Making Sense of Asynchrony in Interactive Data Visualizations

Yifan Wu, Larry Xu, Remco Chang, Joseph M. Hellerstein, Eugene Wu

Abstract—Asynchronous interfaces allow users to concurrently issue requests while existing ones are processed. While it is widely used to support non-blocking input when there is latency, it's not clear if people can make use of asynchrony as the data is updating, since the UI updates dynamically and the changes can be hard to interpret. Interactive data visualization presents an interesting context for studying the effects of asynchronous interfaces, since interactions are frequent, task latencies can vary widely, and results often require interpretation. In this paper, we study the effects of introducing asynchrony into interactive visualizations, under different latencies, and with different tasks. We observe that traditional asynchronous interfaces, where results update in place, induce users to wait for the result before interacting, not taking advantage of the asynchronous rendering of the results. However, when results are rendered cumulatively over the recent history, users perform asynchronous interactions and get faster task completion times.

Index Terms—interactive data visualization, interaction, latency

1 Introduction

Traditional interactive data visualization systems assume that data can be processed quickly to support sub-second "interactive latency". This approach simplifies the design of the visualization UI, and ensures fluid direct manipulation interfaces that facilitate user data exploration [1]. However, as interactive data visualizations are increasingly an integral part of big data analysis, the scale of the datasets and the necessary computational power has made it necessary to shift the data processing and storage to remote data management systems (e.g., a database). In such a client-server architecture, client interactions are translated into server requests that incur network and data processing delays. In this networked world, communication and data processing latency is a reality and must be addressed in the application design.

To reduce the latency introduced with this approach, traditional systems preload all the data into memory and process subsequent user interactions synchronously. Recent massive scale interactive data visualization systems (such as imMens [1], MapD [2] and Graphistry [3]) and progressive visualization systems [4, 5, 6, 7, 8] address the issue of latency by fully embracing the client-server architecture, with each user interaction triggering a new request to the server. In addition to building faster backend systems to reduce processing time, they all leverage asynchrony in the visualization interface—users can manipulate the interface without waiting by sending requests asynchronously. Each request is concurrently sent and evaluated by the server; the responses can then be rendered when received on the client.

The benefits of including asynchronous interactions in a visualization system are well established—asynchrony allows for the parallelization of interaction request handling, which reduces the overall system latency. However, from the user’s perspective, it is less clear whether introducing asynchrony can improve the user’s ability to correctly and efficiently use and understand the visualization system. While there have been studies of the usability of specific asynchronous visualization systems (e.g., [1, 5, 6, 9]), to the best of our knowledge, there has been limited research on how users react to, and make sense of, asynchronous interactions in data visualizations in general.

This paper studies the question “can users effectively interact with asynchronous visualizations?” To help answer this question, we first conducted two pilot studies to better understand factors that determine the usability of asynchronous rendering. The first pilot tests the effect of naively applying asynchronous rendering to a traditional (synchronous) interactive visualization, without changing the design. We find that this approach led to frustrating user experiences. In fact, participants adapted their behavior to the interface in a way that negates the asynchrony—participants waited until the interface responds to their previous interaction before triggering the next one.

Our second pilot study seeks to understand if there exist interaction design techniques for which asynchronous interactive visualization may be effective. Our key design challenge is that user actions and system responses are disconnected in time. Visualizations are intended to help people reason about information by providing a stable frame of reference that can temporarily store information for visual processing [10]. Asynchrony disrupts that shared frame of reference, since the user’s latest action and the system’s latest visualization will often not match.

We borrow inspiration from asynchronous webpage loading design on news or social networking sites, where images are asynchronously loaded within placeholders. We adapt this to data visualization, where visualization request is loaded in a placeholder within an additional small multiples chart (Figure 5). We found that users were able to leverage asynchronous interactions to complete tasks faster, and reported higher user satisfaction.

The pilots showed two points within a design space for asynchronous interactions—on the one extreme, no history is kept, and on the other, all history is kept. Between these two extremes lies the recent past. We hypothesize that visualizing the recent past can stabilize the visualization and create sufficient context for users to interpret visual updates.
We further refine the design inspiration into a framework for transforming interactions based on design guidelines for latency, and dealing with asynchrony and conflict from collaborative groupware research. The simple framework identifies a visual representation of the correspondence between past user interactions and the out-of-order visual responses.

We then analyzed the effect of asynchronous rendering on user interaction behavior using the top-down mental model introduced by Liu and Stasko [11]. We found that three factors—the visualization design of the asynchronous results, the user task, and the latency profiles of the requests—impact the usability of an asynchronous interactive visualization interface. Our final experiment evaluates these three factors.

We find that, although naively applying asynchronous rendering to traditional data visualizations can reduce the usability of interactive visualizations, careful interaction design can improve usability for several common visual analysis tasks. These results point towards a rich design space that explicitly takes latency and asynchrony into account. This holds the potential to unlock previously inaccessible data and queries, requiring only changes to the frontend that are neither too intrusive nor difficult to implement and deploy.

