker, Johann Schleier-Smith, Vikram Sreekanti, and Chenggang Wu, thank you for inspiring me through your work and teaching me a little something new every week.

Thank you to all my friends who have helped keep me sane during my last year at Berkeley, and for all the memories, laughs, and misadventures we shared. And finally, thank you to my parents, Nelson and Connie, and my brother Kevin for supporting me in all that I have failed and accomplished until now, and for their continual support in the future.

Chapter 1

Introduction

Visualization plays a large role in exploratory data analysis as a method of gaining valuable insights from looking at data. Visual analysis leverages human perception to find patterns that may not be immediately evident from looking at a table of numbers. A classical example of this is shown in Figure 1.1, Anscombe's quartet [2]—four datasets with the exact same descriptive statistics (mean, variance, correlation), but that reveal wildly different patterns when plotted. This simple example highlights the ability of visualizations to easily reveal trends and anomalies in data. Heer et al. [19] present a comprehensive list of visualization techniques for exploratory analysis.

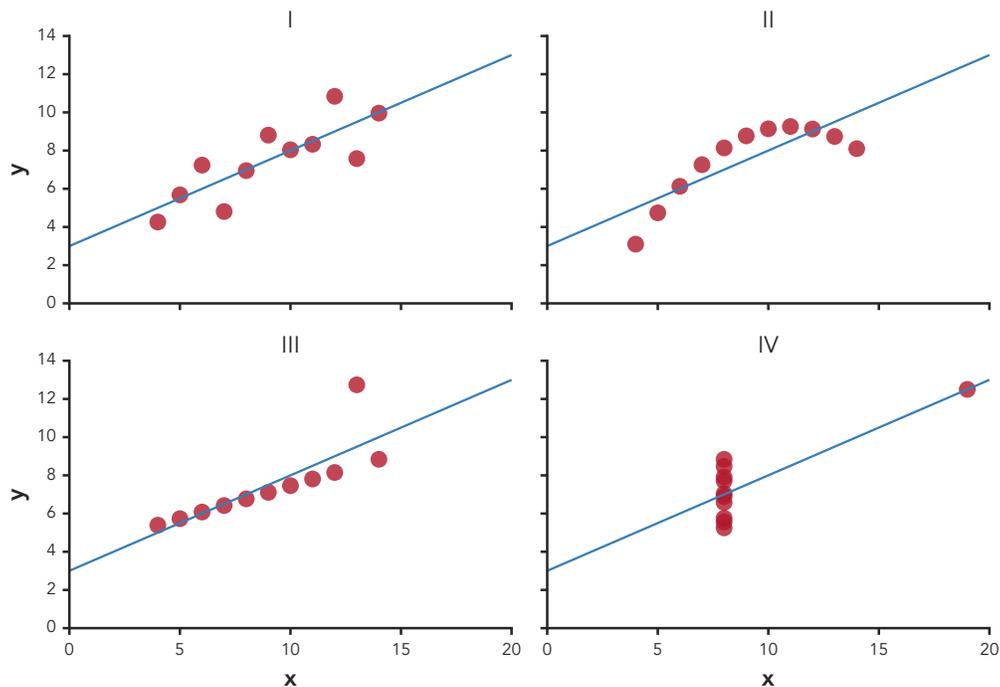


Figure 1.1: Anscombe's quartet

The creation of domain-specific tools and languages has made designing such data visualizations progressively easier. `ggplot2` [43] in R allows for declaratively specifying graphs without any imperative logic involved. With only a few simple commands, users are able to visualize data and customize a visualization through specifying different encodings. D3 [5] has become widely used for its ability to easily author visualizations on the web. It supports a diverse set of charts through visual primitives and plugins. The success of `ggplot2` and D3 has shown that lowering the complexity for creating visualizations enables people to perform quicker analysis and communicate their results faster.

More recently, *interactive* data visualizations have enabled richer user experiences for exploring data. Visualizations that support interactions allow the user to directly manipulate the data and process multiple views or subsets. While static visualizations are useful for analyzing small datasets with simple quantitative or categorical variables, accommodating large datasets with different kinds of variables such as geospatial, time series, or relational data can be challenging. To support a wide variety of variable types, it is useful to display data in an integrated visual dashboard with multiple different coordinated views.

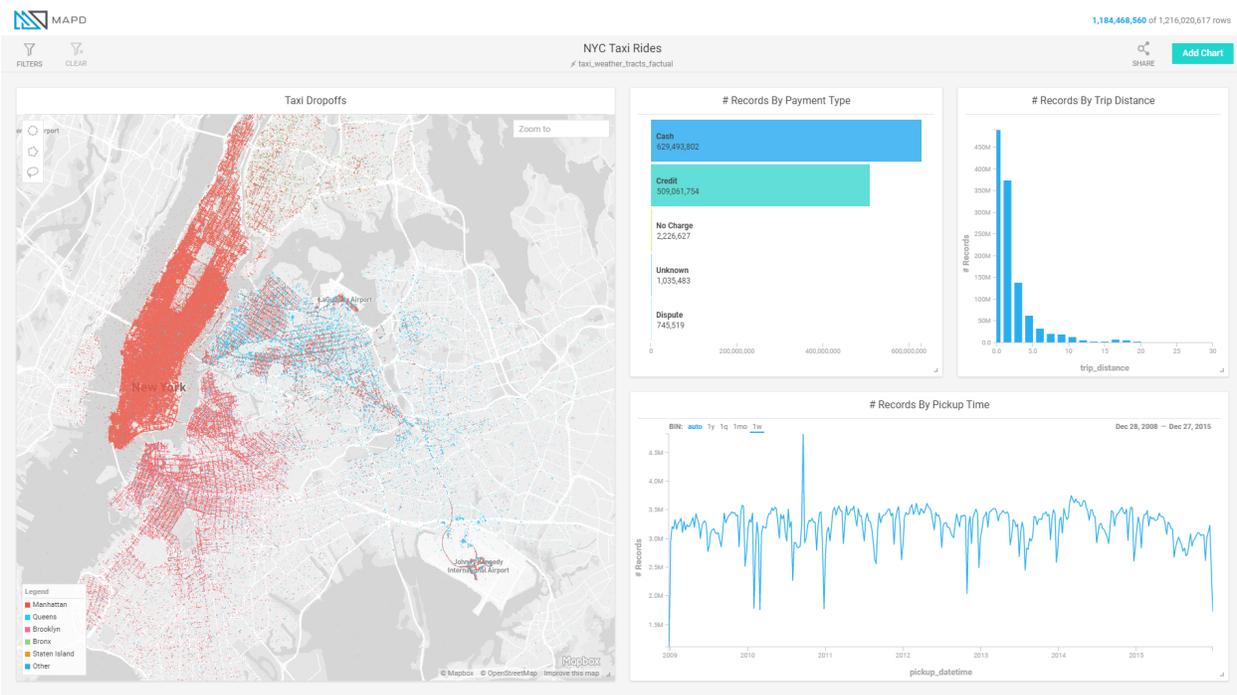


Figure 1.2: MapD NYC Taxi Rides interactive data visualization. This visual dashboard lets users explore multiple dimensions of taxi data and uses a variety of visual displays including map, time series, and histograms. These displays are *coordinated*, such that interactions which filter one display affect the other displays.

Consider the MapD visual dashboard of NYC taxi rides in Figure 1.2 [26]. This high-density display allows exploration of one billion records of data. Such information density cannot be captured with tabular data displays or static visualization displays. For the dashboard to be effective, it relies heavily on interaction to allow users to explore different parts of the data and gain relevant insights. The dashboard itself does not leverage particularly complex visualizations. Maps, line charts, and histograms are common representations used in data analytics. Interactive visual dashboards can contain a diverse assortment of charts and visualizations, and any interactive filtering applied to one view will affect the others in a technique called brushing and linking [3].

Developing interactive visualizations like this can be much more difficult and time-consuming compared to their static counterparts. Designers must reason about imperative event handling and state management to support interactions. Common visualization toolkits such as `ggplot2` and `D3` do not natively support interactions, so new tools have been developed specifically for interactive visualizations. Vega [35] is an interactive visualization grammar which attempts to address these issues through declarative abstractions. It is able to support high-level descriptions of interaction behaviors without needing to specify imperative event handling procedures. Although less declarative than Vega, `Plotly` [36] similarly provides a library for easily creating interactive visualizations without the need for complex code. Both of these tools are targeted at authoring interactive web visualizations.

Despite this progress in easily specifying interactive visualizations, there exists another issue that needs to be addressed when creating high-density interactive visualizations as in the MapD example. The visualization must be able to scale to large datasets to support interactive big data analytics. This typically requires querying data across a network. Data may be distributed across multiple machines and geographically decentralized. When trying to explore and analyze such a large distribution of data, queries must be made to multiple different nodes involving potentially large data transfers across a network. For example, consider a sales analyst looking at data managed by an Apache Spark [45] cluster to aggregate their company sales data from across multiple geographic regions. Each query they run can take on the order of seconds to complete [8], making it prohibitively expensive to interactively explore multiple slices of data in sequence.

One of the prominent challenges of visual analysis today is enabling interactive data visualization for large-scale exploratory data analysis scenarios such as these. The core issue is limiting latency and its effects on the usability of a visualization. Given that the data analyzed in a visualization may not be immediately available on a user's machine, they must make network requests to one or more remote sources for data. These requests inherently incur latency that scales with the size of the data or workload being queried. When every user interaction performs a new request, this can slow down productivity and the ability to effectively explore the data. Variance in latencies can cause out-of-order responses, and potentially result in inconsistencies on the interface. These can create a frustrating user experience by exacerbating cognitive issues in visualizations such as change blindness [31], short term memory limits [6], and the gulf of evaluation [30].

In response to these challenges, special-purpose systems such as MapD’s GPU accelerated database and visual interface have been built to handle low-latency visual analytics queries on large datasets. Tableau [42] offers its own system for composing visualizations from heterogeneous data sources such as multiple different databases. Such tools can be compelling for interactive big data analytics, where visual toolkits like D3 cannot scale. Although proprietary end-to-end visualization systems like Tableau and MapD are making the problems of latency in interactive visualizations easier, they require buying into an ecosystem rather than allowing visualization designers freedom to work with their own tools. Similarly, while a large variety of known backend techniques exist for decreasing latency such as sampling, aggregation, and caching; these are not general-purpose enough to work with all data sources and are still limited by variance in network speeds.

So the motivating question becomes, can we create simple *interaction designs* that are still effective in the face of latency? By starting with the assumption that latency exists in an interactive visualization, we can better design around this issue. In this thesis we present Chronicle Interactions, a model for designing asynchronous interactive visualizations that are resilient to issues of interactive latency. The model is meant to extend existing visualization toolkits to better support large datasets for exploratory analysis across a variety of backend data systems. Through a set of online user studies, we demonstrate that users can effectively leverage these new interaction designs to improve performance on a variety of visual analysis tasks under latency compared to traditional interfaces.

Chapter 2

Modeling Interactive Visualizations

To design effective interactive visualizations for large-scale exploratory data analysis scenarios, we first survey existing methods of modeling interactions and their corresponding visual results. These will help inform the design decisions of our Chronicled Interactions model in the next chapter. We broadly look at literature from the systems, human-computer interaction, and cognitive science communities for ways to design and reason about interactive visualizations.

2.1 Event streams

One approach to working with interactions is to model them as event streams. Each type of user input represents a separate stream, so for example there can be a mousemove stream, a click stream, and a keypress stream; and different interface elements can have their own separate event streams. Thus, the actual mechanism for producing the interaction can be abstracted away as purely an event source in a dataflow architecture. See Figure 2.1 for an example event stream diagram.

This approach is adopted by Reactive Extensions [28] and Cycle.js [11] to build reactive user interfaces that can automatically propagate user input events to the appropriate application functions which re-render the UI. These frameworks are built off of functional reactive programming principles as a way to control asynchronous event handling. The Vega runtime [35] also incorporates this idea and allows for declaratively performing operations on streams of input events. Vega is able to create interactive visualizations through a declarative specification without the need for imperative event handling logic.

The reactive dataflow programming paradigm has been previously explored in other languages such as Haskell [29]. It offers a powerful method for composing programs as a directed operation graph. Data flows through the graph and can be transformed at each operation. The user interface rendering can also be modeled as an operator in such a graph. It processes a stream of render events which are mapped to visual updates

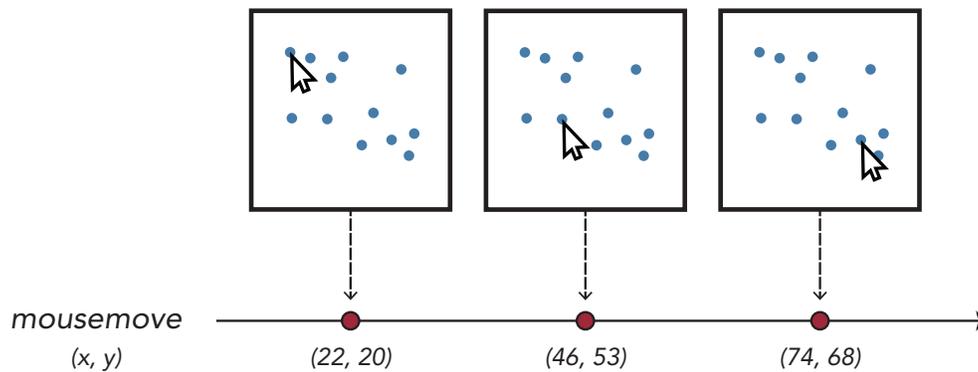


Figure 2.1: Mousemove event stream. Upon each movement of the user’s mouse cursor, the stream propagates an event with the (x, y) coordinates of the cursor position. This stream can be fed into a reactive dataflow architecture or mapped to other streams for further processing of the events.

on screen. The developer defines the operators, and the execution environment streams data through them at runtime.

Van Wijk explored a similar model of interactive visualization as an operation graph [41]. A key difference is that in his model, interactions only modify the visual specification of how data gets mapped into visual marks. We consider this a subset of interactions, and also consider cases where interactions can modify the actual data being visualized. This is the case when an interaction issues a new query for data. For example, when a user pans a map and new data points are drawn according to the new region of interest. In all future discussion of interactions, we consider interactions that affect both the visual specifications and underlying data.

2.2 Human-in-the-loop distributed systems

Typical distributed system models only consider the actual machines communicating over the network, while abstracting away the human user as part of the client machine. However, we can adopt a human-in-the-loop view which considers the user as a node in the system much like any other machine node as in Figure 2.2. The user sends requests to the client machine in the form of interactions and receives responses through visual updates to their screen. And so we can model the user in the context of the whole distributed system, with consideration of how communication is handled between the user and rest of the system. This approach is helpful for considering the needs of the user and applying systems ideas to explain user behaviors. For example, consider the following sequence of events that occur during a single user interaction:



Figure 2.2: Human-in-the-loop view of an interactive visualization system

1. User Joe hovers over a data point A through a mouse interaction, represented here as a “request” from Joe to the client interface.
2. The client receives Joe’s request via a mousemove event handler and processes it by
 - a) immediately responding back to Joe’s request, represented here as an update to the visual marks on the screen (e.g. showing a loading indicator)
 - b) sending a new request to a backend server for more data, parameterized by the id of the data point A which Joe selected
3. Joe receives the immediate visual feedback that his request is being processed
4. The backend server receives the request for data on point A, processes it (potentially requiring even more requests to other remote machines), and then sends back a response
5. The client receives the server’s response, processes it, and then sends another message to Joe by updating the visual marks on the screen (e.g. rendering a new visualization)
6. Joe receives the new information, processes it, and performs a new interaction (back to step 1)

This short example sequence demonstrates the inherent difficulty and complexity of designing an interactive user interface that supports large-scale data exploration. Multiple points of communication between the user, client, and server means that a robust visualization should mitigate failures and handle them appropriately if they occur. Speaking only from the client machine’s perspective, it must carefully keep track of internal state and know how to respond to messages both from the user and the server. Handling this logic and carefully managing the state can become a challenge for the developer working on the visualization software. If not handled properly, subtle bugs are likely to be introduced in complex applications.

For each interaction the user performs, there are actually two responses that they receive. One is the immediate visual **feedback** that acknowledges their request, and the

other is the intended visual **effect** upon receiving data from the server. By decoupling these two, we make an important distinction that many interfaces fail to address. Interfaces that lack proper visual feedback mechanisms such as loading spinners or progress bars can leave the user confused about the state of UI. Such confusion can often lead to the user performing multiple repeated interactions such as repeatedly clicking a button when unsure if previous clicks were being registered and processed. If left in a state of confusion for too long or if the interface actually does become broken from a subtle bug, the user may become frustrated and restart their entire interactive session.

An interaction can also initiate more than one request. For example if the user pans across a map, the client machine could send one request for additional data to load the underlying geographic map, as well as another request for data points within the specified area of the map which are rendered as a layer on top of the map. If the request for the data points finishes before the request for the mapping data, then the developer must decide whether it makes sense to render the data points without the context of the map or to delay the rendering until the mapping data is also received.

Furthermore, consider the fact that within a single interactive session, a user can issue multiple interactions concurrently while other results are still processing. In the mapping example, the user could pan across the map multiple times in quick succession before any data points load. In which case the client should ignore the previous requests for data, and render the response data for only the region of the map which is actually being viewed.

Though not a focus of this work, the model described here can be extended to support real-time streaming updates represented as data being sent from the server irrespective of any requests from the user or client. Therein lies another set of challenges of effectively communicating updates to the user with real-time constraints.

2.3 Visual encodings

So far in this chapter we have examined models of the user interaction process with examples that are grounded in interactive visualization interfaces. However, these interaction models are applicable to any interactive user interface in general. So to better motivate the design of a new model for interactive data visualizations, we now spend the rest of the chapter discussing models focused specifically around data visualization.

Data visualization can be described as the process of encoding data into visual marks. A visual mark is the basic unit of a visualization, such as a point, line, or area. Variables of the data are encoded as visual properties of the marks called visual variables. Examples of commonly used visual variables are position, size, and color. These visual marks can then be decoded by a human's perceptual system to read the data. Figure 2.3 presents a diagram of this model. The process of perceptual decoding is imperfect, and is highly affected by both the visual encodings used and the user's own percep-

tual idiosyncrasies. We discuss the cognitive aspects of reading and interacting with a visualization in the next section, but for now focus on the aspects of visual mapping.

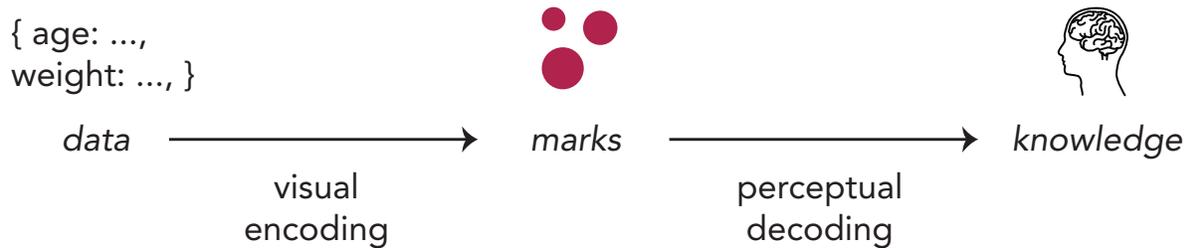


Figure 2.3: The visual encoding and decoding process

In his book, *Semiology of Graphics*, Jacques Bertin describes cartographic visualizations as the composition of marks, which serve as the basic unit of visualization, as well as visual variables which can vary the properties of those marks (e.g. size, color, shape) [4]. His initial list of seven visual variables was later expanded upon by Mackinlay et al. [25], who provided a ranking of their effectiveness for various perceptual tasks in order to guide visual design recommendations. More recently, variables such as time, motion, 3D, and transparency were introduced, made possible through the use of computers and animated graphics for visualization. Beyond Mackinlay et al.'s work, there has been a large body of research over the years which seeks to understand which visual variables are more effective for perceptual accuracy in a variety of tasks [9, 20, 37]. Understanding the effects of different visual encodings is important for designing visualizations that are usable and effective at conveying information. This is especially true in settings when visualizing multi-dimensional data where a separate visual encoding must be used for each dimension of the data.

2.4 Cognition

The process of viewing and interacting with a data visualization can be further examined through a cognitive lens to better understand usability constraints and human difficulties when using a visualization. This is critical for designing human-in-the-loop data analysis systems so that failure cases and human error can be mitigated. There is a variety of cognitive science literature focused on the areas of perception and memory which can be relevant to interactive visualizations. By modeling user behavior when interacting with a visualization, we can design better interfaces that leverage the user's cognitive abilities.