In summary, this paper makes the following contributions:

- We examine the question of “can users interact with asynchronous visualizations?” Our results highlight the potential of asynchronous interactions, and a rich design space where request latency must be taken into account when designing interactive visualizations.
- We identify factors that make asynchrony difficult—unstable representation, and a solution framework—stabilizing designs using interaction history as a buffer, while creating visual representation of the correspondence between past user interactions and the out-of-order visual responses.
- We use the top-down mental model proposed by Liu and Stasko [11] to identify three factors that affect the usability of asynchronous visualizations: visualization designs, tasks, and latency profiles.
- We conduct a controlled experiment to evaluate the impact of these three factors. Based on the results of the experiment, we propose practical design guidelines that can aid visualization researchers and practitioners towards designing better asynchronous visualization systems.

2 Related Work
This section discusses prior research that has informed our thinking: latency’s effect on cognition—more specifically, visual analytics and computer-supported cooperative work (CSCW), and design guidelines for high latency interfaces. We also discuss related solutions to the latency problems, including online aggregation/incremental updates/progressive visual analytics, and fast visualization systems.

2.1 Effects of Latency on Usability
How bad is latency on the user interface? Card et al. investigated the cognitive units involved in human-computer interaction and provided different latency models for different tasks [12]. The cognitive science literature suggests that latencies are difficult because they require users to make use of short-term memory to perform visual analysis [13]. Short-term memory is limited [14], it decays over time [15], and is expensive to use: in an experiment conducted by Ballard et al., subjects serialized their tasks to avoid using short-term memory [16]. Even without latency, visualization results can be easily forgotten [17], [18]—with latency, the risk of forgetting is much higher.

Well-known HCI principles also shed light on the challenge of latency. Hutchins et al. note that “The gulf of evaluation is bridged by making the output displays present a good conceptual model of the system that is readily perceived, interpreted, and evaluated.” [19] The gulf of evaluation is large with latency, and even larger with asynchrony, since the output at any moment is not connected to the user’s most recent input.

Liu and Heer observe that an additional delay of 500ms incurs significant costs: it decreases user activity and dataset coverage, and reduces the rate at which users make observations, draw generalizations and generate hypotheses [11]. They found that different interactions are affected by delay differently (e.g., zooming less than brush-linking). They also noticed a shift in user strategies.

While the results by Liu and Heer are illuminating and hints at a large interplay between latency and visual analysis, only a short latency of 500 milliseconds for a blocking interface has been investigated, which calls for more research on other non-blocking interfaces and higher latencies.

2.2 Designs for Latency
What can we do given the bad effects of latency? Seow provides a systematic discourse on engineering time [20]. One of the key ideas of the book is that delay is subjectively perceived, and responsiveness is relative to the interaction. This motivates our exploration in decoupling the interaction and the response. The book also discusses, in a similar spirit to progress bars, techniques to enhance user satisfaction, such as “underpromise, overdeliver” (in the spirit of past psychology studies [21], [22]). Johnson also discussed common “responsiveness bloopers” [23], identifying techniques to make an application responsive despite latency, such as showing meaningful progress bars, computing user request in a non-serial order (or even anticipating future requests), and acknowledging user input.

One major theme to improve user experience in the face of latency is to communicate the latency and the state of the UI. The most familiar research that comes to mind is percent-done progress indicators, a technique for graphically showing how much of a long task has been completed [24]. Myers have identified progress indicator to be helpful for users to multi-process, understand that the system has acknowledged the request and is less bored with the interface since the progress is animating [24]. Later research has explored variations of progress bar designs to improve the user experience [25].

Besides progress bars, there are other designs championed by researchers that enhances the user experience in the face of latency. Ebling et al. mentioned that even expert users get confused about caching behavior, which changes latency experienced and designed a system to effectively communicate the state of the cache to the user [25].

These designs, combined with basic design principles and frameworks [11], [19], [27], help us understand the results from the pilot study and informed our proposed design.

2.3 Designs for Asynchrony in Groupware
Latency often causes responses to be out of order, leading to correctness and comprehensions issues, beyond just slowness. The
CSCW community has developed different mechanisms and models for effectively dealing with asynchrony across users. Greenberg et al. described the insight that collaborative software’s asynchronous updates and the shared state could be modeled as a distributed system, where asynchrony could be understood as a form of “concurrency control” [28]. The authors helped translate some of the distributed computing mechanisms into user design considerations. Their work has inspired us to model interaction consistency in interactive visualizations through database consistency semantics [29]. Edwards et al. designed Bayou for collaborative software programmers to provide application-specific ways to perform detection and resolution of conflict that arises between [30].

In addition to theoretical models, the CSCW community has also over time proposed concrete designs to help guide the users in the face of asynchrony in groupware. Dourish et al. suggested that making users more aware of the current state of the system helps users navigate asynchrony and prevent conflicts [31]. Gutwin proposed “trace”, a visualization of the immediate past, to maintain context for asynchronous updates [32]. The idea is very close to our formulation of history as a stable anchor for changing updates. Gutwin et al. summarized change awareness in asynchronous systems in their paper on DISCO, a framework to deal with disconnection in synchronous groupware, which is relevant for our designs to make sense of asynchrony [33].