As described in the previous section, the process of visually decoding a visualization inherently relies on the user's perceptual system to read an accurate value from the

marks on their screen. This can be a difficult or easy task depending on the type of visual encoding used. Cleveland et al. studied perceptual accuracy rankings for reading different types of charts such as aligned bar charts, stacked bar charts, and pie charts [9]. Their findings showed significant differences in users' abilities to read certain charts and accurately describe their encoded values. This motivates careful consideration for designing visualizations, which should ideally leverage encodings with high perceptual accuracy.

Change blindness is the perceptual phenomenon of humans being unable to perceive a visual change, even if the change is not very subtle. Applied to an interactive visualization setting, this can happen when there are many updates that are happening too quickly or to different parts of the screen. It can be difficult for a user to focus their attention on multiple coordinated views and be consciously aware of every change that occurs on screen. Finding methods to counteract change blindness is an active area of research that can be applied to visualization designs [31]. For example, one such method is to leverage pre-attentive visual attributes. These are attributes which human eyes are able to notice typically within less than 200-250 ms [17]. The use of pre-attentive visual attributes such as hue, size, and orientation, removes the need for humans to perform focused visual attention processing on areas of the screen, lessening the time required to absorb information and increasing the likelihood of noticing changes of a visual interface.

In an interactive environment with latency, it might be difficult for humans to understand how their interactions affected the results on screen if it takes multiple seconds for interactions to be processed. Referred to as the "gulf of evaluation", the user has difficulty determining the state of the system when no visual feedback is given in response to their interactions [30]. Hutchins et al. advocated the use of direct manipulation to lessen this gulf by directly fulfilling user's expectations of how their actions are interpreted by the interface [22]. While direct manipulation may seem impractical in an interactive setting with latency, there are other methods which can be used to help reduce the gulf of evaluation. For example, visual feedback mechanisms such as progress indicators can be displayed when interactions are being processed so that the user knows their actions have been acknowledged by the system.

While most cognitive information visualization research has focused on the topic of human visual perception, the emergence of highly interactive visualizations makes it relevant to look at a broader scope of HCI research. Card et al. defined the Model Human Processor in 1983 as a method for modeling the time spent by a user performing a cognitive task, broken down by perception, memory, and motor functions [6]. This model is useful for estimating how a user might interact with a visualization. For example, their experiments suggest that an average human working memory can hold about 7 chunks of information with a half-life decay of about 7 seconds. When interacting with a visualization, a user typically employs their short term memory. In an exploratory data analysis setting, the visualization that a user interacts with can change quickly over the course of an interaction sequence and so it is likely for the human memory to fill up.

Unlike computer memory, human memory is limited and so visualization tools can account for this by reducing human memory load when possible, for example by keeping important data visible on screen.

Hollan et al. introduced the Distributed Cognition framework in 2000 for reasoning about the cognitive effects of embodied agents interacting within an environment [21]. Their work explores multiple ideas of how to design tools in the context of a networked and distributed environment. Relevant to the study of interactive visualizations is their notion of history-enriched digital objects and effective use of spatial layouts. They argue that applications which encode the history of past use can facilitate interaction similar to affordances in the physical world such as physical wear of a tool. The more recent work on HindSight [13], a system for encouraging exploration by visualizing past interaction history, embodies this notion. The Distributed Cognition framework also argues that effective use of space can reduce the time and memory demands of certain tasks, as space can be used to encode ordering and off-load human memory. Their primary examples focus on the physical world and spatial arrangement of objects in a 3D environment, so there is an interesting question of whether these ideas can be effectively applied to digital interfaces. We explore this idea further in Section 3.2.

Chapter 3

Chronicled Interactions

In this chapter we introduce the main contribution of this thesis, Chronicled Interactions, a new model for designing *asynchronous* interactive visualizations. This model builds upon ideas discussed in the previous chapter for designing interactive visual interfaces, while introducing new ways to express interactions specifically in the context of latency. The main idea behind Chronicled Interactions is to store and visualize a history of past interactions and their corresponding results, which allows the user to execute interactions concurrently while reducing their cognitive memory load. This chapter details the different aspects of the model starting with a description of the data and visual layers, followed by methods of corresponding the two layers. The visualization designs produced by this model can be demonstrably more effective at targeting large-scale exploratory data analysis problems. The following chapter evaluates the effectiveness of Chronicled Interactions through user experiments incorporating different visual designs from the model design space.

The primary motivation for defining a new model stems from the lack of readily usable means of modeling asynchronous interactions in a visual interface. Traditional methods assume all interactions are resolved immediately, whereby their effects are rendered on screen and no additional loading happens. This is an inaccurate depiction of current data analytics scenarios which require that interactions are resolved after some amount of latency due to data querying and processing times. These scenarios introduce new challenges for the designers of highly interactive visualizations who now need to handle complex asynchronous event handling and potential consistency issues. In addition, new cognitive challenges arise for users of such visualizations who must now endure longer wait times per interaction and can have more difficulty in understanding the state of a continually changing interface.

3.1 Data layer: history of interactions

Chronicled Interactions is comprised of two important abstractions. The data layer describes how interactions and their corresponding data updates are modeled, while the visual layer describes how this data is visualized. The data layer is purely responsible for managing interactions and their responses, irrespective of how these interactions, responses, or any intermediate state is visualized on a user's screen. By introducing these separate abstractions, we are able to build a robust method for dealing with asynchrony: manage a history of interactions and their responses over time, and have methods of visualizing that history as it changes over time.

Interaction History

History of interactions has been well studied as a tool for understanding and making use of interactive processes [13]. Heer et al. [20] provides a description of the design space of graphical histories to support navigation and exploration in visualization interfaces. Chronicled Interactions requires a model of interaction history and methods to visualize it so the user can track their recent results and enable more concurrent interactivity.

As previously discussed in Section 2.1, all interactions can be modeled as a stream of user input events. A single unit of interaction is a user event that causes a *data update*. The data update is a request to a server whose response results are used to modify the underlying dataset of a visualization. In practice there are often well-defined semantics for an interaction, as seen in research projects investigating latency [24, 15]. For example, when clicking on a data point in a chart, a new query for data is processed which is parameterized by the coordinates of the data point selected. The interaction can be translated directly into a SQL query or any other well-formed request to an API endpoint that serves data. Thus the design of Chronicled Interactions is backend-agnostic, which is useful for directly applying it to any application.

This stream of interaction events has a strict total order, which is defined by the time at which each interaction occurred. However the corresponding responses to the data updates for these interactions can occur in an arbitrary ordering, and in some cases may even fail. The ordering of responses is dependent upon the latency of each data update, and some updates may have higher latency than others. This can be caused by several different reasons such as varying query workload, network speed, dataset size, or cache misses. Case 4 of Figure 3.1 demonstrates how responses can be rendered out of order under varying latency.

These potential re-orderings of responses can be resolved by associating them with their corresponding interactions. Using the order that the interactions occurred, the responses are stored and versioned such that the responses can then be later differentiated.

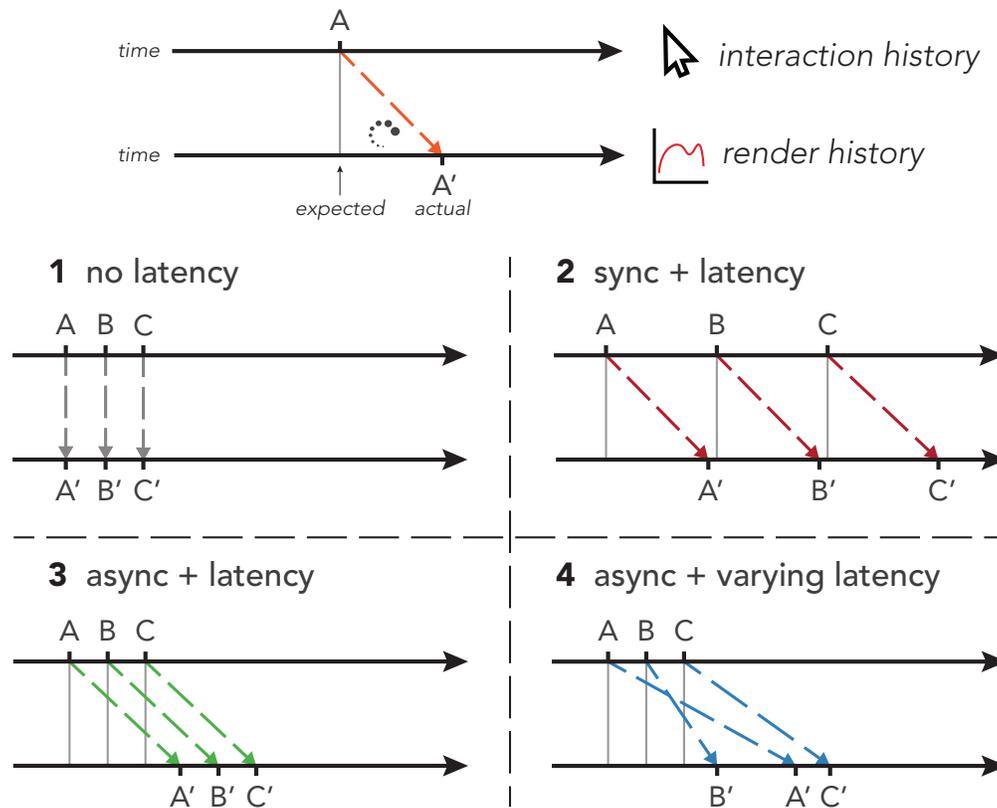


Figure 3.1: A sequence of interaction requests and responses under different conditions visualized on a horizontal time axis. Colored arrows represent request/response pairs over time. Light vertical lines highlight request times. (1) The ideal no latency scenario commonly assumed by visualization designers. (2) With latency, the user waits for each response to load before interacting. (3) With latency, the user interacts without waiting. (4) With varying latency, the user interacts without waiting and may see responses in a different order than requests were issued.

Data Versioning

When a data update is resolved, its results are used to update the underlying dataset of a visualization. As described in Section 2.3, conceptually the visualization maps values in the data domain to marks in the visual domain. Section 3.2 provides details of this mapping for Chronicled Interactions. Upon each new update, this dataset is simply augmented with a new value in a separate dimension which denotes its version. In essence we are now versioning the underlying dataset, and specifying these versions by interaction history. See Table 3.1 for an example.

year	price	⇒	year	price	version
2015	10		2015	10	v1
2016	20		2016	20	v1
2017	50		2017	50	v1
year	price		year	price	version
2015	10		2015	20	v2
2016	20		2016	30	v2
2017	50		2017	20	v2

Table 3.1: Adding History Dimension to a Dataset

Buffer Management of History

Rather than store all of a user’s interaction history, we can maintain a buffer of only the *most recent* interactions and data updates. Providing a bound on the history stems from a variety of design and implementation constraints.

From a cognitive perspective, a user may be able to view and process the results from a few interactions at a time, but the difficulty of understanding and making comparisons among the results increases as more interactions are displayed. Assuming the user iteratively updates their data exploration strategy over time, it makes sense that they care most about the most recent results which were created as a result of their own interactions. As time progresses, users may not be able to keep track of multiple versions and so it is natural to cut off the history.

From a visualization perspective, a simple line chart may be capable of rendering 5 distinct versions of a time series, but it may not be able to effectively render 50 versions, e.g. due to over-plotting points on the same space. While there exist various techniques for reducing the over-plotting problem such as jittering and transparency [14], these can only extend the amount of versions visible on screen. There still exists a physical limit to how much data can be displayed effectively.

From a systems perspective, the resulting data of these interactions can be quite large in memory size, and thus can easily run up against the client machine’s memory limits if stored indefinitely. Thus, we capture this limit via a sliding window semantic on the interaction history and keep a relevant buffer of interactions, which helps guarantee that the complexity of the resulting visualization is bounded.

When maintaining a fixed set of interaction history, a new interaction being pushed into a buffer that is currently full must evict an existing interaction. A simple strategy is to evict the oldest interaction immediately upon user interaction, or evict upon the data update for the new interaction having been resolved. Other buffer management policies could also be implemented, with varying effects on the end visualization.

The notion of a history buffer can also be used for expressing traditional interfaces that show only one result at once. Such an interface can be expressed as having a

buffer size of exactly 1, where only the result of the most recent interaction is stored and managed. Chronicled Interactions extends the size of this buffer, and the history can contain arbitrarily long sequences of interactions. An unbounded history could even store every single interaction and corresponding result in an interactive session. However for practical implementation purposes, a bounded buffer on the history should be used. In Section 4.2, we evaluate the effects of multiple different buffer sizes on a set of visual analysis tasks.

3.2 Visual layer: visualizing history

Given our model of interaction history and the associated versioned data updates, we now take a look at how to expose this history to the user interface through visualization. Properly communicating the state of the interface and leveraging the user's cognitive capabilities can improve the interactive visualization experience in the presence of latency.

Visualizing Interaction History

In our framework, we model visualizations purely as a functional mapping of some dataset to visual components on screen. These visual components can be formed by arbitrary mappings, but it is helpful to think of them in terms of common charts and layouts. Interactive visual components that accept user inputs are referred to here as *widgets*. Interactions on a widget are captured and processed in the data layer as described in the previous section. A widget can either be an interactive chart or a separate component such as a menu which is not meant to visualize data. In the latter case, a widget is generally paired with a corresponding chart which gets updated in response to interactions on the widget component. An example of this can be seen in Figure 3.2.

The only significant requirement for the chart and widget to properly expose a user's interaction history is that they need to be able to visualize another dimension: the version history. For an example, see the mapping in Figure 3.2. The chart introduces color as a third visual encoding on top of its existing positional encodings to represent the version history. Both the input widget and the output chart need to encode the history of interactions and results. In some ways widgets can be more constraining than charts due to a lack of viable visual encoding channels. For example, a more difficult interactive widget might be a checkbox, where it becomes unclear how one would easily visualize the version history of past clicks.

Techniques for Visualizing Versions

Chronicled Interactions requires visualizing the new versions of data without changing existing visual encodings. This could be achieved by adding another visual encoding channel that is not currently used by the existing visualization. Edward Tufte explored

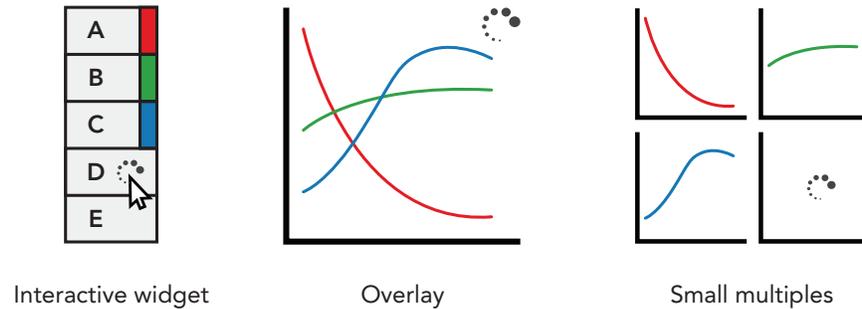


Figure 3.2: Different visual techniques for displaying multiple versions of a dataset at the same time: overlay (middle), and small multiples (right). Visual correspondence is achieved through a shared color encoding on the interactive widget (left) and display charts. All the UI elements are able to indicate whether more data is currently being loaded through the use of a loading spinner.

the idea of viewing multiple connected images through his notion of visual parallelism. In his book, *Visual Explanations*, Tufte describes two different methods for leveraging visual parallelism using either space or time [39]:

Spatial parallelism takes advantage of our notable capacity to compare and reason about multiple images that appear simultaneously within our eyespan. We are able to canvass, sort, identify, reconnoiter, select, contrast, review—ways of seeing all quickened and sharpened by the direct spatial adjacency of parallel elements.

Parallel images can also be distributed temporally, with one like image following another, parallel in time. For time-bound sequences, comparisons must be made by contrasting a remembered image with a currently viewed image, a sometimes difficult task.

We can apply these ideas of visual parallelism to help broadly classify different techniques for displaying multiple datasets in a single visualization. Here we describe in detail three common categories of techniques which encode multidimensional data through an additional visual variable:

- **Small multiples** is an established visualization technique that draws different graphs corresponding to each slice of data to reveal change and patterns. In Tufte’s explanation of spatial parallelism, he explicitly refers to using small multiples in space to effectively enable comparisons. Using multiples to represent history has been extensively studied in the past with favorable results for aiding exploratory analysis [20, 34, 12]. See Figure 3.2 (right) for an example.

- **Overlay** is a common technique used to perform comparisons across multiple data series by overlaying them on top of each other. Tufte describes this technique as “superimposed parallelism, directly overlapping rather than side by side”. See Figure 3.2 (middle) for an example. Overlay makes effective use of space by leveraging an additional encoding such as color to differentiate different sets of data. This technique can only work well for certain types of charts. For example, a line chart or scatterplot is a natural fit for using overlay, but a pie chart or bar chart may require modifications such as stacking in order to make use of overlay.
- **Animation** is a technique that has become more common with the advent of highly interactive visual interfaces. Whereas the previous two techniques leverage additional space on the screen to render multiple versions of data, animation renders different versions at different points in time.

3.3 Correspondence: mapping interactions to results

While the previous section described methods for visually differentiating multiple versions of a dataset in the same visualization, in order for the user to understand which data corresponded to an earlier interaction, the correspondence must be explicitly visualized. In the scenario where the interactive widget and chart are separate UI elements, this means the widget and chart must share a common visual encoding to correspond the interaction with the result. Figure 3.2 presents an example using color.

This visual encoding can be used to match the sequence of interaction history to response history. Explicit visual correspondence is important because it can be unclear which results on screen correspond to the user’s earlier interactions when there is latency and possible reordering. Furthermore, the shared encoding must be unique across the UI to prevent mistaken inferences of correspondence and confusion for the user.

When choosing the visual variable for encoding interaction history, it is important to decide whether the variable should support an ordered reading. An **ordered** visual variable allows for perceiving the order in which interactions occurred, whereas an **unordered** one does not. Figure 3.3 shows two different color encodings that could be used for visually representing history. In the ordered case, darker colors could be used to represent more recent interactions, with the least recent ones appearing lightly faded as they are less likely to be relevant to the user. In the unordered case, each interaction is represented with a distinct color which makes differentiating between them easier, but the ordering is lost.

Comparisons between ordered and unordered encoding types have been well studied [25, 38], and ordered encodings of interactions have proven valuable in previous data exploration designs [13]. However, while it might seem beneficial to always use an ordered color encoding, we have found that this might not always be the case. Some visual analysis tasks such as trend detection can make use of the visual ordering, whereas for



Figure 3.3: Two possible color encodings used for visualizing interaction history. The left is an ordered sequential encoding of blue values, and the right is an unordered categorical encoding of multiple color values. These palettes were selected from ColorBrewer [16].

other tasks the ordering may be irrelevant. For example, consider the case when a user is exploring multiple subsets of data to find out whether any subset satisfies some condition. This is a common scenario when investigating multiple regions of a map, locating outliers among many points, or aggregating results from multiple sources. Such a task may not require investigating the data in a certain order, and may instead require being able to distinguish the data from one another. In such cases, an unordered encoding can actually make the task easier. In Section 4.2, we evaluate the effects of ordered and unordered color encodings on different visual analysis tasks.

3.4 Summary

Chronicled Interactions manages and visualizes user interaction history to support asynchronous interactive visualization. Showing graphical histories was motivated by existing visual interfaces that effectively leverage history to aid data exploration. Users can perform multiple asynchronous interactions concurrently and process results as they arrive, reducing wait times caused by latency. This enables quicker iterative analysis by shortening the feedback loop between the user and the interface. Furthermore, visualizing multiple results at once lets users easily make comparisons, and alleviates the need for them to remember previous results or open multiple sessions of the same visual interface. We summarize here the three main aspects of the model:

1. **Data layer:** Store and manage a buffered history of interactions and their corresponding results. The history is modeled as a multi-versioned dataset. Even when results are received out of order due to varying latency, they can be assigned versions according to the order of the original interaction sequence.
2. **Visual layer:** Visualize interaction history using available encoding channels. Existing methods for visualizing multidimensional data such as small multiples, overlay, and animation can be used. Both the interactive widgets and chart displays should be embedded with the visual interaction history.

3. **Correspondence:** Establish visual correspondence of interactions and results via a shared encoding of interaction history on both charts and widgets. The choice of encoding can be visually ordered or unordered depending on the analysis tasks supported.

Chapter 4

User Experiments

In this chapter we evaluate the effectiveness of the Chronicled Interactions model by testing a set of visual designs expressed by different design parameters of the model. The goal was to understand whether the model is useful for designing interactive visualizations that encounter latency, and measure any tradeoffs in the design parameters. We conducted a set of online user studies in which participants were tested on a variety of visual analysis tasks, and found that user performance improved overall with the Chronicled Interactions designs compared to a traditional interface. These results suggest that large-scale interactive data visualizations can be improved by leveraging interaction history.