Savery et al. described a programming model for time to deal with asynchrony, where shared state variables are not represented as a single value, but as a series of values in time [34]. As we will illustrate, treating an interaction not as a single value at any point in time, but a series of values in time, with the corresponding relationship, is critical to gaining an understanding of asynchrony and coming up with new designs. Savery et al. further illustrated and summarized a gallery of designs enabled by the programming model. In particular, the idea of smoothly animating the change to preserve the context of interaction is much aligned with our pilot findings in search of designs to support asynchronous interactions [35].

Ideas from the CSCW community informed our thinking about asynchrony. While our use case does not have multiple users, asynchronous results still form a “conflict”, and the fact that the same UI is being edited can be seen as “shared state”. It is no surprise that there are common factors between our finding for interactive visualizations and those for collaborative groupware, as asynchrony is fundamentally about interactions in time, and an improved user experience involves some “context” using history.

2.4 Systems to Enhance Interactivity

Much work has been done on making data processing systems more efficient. Construction of indexes, compressed columns and multidimensional (data cube) summary statistics can effectively cut down processing time for certain visualization tasks [35], [36], [37], [38], [39]. Our work is complementary to the performance enhancement, as users tend to push the envelope of computation—with more processing power comes larger data sets and more complex computations.

Making a usable interactive visualization system does not stop at just enhancing system performance. Weaver and Livny designed a system that prioritizes computation of the UI, and data computation as secondary so that the interaction context is maintained over the content [40]. Chan et al. developed an interactive visualization system for time series analysis, ATLAS, where interactivity is guaranteed, but under assumptions of an expandable network of computing resources (e.g., new machines), or limits to the interaction speed (e.g., panning speed) [41]. Piringer et al. developed a multi-threading architecture that could cancel requests based on new user interactions and guarantees responsiveness and non-blocking interactions [42], but all but the most recent request is essentially canceled. This paper offers an idea that makes use of asynchronous, instead of managing asynchrony by canceling asynchrony results.

Similarly, many new frontend programming frameworks provide declarative mechanism to deal with asynchrony. One prominent example is Elm [43]. The default behavior the framework supports is to cancel concurrent requests from the same “signal” (or stream). Other mechanisms are of course possible since the language is Turing complete, but as is in the case of the previous research [42], canceling concurrent requests issued in the past is the natural default.

Compared to these work that deals with asynchrony and implementing only a subset of asynchronous rendering behaviors, our work puts asynchrony front and center and utilizes asynchrony. We show that there are more kinds of asynchronous interactions than just maintaining a single current state of interaction.

2.5 Progressive Visualization Updates

An especially relevant class of interaction design for visualizations is progressively updating visualizations. The idea has been developed in the database and visualization community over the past couple decades.

In the database community, Hellerstein et al. initially proposed the concept of "online aggregation" in 1997 to allow users to observe partial results, progress, and confidence intervals [4]. In the visualization community, Hetzler et al. proposed a visual analysis system to support constantly evolving text collections [44], including widgets to control the update and visually indicating the change of old data points and new ones.

More recently, Fisher et al. developed an incrementally updating interactive visualization system following the concept of online aggregation [5]. Stoplper et al. extended the idea further to "progressive visual analytics" (PVA) [6] where both the rendering and the data analysis are performed in a progressive manner. Zgraggen et al. further explored the effect of PVA on end users [8] and found that the PVA approach shows great potential, and the asynchronous interactivity improves performance, but there are also challenges posed by the new interpretation of error terms and the unnaturalness of changes to some participants. Moritz et al. [9] proposed optimistic visualization, where the users could eventually see the results in full accuracy, and verified that their Pangloss system were effective at avoiding latencies as high as minutes, and avoiding the downside of approximate results.

We see our work as complementary to the ongoing research in the PVA community. Recent advances in PVA have shown that multiplexing a single interaction with a stream of progressive updates can help alleviate the negative impact of latency. Our work seeks to augment this line of thinking by investigating a different mechanism for creating opportunities to see more results given the same processing complexity. Instead of incrementally improving the responses of a single long-running request with a progressive backend, we look at multiple independent and concurrent requests with traditional backends.
3 Anatomy of an Asynchronous Interactive Visualization

When the underlying dataset is small, the visualization system can quickly respond to user interactions and support fluid user interactions. However, as datasets continue to grow and shift to cloud-based data management, client requests will invariably incur non-trivial data processing and network communication delays that affect the usability of the visualization.

The terminology around asynchrony for interactive visualizations is often referred to as “blocking” or “non-blocking”. For this paper, we need a more precise definition, as there are different behaviors within the two categories. This section categorizes the different effects that latency and asynchrony can have on the user-interface, and outlines the challenges introduced by the use of asynchrony.

3.1 Anatomy of Asynchronous Rendering

Fig. 1. (a) Line chart with facet filter on the left. (b) When a user hovers over a facet button, a request is sent to the server, and the visualization renders a spinner until the response is received and rendered.

Our discussion is grounded in the visualization in Figure 1. When the user hovers over a button (e.g., ‘A’) in the left facet, it triggers a request to fetch the corresponding data that is used to update the plot on the right (Figure 1(a)). When request latency is introduced, the user will need to wait for the response, and visualizations typically render a spinner while the request is processed (Figure 1(b)).