4.1 Spatial vs. temporal designs

We designed an initial pilot study meant to explore the different methods of visualizing interaction history as described in the previous chapter. Specifically, the study focused primarily on animation as a means of displaying history, while including a small multiples design for comparison. Users were presented with a simple interactive visualization of a line chart and asked to perform various visual analysis tasks. Conditions of latency were simulated upon each interaction.

The pilot study found that animation did not significantly improve a user's ability to perform various visual analysis tasks compared to a traditional interactive visualization that can show only one result at a time. Users would often wait for the results after performing an interaction, and repeat until fulfilling the task. These findings are consistent with previous research on the cognitive issues underlying the effective use of animation for perceptual tasks [39, 34, 40]. In general animation appears to be most effective as a presentational tool, and when performing exploratory analysis it can be difficult to perceive changes as they happen very quickly. Furthermore, animation relies heavily on the user's working memory, since they must remember previous frames in order to perform comparisons.

In contrast, the small multiples design showed a significant effect on user behavior. Under conditions of latency, users would perform multiple interactions concurrently and analyze the results after they all loaded. This behavior was possible because the user could view multiple data series on screen at once. This finding led to the insight that spatial parallelism might be an effective means of visualizing interaction history, an idea that we decided to further explore in a full user study described in the next section. These findings are consistent with previous research on Distributed Cognition which suggests that effective use of space can be a tool for off-loading human memory workloads [21].

4.2 Experiment design

The experiment used a 4 (buffer size) \times 2 (order) \times 3 (task) \times 3 (latency) \times 2 (encoding) mixed factorial design. The between-subjects parameters were the task and encoding (small multiples, overlay). The within-subjects parameters were the buffer size, order, and latency distribution as described below:

- **Buffer size:** The number of recent interactions stored and visualized on screen. The four values tested were **1**, **4**, **8**, and **12**. **1** is the baseline used in traditional visualizations, and **12** is the total number of facets in the experiment visualization. The other two values were chosen by a linear interpolation between the two extremes. For the rest of this paper we will refer to buffer size of **1** as baseline, **4** as small buffer, **8** as medium buffer, and **12** as large buffer for the ease of discourse.
- **Order:** Whether the visual variable for encoding interaction history supports ordered reading. An **ordered** visual variable allows for perceiving the order in which interactions occur, whereas an **unordered** one does not. Since the visual correspondence encoding for the experiment was color, two different ColorBrewer palettes [16] were chosen: sequential shades of blue for ordered, and discrete colors for unordered. Figure 3.3 shows the different encodings used for the experiment.
- **Latency:** The distribution of the amount of time to process each interaction. Three different latency distributions were tested. The first is the **none** latency case where each interaction is processed immediately. This was chosen as a baseline to emulate existing interactive visualizations in small data scenarios. The second is the **short** latency case where latency for each interaction is drawn uniformly at random between 0 and 1 seconds. The uniform distribution was chosen to simulate some amount of variance, which can happen under multiple scenarios as described in Section 3.1. The average latency of this distribution is aligned with the latency introduced in previous experiments [24]. The third is the **long** latency case where latency for each interaction is drawn uniformly at random between 0 and 5 seconds. The range for this latency distribution corresponds with benchmarks for

streaming workloads in production scenarios [8]. In all cases the latency was simulated upon each interaction, so no actual network requests were made to prevent unpredictable network interference from polluting the experiment setup.

The experiment design parameters are summarized in Table 4.1.

buffer size	order	task	latency	encoding
1	ordered	threshold	none	overlay
4	unordered	maximum	short	small multiples
8		trend	long	
12				

Table 4.1: Experiment Design Parameters

For the experiment visualization, we created an interactive faceted line chart representing stock prices shown in Figure 4.1. Faceted visualizations are common in analytics dashboards [7, 10, 33, 44], and the specific interactive visualization chosen is versatile and easily generalizable. For instance, the line chart could be changed to a bar chart, area chart, or scatter plot, and the month selector widget could be changed to a slider or a dropdown menu. Furthermore, time series data is a natural candidate for visual analytics, since it appears in nearly every domain and has been extensively studied [1, 34]. The experiment setup visualizes stock price data, which is easy to understand and familiar to a wide audience. The actual data that the users interacted with was fictitious and randomly generated based on real stock data. This was important in preventing any prior knowledge of real-world stocks from affecting how participants responded to the tasks.

We tested three simple tasks that are similar to those used in previous visual analysis experiments [1, 34]: maximum, trend, and threshold. These tasks were selected based on two criteria. The first is ecological validity; these tasks are appropriate because they model real-world visual analytics tasks such as outlier detection, pattern recognition, and trend detection. The second criteria is variety; these tasks pose different memory, correspondence, and ordering requirements. This allows for drawing insights into how latency affects tasks based on their specific cognitive difficulties. We describe the three selected tasks below, along with how each task was phrased in the context of the experiment visualization shown in Figure 4.1.

- **Maximum:** Identify the maximum value. This requires the participant to remember the largest value seen so far, and the corresponding month.
“Which month had the highest stock price for the year 2010?”
- **Trend:** Identify a trend across multiple interactions. This requires the participant to read the values and remember their order.
“What is the trend in stock price from Jan to Dec for the year 2010?”

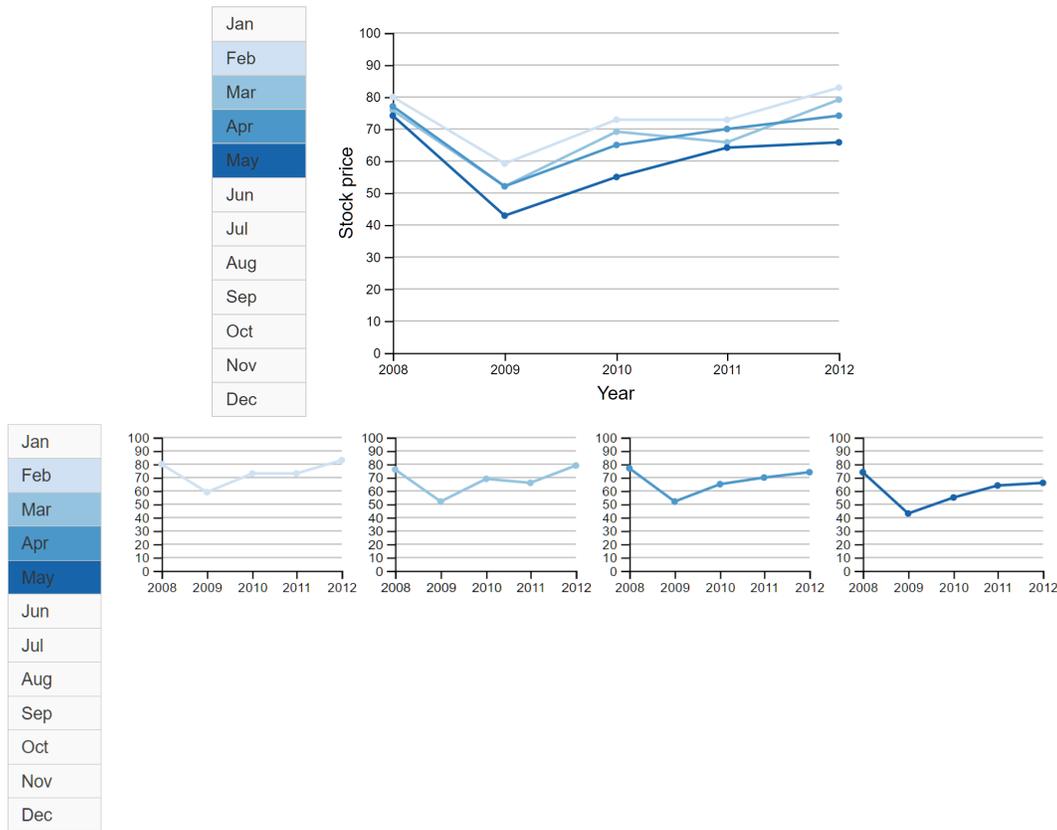


Figure 4.1: Two separate experiment visualization setups featuring a faceted line chart visualization representing stock prices for the years 2008-2012, split by month, with small buffer (size 4), and ordered color encoding. (top): overlay design, (bottom): small multiples

- **Threshold:** Identify if any value satisfies a condition. This requires the participant to scan each value of the chart to check if any value crosses the threshold.
“Does any month have a stock price higher than 80 for the year 2010?”

Data Generation

We generated 29 datasets per task type, 5 for training, and 24 for evaluation. Each dataset contained 70 numbers (5 years \times 12 months) between the values 5 and 95. To ensure that the setup of each task was comparable, we normalized raw stock data [32] and made specific modifications to the data based on task:

- **Threshold:** For 50% of the datasets, exactly one month crossed the threshold, otherwise none did.

- **Trend:** Generated simple, smooth sequences that follow one of the following four patterns: (1) increasing, (2) increasing then decreasing, (3) decreasing then increasing, and (4) decreasing.
- **Maximum:** The maximum value was greater than all the others by at least 5.

The datasets were randomly generated offline, and randomly matched to different experimental conditions for each participant.

Procedure

Study participants were recruited online using Amazon Mechanical Turk, a popular experiment platform for visualization studies [18, 37]. All participants were US citizens above the age of 18. Participants were rewarded \$0.10 per task within an experiment, and a \$1.00 bonus for completing all tasks. Participants were allowed a maximum of 60 minutes to complete the experiment. 50 participants were recruited for each combination of between-group parameters, for a total of 300 ($50 \times 3 \text{ task} \times 2 \text{ encoding}$) participants. The participant population was 39% female and 61% male. 57% of participants held a college degree or higher. The average age of the participants was 35 years old. At the beginning of each experiment, the participant was randomly assigned to one of the 6 different groups which determined the task and visualization that they would be exposed to throughout the experiment. Each experiment consisted of the participant going through the following procedure in order: training, tasks, and survey.

Training

Before beginning the experiment tasks, participants were first instructed on how to read and interact with the visualization. They were presented with a single buffer visualization with no latency, then one with short latency, and then with increased buffer sizes. The same dataset was used throughout to ensure that participants focus on the change in the conditions. After becoming familiar with the interactive visualization setup, the participants watched two short videos demonstrating contrasting interaction behaviors: one where the user would serially interact and look at results, and one where the user would interact concurrently and then look at all the results. Participants were recommended to try interacting concurrently to reduce the time spent waiting for results to load.

Afterwards, participants were presented with one task out of the 3 different task parameters, determined by the experiment group of the participant. This task remained the same throughout the experiment for each participant. After completing the task and submitting their answer, the correct answers were shown for comparison. Participants completed four training tasks under different conditions: (1) no latency, single buffer, (2) short latency, single buffer, (3) short latency, ordered, small buffer, and (4) long latency, unordered, large buffer.

The inclusion of training tasks was meant to validate the user's understanding of the tasks and prevent invalid responses. This technique has been successfully used in other online studies to improve response quality [23]. This can also counter any initial learning effects that might occur when the participant is familiarizing themselves with the visualization interface. Showing the user the correct answer after their submission allowed for them to correct for any mistakes and adjust their understanding of the visualization. The answer submissions to these tasks were recorded, but not included for the purposes of analysis.

Tasks

Each participant completed one task for each combination of within-subjects parameters, for a total of 24 (4 buffer size \times 2 order \times 3 latency) tasks per participant. The tasks were randomly ordered, so participants did not know the buffer size, order, or latency parameters of the task beforehand. No time limit was imposed per individual task, though participants were advised to complete tasks as quickly as they could to get an accurate assessment of task completion time. The tasks followed the same setup as in the training phase, except participants were not shown the correct answer before moving on to the next task.

Survey

To assess the usability of different visualization designs beyond task performance, users were asked to complete a survey at the end of the experiment after finishing all the tasks. The goal of the survey was to gauge the perceived usability and preferences for different designs among the study participants. The questions were presented as 5-point Likert scales asking about (1) what buffer size they preferred, (2) task difficulty with latency, and (3) whether they preferred the ordered or unordered color encoding. Additional space was left at the end for open-ended comments about their experience. These helped form valuable insights into user interaction strategies and explanations for user behavior during the experiment analysis.

4.3 Experiment analysis

In this section we report the findings of our experiment analysis based on the experiment design described in the previous section. We form four separate hypotheses which test the effectiveness of Chronicled Interactions designs both from a task performance and usability perspective. We then describe the methods of analysis and statistical tests used to verify the hypotheses. Results are reported with accompanying figures and a discussion of the key findings.

Hypotheses

- **H1: Users demonstrate a similar level of accuracy when using either the Chroni-cled Interactions design or traditional visualization.** We expect that Chroni-cled Interactions are intuitive to use and do not cause confusions.
- **H2: Users are more efficient using Chroni-cled Interactions when there are long latencies.** Specifically with regard to the design parameters discussed in Section 4.2, we expect that (1) increasing the buffer size lets users complete tasks faster, and (2) ordered encoding will speed up tasks that are order sensitive, and unordered encoding will speed up the tasks that require correspondence of the visual responses.
- **H3: Users demonstrate increased concurrent interactions using the Chroni-cled Interactions design when there are long latencies.** We expect a negative correlation between degree of concurrency and task completion time.
- **H4: Users prefer a Chroni-cled Interactions interface over a traditional visualiza-tion.** We expect the participants to find the usability of Chroni-cled Interactions designs less impaired by latencies.

Measures

- **Accuracy:** whether a task response is correct. For the purposes of analysis, this was treated as a binary measure so that meaningful comparisons could be made across tasks.
- **Completion time:** total time to complete a task, in seconds.
- **Concurrency:** percentage of time during a task when there were multiple inter-action requests waiting for a response (as a result of latency). When participants serialize their interactions such that they wait for a response before making a new request, concurrency is 0. When the participant interacts with everything before the first response is loaded, concurrency is close to 1. We chose this measure as an approximation of the degree of asynchrony. Note that even with buffer size of one, it is still possible to have multiple requests waiting for a response, except only at most one response is shown. Concurrency is derived from a log of request and response data and timestamp.
- **Preference:** usability preferences measured by a 5-point Likert scale, the questions are described in more detail in Section 4.2.

Results

The participant responses were analyzed by performing pairwise comparisons across different within-group experiment conditions using the Wilcoxon signed-rank test, which is more robust to outliers and skewed distributions, while being almost as efficient as the parametric version when the underlying distribution is normal. For the survey results we use the one-sample version of Wilcoxon test. We report Wilcoxon T -value and p -value for the test, along with the medians of the two groups (M_1 for controlled baseline, and M_2 for treatment). The Holm-Bonferroni method is used to correct the significance levels when considering multiple hypotheses. We use the Hodges-Lehmann estimator to report population shift in the test as well when needed, abbreviated to hl.

Before analyzing the results, we performed a data cleaning step to filter out responses with low quality, as is common when running online experiments [27]. Examples of undesirable behavior include users who would submit the same answer for every task, or users who would take prolonged breaks during the experiment. Such behavior can skew the data, and cause inaccurate conclusions to be made. Because of the difficulty of enforcing behavior over an online experiment, we still collected these responses and filtered them based on common patterns.

We removed tasks completed by participants who got the majority of the tasks wrong (10%), tasks that took longer than two minutes ($< 1\%$), and responses where the participant did not interact at all with the visualization ($< 1\%$). The effect of pruning using 2.5 median absolute deviation had no significant impact on the analysis, so we did not perform further outlier removal.

Accuracy

Study participants completed all tasks with very high accuracy regardless of the visualization design. Only 6% of all 6,362 tasks were inaccurate, suggesting that all three tasks selected for the experiment were easy enough to complete for most users. The largest factor in task accuracy was the type of task itself; the threshold task showing the least amount of inaccurate responses and the maximum task showing the greatest amount. The absolute numbers are quite small in comparison to the total number of submitted answers, but Figure 4.2 shows the relative differences between number of inaccurate responses among different tasks and latency distributions. As expected, longer latencies had the effect of producing less accurate answers among study participants.

No discernible difference was found when comparing accuracy between different buffer sizes, designs, or encodings. This suggests that hypothesis H1 is correct. It would be interesting to study how accuracy might be affected among more difficult analysis tasks. It might be the case that accuracy could be improved with Chronicled Interactions designs for scenarios where the average accuracy is much lower. We leave this open as future work to be considered.

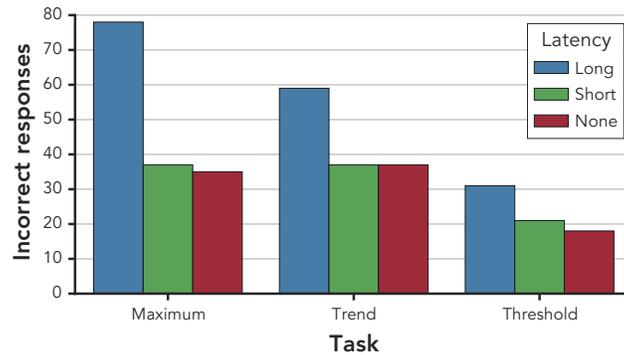


Figure 4.2: Number of incorrect responses by task and latency. All other conditions were pooled due to negligible differences across designs. The larger number of incorrect responses for long latency suggests latency can have detrimental effects on participant accuracy. The number of incorrect responses is reported because the vast majority of tasks were completed correctly.

Completion time

While accuracy remained consistently low across conditions, completion time varied significantly. Figure 4.3 summarizes the completion time results. Each plot compares the median task completion time to the buffer size for three latency conditions: no latency, uniform random latency between 0 and 1 seconds (short), and uniform random latency between 0 and 5 seconds (long). The plots are faceted such that the rows are split by encoding design: overlay (top) and multiples (bottom); while the columns are split by task: maximum (left), trend (middle), and threshold (right).

Participants completed tasks much faster with Chronicled Interactions when there is high latency i.e. increasing the amount of viewable interaction history correlated with lower task completion times.

Figure 4.3 shows that for all the completion time plots, under the long latency case users complete all tasks faster when the buffer size is greater than the baseline of 1. The test statistics are reported in Table 4.2. The test statistics for the overlay design suggest that users complete tasks faster with small buffer than baseline for all three tasks: maximum (top left, blue), trend (top middle, blue), and threshold (top right, blue). A similar behavior for the small multiples design is observed: maximum (bottom left, blue), trend (bottom middle, blue), and threshold (bottom right, blue). For each comparison, the p -value suggests a highly statistically significant outcome.

Under short latency, no significant completion time differences were observed between small buffer and baseline. The test statistics are reported in Table 4.3. Comparing large buffer with the baseline, we observe more significant differences for the overlay design on the maximum task (top left, green), and the threshold task (top right, green). Similarly for small multiples on maximum (bottom left, green), and threshold (bottom

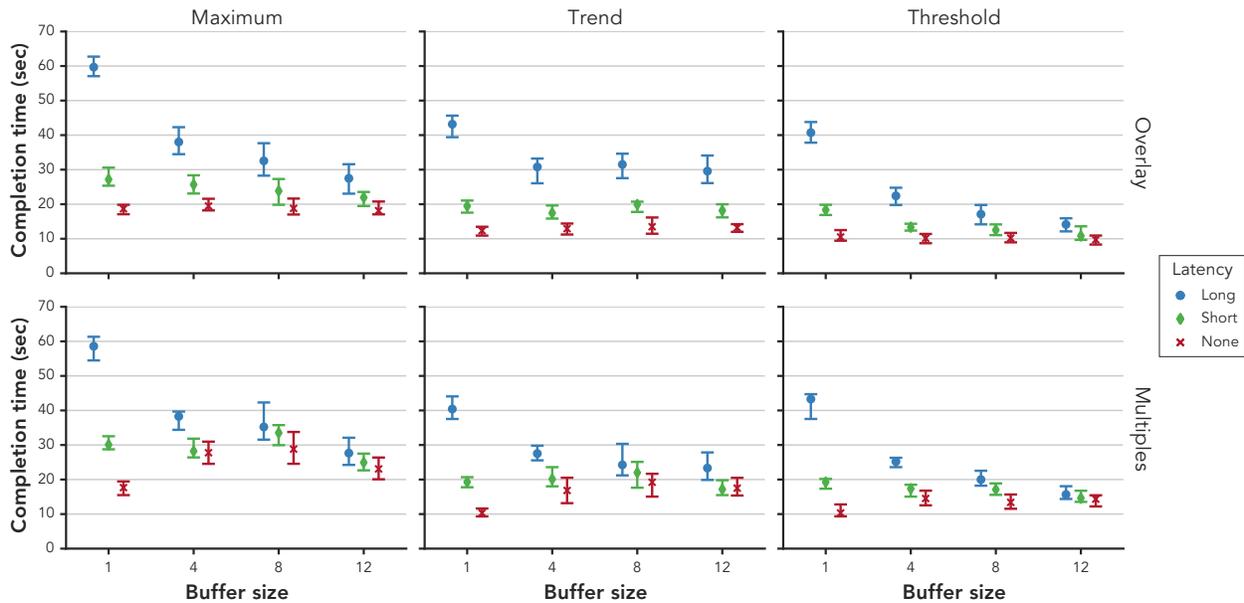


Figure 4.3: Each chart of the plot visualizes median task completion time with 95% CI (y-axis), for the conditions within an experiment group: buffer sizes (x-axis) with 1 as the baseline, and latencies (hue); ordered and unordered encoding was combined due to lack of significant difference. The charts are faceted by between-group conditions: interaction design across rows, and task type is plotted across columns.