Even in this simple example, some design considerations arise. For instance, while the user waits for the response to the ‘B’ interaction, is she blocked from any interactions until the response is rendered? Or is she allowed to interact with the rest of the visualization? If so, then what can she expect to see as she hovers over other buttons? To understand these questions, we assume that the user wants to hover over the buttons A, B, and C in order, where their responses are respectively A’, B’, and C’. We categorize the ways that the interface may respond under different interaction and asynchrony modalities (Figure 2).

The top diagram in Figure 2 depicts a time-ordered model, where time increases from left to right. User inputs are depicted along the top line (interaction history), and the responses are rendered along the bottom line (render history). A dashed arrow between the interaction and render history corresponds to the time to respond to the request—a more horizontal arrow means a longer request latency.

Figure 2(a) shows the ideal case: requests respond instantly (vertical arrows) so that users can interact with the interface without waiting for slow requests. In contrast, Figure 2(b) shows the case where the user is not allowed to submit a new request until the prior one is rendered (their input is blocked). For instance, B is not triggered until A’ is rendered.

To avoid blocking the user input, many visualizations use asynchrony for user input and block the rendering (Figure 2(b)). Users freely interact with the visualization. However, new requests supersede and cancel previous requests (the × over the first two arrows). Thus, only the most recent request will be fully processed and rendered.

A further design choice is to not block the input nor rendering (Figure 2(c)). Consider the case where each request latency is identical. The user triggers requests A, B, and C, and after a fixed delay, their responses are rendered in order. Although this design choice ultimately presents all responses to the user, it can potentially introduce a request-response mismatch. For instance, A’ is rendered immediately after the user hovers over C, which can cause the user to incorrectly infer a correspondence between the two.

Although this may already cause some confusion, it is even more challenging if the latency varies for different requests. Figure 2(d) depicts such a case with arrows that have different widths. Note that the arrows for A and B crossover, and the response B’ is rendered before A’ even though the user input ordered A before B. This is an out-of-order interaction. Further, note that the responses for A and C occur at nearly the same time and cause flashing updates, where A’ is flashed on the screen and immediately replaced with C’.

Fig. 2. 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. Case (1) is the ideal no-latency scenario commonly assumed by visualization designers—everything works as expected. (2) With latency, the user waits for each response to load before interacting. (3) With latency, the user interacts without waiting, and in-flight responses are not rendered. (4) With latency, the user interacts without waiting, and all responses are rendered. (5) With latency, the user interacts without waiting and may see responses in a different order than requests were issued.
3.2 Potential Difficulties Using Asynchronous Rendering

The above examples illustrate the complexities when applying asynchronous rendering to even a simple interactive data visualization because it can cause unexpected effects that run contrary to well-established design principles such as direct manipulation. As shown by Hutchins, Hollan, and Norman [19], there are cognitive benefits to a direct manipulation interface because it reduces the semantic and articulatory distances between the user’s actions and goals. In effect, direct manipulation assumes that there is an immediate, one-to-one correspondence between user actions and their impact on the interface so that the user has a sense that their actions directly affect the interface environment.

In contrast, asynchronous rendering can break this illusion. When a user manipulates multiple interaction elements, the system might not respond immediately, nor in the sequence that the user’s actions are performed. This is confusing because either the UI displays an update corresponding to a different interaction than the one just issued (Figure 2.4), or it displays a response from a request that is even older than that of the last response displayed (Figure 2.5). A careful user may catch the discrepancies and re-do their interaction, causing a poor user experience, but a less careful user may read the wrong values, leading to erroneous analysis results. In either case, it can increase the cognitive load for the user.

These examples highlight the need for careful visualization design when using asynchrony in high latency settings.

4 Pilot: Can People Use Asynchronous Vis?

Asynchronous rendering of visualization allows for parallel computation and data fetching, thus reducing the total latency and improving task completion time. However, it is unclear whether a user can correctly and efficiently utilize such an asynchronous rendering system. Scenarios 4 and 5 in Figure 2 illustrate that asynchronous rendering systems can introduce complex ordering relationships between the user’s interactions and the system’s responses due to latency in the network or the system. So this begs the question, “can users successfully use asynchronously rendered visualizations?”

To answer this question, we conducted two pilot studies to understand how asynchronous rendering is used today. Our first pilot study seeks to replicate common visualizations similar to the one shown in Figure 1 but with asynchronous rendering. Although simplistic, this hover-response visualization is at the heart of a range of popular, but more complex designs such as cross-filtering, brushing-and-linking, and more broadly coordinated multiple visualizations (CMVs) [45]. We chose simple visual analytic tasks, as opposed to open-ended exploratory studies used in previously mentioned studies [1, 8, 9], to control for potential confounding factors. It is important to get a basic understanding of the effects before proceeding to more complex scenarios.

Our second pilot seeks to replicate people’s ability to use web pages asynchronously but in the context of visualization. Our premise is that if users can successfully use asynchronously updating web pages, presumably they can do the same with a visualization if the visualization is designed similarly. Through these two pilot studies, we aim to identify the challenges of using asynchronous rendering and the design factors that make these visualizations difficult, or easy, to use.

4.1 Pilot 1: Replicating Naive Asynchronous Rendering

We use a faceted bar chart visualization (see Figure 3) in this pilot study similar to the one described earlier (Figure 1). Although this visualization is simple, the interaction used in this example is the same as the one used in more complex techniques such as cross-filtering and brushing-and-linking. In all these visualizations, a user would interact with a user interface component (which can be UI elements like buttons or a separate visualization), and observe the response in a separate visualization.

Task: Since asynchronous rendering causes reordering of the results (Figure 2.5), we chose a visual task that is not order-dependent, to encourage asynchronous interactions. For a bar chart that displays sales data for a company across months and years, we asked users to identify if any of the months crossed the sales threshold of 80 units sold.

To answer this question, we conducted two pilot studies to understand how asynchronous rendering is used today. Our first pilot study seeks to replicate common visualizations similar to the one shown in Figure 1 but with asynchronous rendering. Although simplistic, this hover-response visualization is at the heart of a range of popular, but more complex designs such as cross-filtering, brushing-and-linking, and more broadly coordinated multiple visualizations (CMVs) [45]. We chose simple visual analytic tasks, as opposed to open-ended exploratory studies used in previously mentioned studies [1, 8, 9], to control for potential confounding factors. It is important to get a basic understanding of the effects before proceeding to more complex scenarios.

Our second pilot seeks to replicate people’s ability to use web pages asynchronously but in the context of visualization. Our premise is that if users can successfully use asynchronously updating web pages, presumably they can do the same with a visualization if the visualization is designed similarly. Through these two pilot studies, we aim to identify the challenges of using asynchronous rendering and the design factors that make these visualizations difficult, or easy, to use.

4 Pilot: Can People Use Asynchronous Vis?

Asynchronous rendering of visualization allows for parallel computation and data fetching, thus reducing the total latency and improving task completion time. However, it is unclear whether a user can correctly and efficiently utilize such an asynchronous rendering system. Scenarios 4 and 5 in Figure 2 illustrate that asynchronous rendering systems can introduce complex ordering relationships between the user’s interactions and the system’s responses due to latency in the network or the system. So this begs the question, “can users successfully use asynchronously rendered visualizations?”

To answer this question, we conducted two pilot studies to understand how asynchronous rendering is used today. Our first pilot study seeks to replicate common visualizations similar to the one shown in Figure 1 but with asynchronous rendering. Although simplistic, this hover-response visualization is at the heart of a range of popular, but more complex designs such as cross-filtering, brushing-and-linking, and more broadly coordinated multiple visualizations (CMVs) [45]. We chose simple visual analytic tasks, as opposed to open-ended exploratory studies used in previously mentioned studies [1, 8, 9], to control for potential confounding factors. It is important to get a basic understanding of the effects before proceeding to more complex scenarios.

Our second pilot seeks to replicate people’s ability to use web pages asynchronously but in the context of visualization. Our premise is that if users can successfully use asynchronously updating web pages, presumably they can do the same with a visualization if the visualization is designed similarly. Through these two pilot studies, we aim to identify the challenges of using asynchronous rendering and the design factors that make these visualizations difficult, or easy, to use.

4.1 Pilot 1: Replicating Naive Asynchronous Rendering

We use a faceted bar chart visualization (see Figure 3) in this pilot study similar to the one described earlier (Figure 1). Although this visualization is simple, the interaction used in this example is the same as the one used in more complex techniques such as cross-filtering and brushing-and-linking. In all these visualizations, a user would interact with a user interface component (which can be UI elements like buttons or a separate visualization), and observe the response in a separate visualization.

Task: Since asynchronous rendering causes reordering of the results (Figure 2.5), we chose a visual task that is not order-dependent, to encourage asynchronous interactions. For a bar chart that displays sales data for a company across months and years, we asked users to identify if any of the months crossed the sales threshold of 80 units sold.

To answer this question, we conducted two pilot studies to understand how asynchronous rendering is used today. Our first pilot study seeks to replicate common visualizations similar to the one shown in Figure 1 but with asynchronous rendering. Although simplistic, this hover-response visualization is at the heart of a range of popular, but more complex designs such as cross-filtering, brushing-and-linking, and more broadly coordinated multiple visualizations (CMVs) [45]. We chose simple visual analytic tasks, as opposed to open-ended exploratory studies used in previously mentioned studies [1, 8, 9], to control for potential confounding factors. It is important to get a basic understanding of the effects before proceeding to more complex scenarios.

Our second pilot seeks to replicate people’s ability to use web pages asynchronously but in the context of visualization. Our premise is that if users can successfully use asynchronously updating web pages, presumably they can do the same with a visualization if the visualization is designed similarly. Through these two pilot studies, we aim to identify the challenges of using asynchronous rendering and the design factors that make these visualizations difficult, or easy, to use.

4.1 Pilot 1: Replicating Naive Asynchronous Rendering

We use a faceted bar chart visualization (see Figure 3) in this pilot study similar to the one described earlier (Figure 1). Although this visualization is simple, the interaction used in this example is the same as the one used in more complex techniques such as cross-filtering and brushing-and-linking. In all these visualizations, a user would interact with a user interface component (which can be UI elements like buttons or a separate visualization), and observe the response in a separate visualization.