Design	Task	Samples	Medians (sec)	Test statistics
overlay	maximum	38	$M_1 = 60, M_2 = 39$	$T = 83, p = 3.054e^{-5}$
overlay	trend	38	$M_1 = 44, M_2 = 30$	$T = 76, p = 1.947e^{-5}$
overlay	threshold	41	$M_1 = 42, M_2 = 25$	$T = 209, p = 4.101e^{-3}$
small multiples	maximum	30	$M_1 = 62, M_2 = 40$	$T = 30, p = 3.112e^{-5}$
small multiples	trend	39	$M_1 = 38, M_2 = 28$	$T = 150, p = 8.105e^{-4}$
small multiples	threshold	41	$M_1 = 45, M_2 = 26$	$T = 106, p = 2.611e^{-5}$

Table 4.2: Completion time statistics for **long** latency comparing between baseline and **small** buffer using the Wilcoxon signed-rank test.

right, green).

No significant difference was found for any of the tasks with the overlay design under the no latency condition. However, participants were *slower* using small multiples designs when there is no latency, indicating that the design is not a silver bullet that can improve all interactive visualization cases. The test statistics are reported in Table 4.4. In Figure 4.3, for small multiples on the bottom row, the red completion time plots corresponding to no latency increase as buffer size increases. Reporting on test statistics

Design	Task	Samples	Medians (sec)	Test statistics
overlay	maximum	39	$M_1 = 29, M_2 = 23$	$T = 159, p = 1.266e^{-3}$
overlay	trend	42	$M_1 = 20, M_2 = 19$	$T = 367, p = 2.907e^{-1}$
overlay	threshold	42	$M_1 = 18, M_2 = 12$	$T = 219, p = 3.648e^{-3}$
small multiples	maximum	36	$M_1 = 30, M_2 = 25$	$T = 256, p = 2.264e^{-1}$
small multiples	trend	46	$M_1 = 20, M_2 = 18$	$T = 386, p = 9.142e^{-2}$
small multiples	threshold	42	$M_1 = 19, M_2 = 14$	$T = 201, p = 1.735e^{-3}$

Table 4.3: Completion time statistics for **short** latency comparing between baseline and **large** buffer using the Wilcoxon signed-rank test.

between small buffer and baseline, we see a highly significant result for the maximum task (bottom left, red), trend task (bottom middle, red), and threshold (bottom right, red). We omit reporting the test statistics between medium buffer and baseline, and between large buffer and baseline as they show similar statistics.

Design	Task	Samples	Medians (sec)	Test statistics
overlay	maximum	41	$M_1 = 19, M_2 = 19$	$T = 424, p = 9.329e^{-1}$
overlay	trend	40	$M_1 = 12, M_2 = 13$	$T = 375, p = 6.380e^{-1}$
overlay	threshold	46	$M_1 = 10, M_2 = 10$	$T = 488, p = 5.663e^{-1}$
small multiples	maximum	36	$M_1 = 17, M_2 = 27$	$T = 42, p = 4.836e^{-6}$
small multiples	trend	42	$M_1 = 10, M_2 = 15$	$T = 96, p = 8.787e^{-6}$
small multiples	threshold	45	$M_1 = 10, M_2 = 14$	$T = 196, p = 2.846e^{-4}$

Table 4.4: Completion time statistics for **no** latency comparing between baseline and **small** buffer using the Wilcoxon signed-rank test.

In general, there was no statistically significant difference between completion times for ordered and unordered encodings within each of the six between-group conditions. The only exception to this was the overlay design with medium and large buffer sizes. Users completed the maximum task significantly faster with unordered color encoding (distinct colors) than ordered color encoding (shades of blue), as illustrated in Figure 4.4. The figure shows where the two series diverged with medium and large buffer. With medium buffer, the completion time for ordered encoding vs. unordered is $C=40, T=29, Z=-2.4, p=.007$. Similarly for large buffer: $C=33, T=23, Z=-3.9, p=<1e-5$. This agrees with our initial analysis that shades of blue (ordered) causes more difficulty in perceiving correspondence compared to distinct colors (unordered). Thus it is likely that the ordered encoding would decrease efficiency when the task requires correspondence, as is the case in the maximum task.

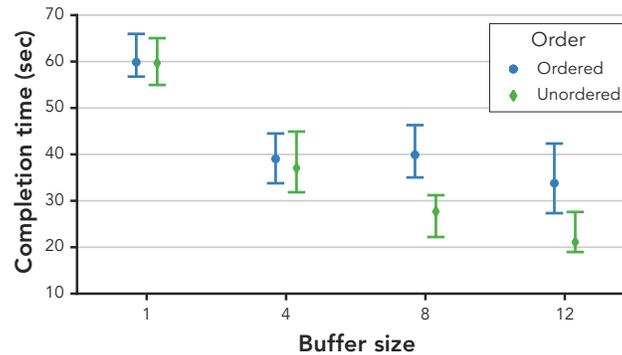


Figure 4.4: Median task completion time for the maximum task with overlay design under long latency. At larger buffer sizes, the unordered encoding shows significantly lower completion time than ordered encoding. This chart corresponds to a more detailed breakdown of the blue long latency series in the top left facet in Figure 4.3.

Concurrency

Analysis of the interaction event logs show that users performed more concurrent interactions with visualizations that leveraged Chronicled Interactions. Furthermore, task completion time exhibits a negative correlation with concurrency. Specifically, we found that (1) the higher the concurrency, the lower the task completion time, and that (2) increased buffer size also tends to increase concurrency, shown in Figure 4.5. As a result, the baseline case had low concurrency and higher completion time.

We also verify that the time savings are not due to the change of chart design, because without latency, participants were not faster with Chronicled Interactions than baseline (sometimes slightly slower, as in the case of small multiples), but were a lot faster using Chronicled Interactions than baseline when there is interactive latency.

Under long latency, across both overlay and small multiples design, with all three tasks, we found the concurrency of small buffer to be higher than baseline ($Z > 4$, $p < e-4$), and similarly for larger buffer sizes. The Pearson correlation coefficient r reveals moderate correlation for long latency for Chronicled Interactions designs. For the maximum task, $r = -0.50$; for the trend task, $r = -0.54$; and for the threshold task, $r = -0.45$. This verifies H_3 .

User preference

So far the experiment analysis has been focused on understanding the effects of Chronicled Interactions on user performance, specifically with regards to accuracy, completion time, and concurrency. Another critical component for evaluating the benefits of a new visualization interface is a usability assessment. This can help increase confidence in the designs and explain user rationale and behaviors when interacting with the visual-

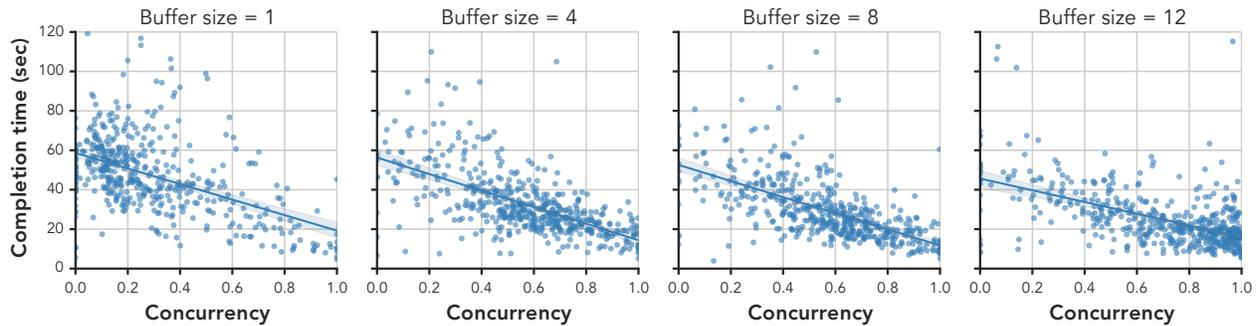


Figure 4.5: Task completion time vs. concurrency for all responses split across different buffer sizes, under long latency. The plots exhibit a similar negative correlation at all buffer sizes, suggesting increased use of concurrent interactions enables lower completion time. Designs which make use of larger buffer sizes have a higher density of respondents with high concurrency and low task completion time, corresponding to the lower right region of each graph. This is in contrast to the baseline where buffer size is 1 whose points are clustered to the upper left.

izations. For the rest of this section, we focus on analyzing the results of the usability survey that study participants completed at the end of each experiment.

Overall, usability survey results were aligned with the user performance results from earlier as expected. Users had a significant preference for Chronicled Interactions across different designs and tasks. When asked about buffer size preference on a 5-point Likert scale, participants greatly preferred a buffer size greater than the baseline of 1 ($p < e-5$, 95% CI value=4.5, where 5 is “strongly prefer” buffer size greater than one). This verifies H4.

Participants found a larger buffer size to decrease the difficulty introduced by latency ($p < e-5$, $hl=1.5$). This is consistent with some of the free response comments shared, where negative words such as “painful”, “frustrating”, “tedious”, and “awful” were often used to describe tasks with latency using the baseline buffer size. Some users said that responses were hard to remember—*“I had a hard time remembering what I’d just seen a second ago.”* In contrast, many commented on the ease of Chronicled Interactions when faced with latency—*“The ability to load several months at once definitely offsets any loading latency – difficulty was roughly the same as one month with no latency. One month with latency was a bit painful.”*

Interestingly, the perceived speed of loading seemed to have changed as well—*“Some of the tasks loaded really slow, single month got irritating waiting. Most of the multiple tasks loaded fairly quickly.”* This perceptual shift in how long interactive requests appear to load is immensely valuable for designing visualizations which must commonly make high-latency requests. This is a similar motivation for why interfaces display a progress indicator such as a loading spinner or progress bar when making the user wait for a process to complete [cite]. Giving the user feedback can reduce the perceived waiting

time without actually increasing any processing speeds. This finding provides strong support for use of Chroniced Interactions.