Task: Since asynchronous rendering causes reordering of the results (Figure 2.5), we chose a visual task that is not order-dependent, to encourage asynchronous interactions. For a bar chart that displays sales data for a company across months and years, we asked users to identify if any of the months crossed the sales threshold of 80 units sold.

To answer this question, we conducted two pilot studies to understand how asynchronous rendering is used today. Our first pilot study seeks to replicate common visualizations similar to the one shown in Figure 1 but with asynchronous rendering. Although simplistic, this hover-response visualization is at the heart of a range of popular, but more complex designs such as cross-filtering, brushing-and-linking, and more broadly coordinated multiple visualizations (CMVs) [45]. We chose simple visual analytic tasks, as opposed to open-ended exploratory studies used in previously mentioned studies [1, 8, 9], to control for potential confounding factors. It is important to get a basic understanding of the effects before proceeding to more complex scenarios.

Our second pilot seeks to replicate people’s ability to use web pages asynchronously but in the context of visualization. Our premise is that if users can successfully use asynchronously updating web pages, presumably they can do the same with a visualization if the visualization is designed similarly. Through these two pilot studies, we aim to identify the challenges of using asynchronous rendering and the design factors that make these visualizations difficult, or easy, to use.

4.1 Pilot 1: Replicating Naive Asynchronous Rendering

We use a faceted bar chart visualization (see Figure 3) in this pilot study similar to the one described earlier (Figure 1). Although this visualization is simple, the interaction used in this example is the same as the one used in more complex techniques such as cross-filtering and brushing-and-linking. In all these visualizations, a user would interact with a user interface component (which can be UI elements like buttons or a separate visualization), and observe the response in a separate visualization.

Task: Since asynchronous rendering causes reordering of the results (Figure 2.5), we chose a visual task that is not order-dependent, to encourage asynchronous interactions. For a bar chart that displays sales data for a company across months and years, we asked users to identify if any of the months crossed the sales threshold of 80 units sold.

To answer this question, we conducted two pilot studies to understand how asynchronous rendering is used today. Our first pilot study seeks to replicate common visualizations similar to the one shown in Figure 1 but with asynchronous rendering. Although simplistic, this hover-response visualization is at the heart of a range of popular, but more complex designs such as cross-filtering, brushing-and-linking, and more broadly coordinated multiple visualizations (CMVs) [45]. We chose simple visual analytic tasks, as opposed to open-ended exploratory studies used in previously mentioned studies [1, 8, 9], to control for potential confounding factors. It is important to get a basic understanding of the effects before proceeding to more complex scenarios.

Our second pilot seeks to replicate people’s ability to use web pages asynchronously but in the context of visualization. Our premise is that if users can successfully use asynchronously updating web pages, presumably they can do the same with a visualization if the visualization is designed similarly. Through these two pilot studies, we aim to identify the challenges of using asynchronous rendering and the design factors that make these visualizations difficult, or easy, to use.
4.2 Pilot 1 Results and Discussions

There were no statistically significant difference between the two conditions in terms of either accuracy nor completion time; we report the unsigned Wilcoxon Rank-Sum test: baseline median=37 sec (N = 31), treatment median=33 sec (N=52), Z = 0.63, p<0.5, where N denotes the count of the group.

This suggests that: (1) the participants were not able to take advantage of the asynchronous rendering in the treatment condition to complete the task faster. However, (2) the participants were not confused by the unfamiliar UX, and they were able to complete the task accurately.

Through the comments, we find that although the participants were able to complete the tasks accurately, they were frustrated by the interface—half of our participants reported their experience using words like “irritating” and “angry”. Despite the frustration, they seemed unaware or unwilling to make use of asynchronous rendering.

The most interesting aspect of the results came from analyzing the participants’ interaction logs. It turns out that the reason that the participants had the same completion accuracy and time using the treatment condition as they did in the baseline condition is that they only make a new interaction after seeing the results of the previous. Referred to as “self-serialization”, what we realized is that the participants were in fact confused by the asynchronous behaviors of the system. To improve the confusion, the participants blocked their own interactions to make the interaction-response relationship serial and synchronous. In other words, when confronted with asynchronous rendering systems, the participants introduced delays and turned the situation in Figures 2.5 to Figures 2.2. As a result, the participants’ task completion time and accuracy are indistinguishable between the treatment and the baseline conditions.

4.3 Pilot 2: Replicating Asynchronous Webpage Loading

While the previous pilot demonstrates that a naive implementation of an asynchronously rendered visualization can lead to a system that is difficult to use, studies in other domains suggest that design may play a role. Guse et al. studied the user’s ability to select specific images from an asynchronously rendered webpage (Figure 4). With no delay, the average task completion time is 7.6 seconds; with loading delay of 8 seconds, the task completion time increased to 10.8 seconds. This is 3 seconds longer than the no-latency case, and lower than the full 8s delay.

Inspired by this observation, our second pilot emulates the effect of asynchronous page loading for interactive visualizations. Our design of the new visualization replicates common AJAX-based page loading (e.g., see Figure 4) when a new interaction response is received, instead of updating in place, the results are appended to the screen and do not replace existing ones. Figure 5 shows the design of this kind visualization, which we describe this as “Cumulative Asynchronous Rendering” (hereafter referred to as Cumulative).

![Fig. 4. The asynchronous web page interface from Guse et al. loads images asynchronously; pending requests are rendered with a throbber.](image)

![Fig. 5. Interface for pilot 2. User interaction requests are asynchronously loaded and rendered. Light blue boxes are annotations depicting the locations where the new interaction results will be shown. The light blue boxes are not shown to users.](image)

Similar to the previous pilot, this study was conducted on Amazon Mechanical Turk. The data, tasks, procedures, recruiting, and measures are all kept the same as the previous pilot for consistency. Since previous studies have shown people can use AJAX-based webpages and benefit from asynchrony with faster task completion (and without loss of accuracy), we hypothesized that this asynchronously rendered visualization design should also allow users to be able to take advantage of asynchrony and complete tasks faster.

4.4 Pilot 2 Results

Consistent with our hypothesis, we find that participants completed the tasks faster: baseline median=37 sec (N=31), treatment median=17 sec (N=54), Z=3.22, p<0.002, (the baseline is shared with Pilot 1).

Figure 6 shows the completion times from the two pilot studies. Blocking and Naive refer to the baseline and treatment conditions in the first pilot study, respectively. Cumulative refers to the webpage-inspired design used in the second pilot study. As shown in the results, Cumulative is significantly faster than the other two, by a factor of 2, whereas there is no statistically significant difference between Blocking and Naive.

![Fig. 6. Completion time of baseline and two conditions. Cumulative helps users complete tasks much faster.](image)

To better understand why participants completed tasks faster under asynchrony, we measured the degree of concurrency that the users exhibited, meaning the percentage of task completion time where there was more than one concurrent request. Low concurrency (=0) means that, for the entire task, users did not trigger more than one request at a time, while high concurrency (=1) means that the user always triggered concurrent requests. Although this does not measure the number of concurrent requests at any given time, it provides a sense of whether asynchronous interactions were used consistently, or spuriously.
Figure 7 shows that the Cumulative condition induced significantly more concurrency (median=0.86, mean=0.59) than the baseline and naive conditions. Figure 8 shows that increased concurrency is correlated with a decrease in task completion time. Encouragingly, the Cumulative condition did not receive negative feedback, and comments (if any) were of the form “it loaded just fine”.

![Fig. 7. Amount of concurrency. The Cumulative condition significantly increases user concurrency.](image)

![Fig. 8. Completion time correlated with level of concurrency, we can see a clear negative correlation.](image)

5 Why is Asynchronous Rendering Difficult?

The results of the pilot studies paint a complex picture of how users interact with asynchronously rendered visualizations. The conflicting outcomes suggest that our initial question of “can users successfully use asynchronously rendered visualizations?” cannot be quickly answered with a simple yes or no. Instead, the user’s ability seems to depend on a range of factors, including but not limited to the visualization design, the choice of the rendering algorithm, the types of latency, and others.

To reason about the outcomes of the pilot studies and better understand the relationship between these factors, we first examine why asynchronous rendering can be challenging to use. Using the “top-down model” of interactive visualization proposed by Liu and Stasko [11], we observe that asynchronous rendering affects the user in three ways: (1) it weakens the “external anchoring” of the user’s reasoning process, (2) it interrupts the user’s “information foraging” process, and finally, (3) it disrupts the user’s “cognitive offloading” ability when using a visualization. Based on these three observations, we derive three corresponding design factors that we further evaluate in a formal experiment.

5.1 Mental Model, Interaction, and Visualization

Figure 9 shows the cycles of actions in using visualization for reasoning as proposed by Liu and Stasko [11]. External Anchoring is the process of a user projecting their reasoning process onto an external representation. Similar to the theory of distributed cognition [47], it is believed that a stable representation as the external anchor is necessary for a user to perform reasoning successfully. In the case of using an asynchronously rendered visualization, the visualization can be shifting and changing seemingly without reason. Without a stable anchor, the user’s reasoning process would be compromised.

![Fig. 9. Cycles of human action in using visualizations for reasoning, from Liu and Stasko.](image)

6 What Makes Asynchronous Rendering Easy?

The previous analysis shed light on the success of Pilot 2’s design: since the results are shown in placeholders, the users had a stable anchor, and were able to offload the information to the screen as they are looking for new information. How do we transform interactive visualization into asynchronously loading placeholders?

The key to a stable representation of asynchronously loading results is that it captures history, as opposed to just the most recent result. We hypothesize that a visual buffer can provide an easy-to-reference visual memory of the user’s previous interactions, and their corresponding responses. This buffer serves as a short-term memory aid to help the user remember their interactions, and a way to form correspondences between their interactions and potentially late-arriving responses. While each response may cause quick and unexpected changes to the visualization, the visual buffer is intended to reestablish a stable and expected frame of reference.

In Pilot 1, the buffer size is one, where only the most recent interaction or response is shown. In Pilot 2, at the other extreme, an unbounded buffer the size of all possible interactions could hold all the distinct user requests and responses in a session. The pilots inform us that buffer size of one discourages concurrent interactions with asynchronous interfaces, and the unbounded buffer size could encourage concurrent interactions and improve task efficiency, for the specific task and visualization design chosen.