Participants preferred larger buffer sizes: they preferred medium to small, as opposed to being neutral ($p < e-5$, 95% CI=4.5, variance=1.4), and large to medium ($p < e-5$, 95% CI=4.9, variance=1.3).

When asked to rate preference for an ordered color encoding versus an unordered color encoding, respondents tended to prefer the unordered encoding of distinct colors. With small buffer size, they preferred unordered to ordered, ($p < e-5$, 95% CI=4.0, variance=1.9), and with large buffer size ($p < e-5$, 95% CI=4.5, variance=1.7). This preference increased with a larger buffer size, ($p=0.00016$, $hl=1.5$).

While in aggregate there was no evidence for the preference for ordered encoding, some participants voiced their favor for it such as one user who commented that they *“generally preferred the distinct colors but to figure out the corresponding month was a little tedious”*, and another *“I actually though[t] the shades of blue was much easier to read rather the[n] having to reference the chart to see what color belonged to what month, the shades were much more intuitive.”*. Furthermore, for the trend task about 20% of the participants strongly preferred ordered encodings. This indicates that a number of participants, albeit small, were able to make use of ordered color encoding to better perform the task. The trend task seems to be a good fit for ordered encodings because reading the ordered history of past interactions is helpful for determining trends. Whereas the maximum and threshold task do not require remembering the order of interactions. Enabling the user to make better use of ordered encoding is of interest for future work.

Chapter 5

Conclusion

In this thesis we presented Chronicled Interactions, a new model for designing asynchronous interactive visualizations that are resilient to the effects of interactive latency. The core idea was to explicitly manage a user's interaction history, and visually encode it within the interface so that users can interact concurrently while viewing results as they are processed. Through a series of experiments, we verified that users are able to leverage concurrency through using Chronicled Interactions for better performance and usability compared to existing visual interfaces.

Designing a new model for interactions was motivated by current challenges facing interactive visualizations as they scale to larger datasets and more complex workloads, where data is distributed and partitioned among a network. Traditional interactive visualizations assume that all the data being visualized is locally available, and thus a gap exists between these tools and the needs of new data processing platforms. To bridge this gap, our interaction model assumes that latency exists rather than assuming interactions are processed instantaneously.

While this new interaction design shows promising benefits for interactive visualizations of large datasets, there are multiple directions for future work. One aspect of big data analytics systems that we have not addressed is support for real-time streaming data from a server. Chronicled Interactions currently assumes a request-response model, but could be augmented to express the history of streaming data as well. And as alluded to in the first chapter, designing complex coordinated views with multiple linked filters and dependencies is still currently a challenge. We have demonstrated success on simple visualizations, and so extending the model to support interaction techniques like brushing and linking across multiple views is a useful follow-up.

We hope to incorporate the ideas presented here into a programming framework that allows visualization designers to leverage the benefits of Chronicled Interactions directly. Such a framework could automatically handle the storage of interaction history and reactively render updates to the visualization UI. This could help ease the burden of developing user interfaces that are resilient to latency, and improve existing visualization tools to support broader use cases.

Bibliography

- [1] Muhammad Adnan, Mike Just, and Lynne Baillie. “Investigating time series visualisations to improve the user experience”. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM. 2016, pp. 5444–5455.
- [2] Francis J Anscombe. “Graphs in statistical analysis”. In: *The American Statistician* 27.1 (1973), pp. 17–21.
- [3] Richard A Becker and William S Cleveland. “Brushing scatterplots”. In: *Technometrics* 29.2 (1987), pp. 127–142.
- [4] Jacques Bertin. “Semiology of graphics: diagrams, networks, maps”. In: (1983).
- [5] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. “D³ data-driven documents”. In: *IEEE transactions on visualization and computer graphics* 17.12 (2011), pp. 2301–2309.
- [6] Stuart K Card, Allen Newell, and Thomas P Moran. “The psychology of human-computer interaction”. In: (1983).
- [7] Chartio. *Chartio Support Variables*. 2017. URL: <https://support.chartio.com/docs/variables/> (visited on 05/06/2017).
- [8] Sanket Chintapalli et al. “Benchmarking streaming computation engines: Storm, Flink and Spark streaming”. In: *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE. 2016, pp. 1789–1792.
- [9] William S Cleveland and Robert McGill. “Graphical perception: Theory, experimentation, and application to the development of graphical methods”. In: *Journal of the American statistical association* 79.387 (1984), pp. 531–554.
- [10] Cloudera. *Charting Time-Series Data*. 2017. URL: https://www.cloudera.com/documentation/enterprise/latest/topics/cm_dg_chart_time_series_data.html (visited on 05/06/2017).
- [11] Cycle.js. *Streams*. 2017. URL: <https://cycle.js.org/streams.html> (visited on 05/06/2017).
- [12] Stef van den Elzen and Jarke J van Wijk. “Small multiples, large singles: A new approach for visual data exploration”. In: *Computer Graphics Forum*. Vol. 32. 3pt2. Wiley Online Library. 2013, pp. 191–200.

- [13] Mi Feng et al. "HindSight: Encouraging Exploration through Direct Encoding of Personal Interaction History". In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 351–360.
- [14] Stephen Few. "Solutions to the Problem of Over-plotting in Graphs". In: *Visual Business Intelligence Newsletter* (2008).
- [15] Danyel Fisher, Igor Popov, Steven Drucker, et al. "Trust me, I'm partially right: incremental visualization lets analysts explore large datasets faster". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2012, pp. 1673–1682.
- [16] Mark Harrower and Cynthia A Brewer. "ColorBrewer.org: an online tool for selecting colour schemes for maps". In: *The Cartographic Journal* 40.1 (2003), pp. 27–37.
- [17] Christopher Healey and James Enns. "Attention and visual memory in visualization and computer graphics". In: *IEEE transactions on visualization and computer graphics* 18.7 (2012), pp. 1170–1188.
- [18] Jeffrey Heer and Michael Bostock. "Crowdsourcing graphical perception: using mechanical turk to assess visualization design". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2010, pp. 203–212.
- [19] Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. "A tour through the visualization zoo." In: *Commun. Acm* 53.6 (2010), pp. 59–67.
- [20] Jeffrey Heer et al. "Graphical histories for visualization: Supporting analysis, communication, and evaluation". In: *IEEE transactions on visualization and computer graphics* 14.6 (2008).
- [21] James Hollan, Edwin Hutchins, and David Kirsh. "Distributed cognition: toward a new foundation for human-computer interaction research". In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 7.2 (2000), pp. 174–196.
- [22] Edwin L Hutchins, James D Hollan, and Donald A Norman. "Direct manipulation interfaces". In: *Human-Computer Interaction* 1.4 (1985), pp. 311–338.
- [23] Aniket Kittur, Ed H Chi, and Bongwon Suh. "Crowdsourcing user studies with Mechanical Turk". In: *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM. 2008, pp. 453–456.
- [24] Zhicheng Liu and Jeffrey Heer. "The effects of interactive latency on exploratory visual analysis". In: *IEEE transactions on visualization and computer graphics* 20.12 (2014), pp. 2122–2131.
- [25] Jock Mackinlay. "Automating the design of graphical presentations of relational information". In: *Acm Transactions On Graphics (Tog)* 5.2 (1986), pp. 110–141.
- [26] MapD. *NYC Taxi Rides*. 2017. URL: <https://www.mapd.com/demos/taxis/> (visited on 05/06/2017).

- [27] Winter Mason and Siddharth Suri. “Conducting behavioral research on Amazon’s Mechanical Turk”. In: *Behavior research methods* 44.1 (2012), pp. 1–23.
- [28] Erik Meijer. “Reactive extensions (Rx): curing your asynchronous programming blues”. In: *ACM SIGPLAN Commercial Users of Functional Programming*. ACM. 2010, p. 11.
- [29] Henrik Nilsson, Antony Courtney, and John Peterson. “Functional reactive programming, continued”. In: *Proceedings of the 2002 ACM SIGPLAN workshop on Haskell*. ACM. 2002, pp. 51–64.
- [30] Donald A Norman. “Cognitive engineering”. In: *User centered system design: New perspectives on human-computer interaction* 3161 (1986).
- [31] Lucy Nowell, Elizabeth Hetzler, and Ted Tanasse. “Change blindness in information visualization: A case study”. In: *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS’01)*. IEEE Computer Society. 2001, p. 15.
- [32] QuantQuote. *Historical Stock Data*. 2017. URL: <https://quantquote.com/historical-stock-data> (visited on 01/15/2017).
- [33] New Relic. *Filter your Insights data*. 2017. URL: <https://docs.newrelic.com/docs/insights/using-insights-ui/filter-segment/filter-your-insights-data> (visited on 05/06/2017).
- [34] George Robertson et al. “Effectiveness of animation in trend visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008).
- [35] Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. “Declarative interaction design for data visualization”. In: *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM. 2014, pp. 669–678.
- [36] C Sievert et al. “plotly: Create Interactive Web Graphics via ‘plotly.js’”. In: *R package version 4.6* (2016).
- [37] Justin Talbot, Vidya Setlur, and Anushka Anand. “Four experiments on the perception of bar charts”. In: *IEEE transactions on visualization and computer graphics* 20.12 (2014), pp. 2152–2160.
- [38] Diane Tang, Chris Stolte, and Robert Bosch. “Design choices when architecting visualizations”. In: *Information Visualization* 3.2 (2004), pp. 65–79.
- [39] Edward R Tufte. *Visual explanations: images and quantities, evidence and narrative*. Vol. 36. Graphics Press Cheshire, CT, 1997.
- [40] Barbara Tversky, Julie Bauer Morrison, and Mireille Betrancourt. “Animation: can it facilitate?” In: *International journal of human-computer studies* 57.4 (2002), pp. 247–262.
- [41] Jarke J Van Wijk. “The value of visualization”. In: *Visualization, 2005. VIS 05. IEEE*. IEEE. 2005, pp. 79–86.

- [42] Richard Wesley, Matthew Eldridge, and Pawel T Terlecki. “An analytic data engine for visualization in tableau”. In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM. 2011, pp. 1185–1194.
- [43] Hadley Wickham. *ggplot2: elegant graphics for data analysis*. Springer, 2016.
- [44] Kanit Wongsuphasawat et al. “Voyager: Exploratory analysis via faceted browsing of visualization recommendations”. In: *IEEE transactions on visualization and computer graphics* 22.1 (2016), pp. 649–658.
- [45] Matei Zaharia et al. “Spark: Cluster Computing with Working Sets.” In: *HotCloud* 10.10-10 (2010), p. 95.