If the hypothesis is true, it must be the case that a modest-sized bounded buffer should have some positive effect on the usability of the interactive visualization when there is latency. Also, if the hypothesis is true, it must not be limited to loading visualizations.
in “placeholders”, but any visualization design that can convey a collection of results.

Visual interaction history is a well-known technique in different aspects of visual analysis. Direct encoding of interaction history overlaid on existing visualization provides a “footprint” to navigate information-saturated visualizations [48]. Thumbnails showing previous visualization states, and labels describing the actions performed to help users iterate on analysis and communicate findings [49].

Our challenge for supporting asynchrony has three parts: visualizing interaction history, displaying multiple visualization states corresponding to multiple interactions, and visualizing the correspondence between interaction requests and visualization responses so that users can pair them up intuitively. We first discuss the representation of interaction history, then review three techniques to show a history of visualizations using classical techniques of visual parallelism, and finally, we consider the establishment of correspondence between requests and responses.

6.1 Interaction History
The benefit of visualizing interactions is twofold: (1) it provides context to remind the user of the actions that caused the history in the response buffer and (2) every user interaction immediately updates the visualization of interaction history, which provides feedback to the user and acknowledges the interaction. Together with a spinner indicating progress, visualizing interaction history externalizes the current state of the visualization, making it easier for users to understand what is currently shown and what to anticipate.

How should one visualize the history of interactions? One attractive solution is to treat widgets as visualizations. In fact, enhancing widgets to be more than request-specifying tools is an idea that has been explored previously—“interactive widgets” turns legends into widgets [50], scented widgets annotate widgets with further information [51], and HindSight directly annotates the marks being interacted on with another visual encoding [48].

Here we adopt similar mechanisms, where the state of the interaction must be visualized explicitly by overlaying with an additional visual encoding (e.g., the facet in Figure 10 using color) and when needed, creating multiples of past states (e.g., the slider and brush widgets in Figure 10).

Fig. 10. Example visualization of recent request history for a facet (left), slider (top right) and brush (bottom right) widget. History is encoded by color where lighter the color means further in the past. Although facet elements can simply change their color, the slider handle and brush must show copies of themselves.

6.2 Multiple Visualization States
To visualize multiple visualization states, we can borrow techniques for simultaneous (“parallel”) plotting of multiple charts, e.g., the following description by Tufte in Visual Explanations, which we discuss in turn:

“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.”

Small Multiples:

Traditional interfaces update a single visualization in place, and hence do not provide “parallelism” in Tuft’s sense. One option is to use spatial parallelism to show each response visualization side-by-side, as in the small multiplexes design in Figure 11. When a new interaction response is received, instead of replacing the existing visualization, we simply render a scaled-down version of the response on the side. This can be effective for visualizations that are robust to scaling [52], [53] and has been shown to perform well as compared to alternatives such as animation [54].

Overlay:

Similar to small multiplexes, this design also renders past responses, however rather than rendering scaled-down responses side-by-side, we overlay the new response on top of the existing visualization. This design requires an available visual encoding (e.g., color, shape, texture, size, etc [55]). For instance, Figure 11 shows the use of overlays using color as visual encoding; this design can be effective when visual space is limited and must be balanced with increasing the complexity of the visualization.

![Fig. 11. Examples of overlay (left) and small multiples (right) design options. D’ is still loading, as indicated by the spinners.](image)

Animation:

When a new response arrives, instead of rendering directly, the response could be held back temporarily to ensure that the previous rendered response has had enough time on the screen for the user to read. Further, it could be held until the previous interaction’s response has been rendered. While the animation is perceptually more complex than the previous two techniques [54], [56], [57], it has been successfully used in popular visualization tools such as GapMinder [58].

6.3 Visualizing Request-Response Correspondence
A shared visual encoding can help users easily establish the correspondence between the two sequences of history that they perceive—requests and responses. Figure 11 is an example using color. This encoding could be any of the seven retinal variables: position, size, shape, value, color, orientation, and texture [55] so long as it does not conflict with existing encodings used.

7 Experiment
The use of the model by Liu and Stasko to examine the challenges of using asynchronously rendered visualizations suggest that there
are three primary reasons as to why asynchronous rendering can be difficult to use. Based on these three reasons, we identify three corresponding design factors that can affect how a user interacts with asynchronously rendered visualizations.

- **Visualization Designs**: As noted, a stable external anchor is necessary for the user to reason about a visualization. Consistent with the finding from our pilot studies, some visualization designs can better serve as anchors and afford the user to more effectively utilize asynchronous rendering.
- **Tasks**: Different information foraging tasks require the user to interact with a visualization differently. For example, tasks that require the user to locate information within a visualization would be easier for a user to perform compared to having to interact with a dynamic visualization for exploring new information.
- **Latency Profiles**: Being able to perform cognitive offloading is critical in allowing the user to perform complex tasks. When using asynchronous rendering where cognitive offloading is limited, a user’s cognitive resources can be further stressed if a system’s latency is high and the user needs to store information in their short-term memory for a longer period.

We conducted an experiment to evaluate the observations made in the previous section and their impact on the participants’ ability to use asynchronous rendering systems. Utilizing the three design factors identified above, the experiment uses a 3 (visualization design) x 3 (task) x 3 (latency profile) mixed factorial design. The within-subject parameter was the latency profile.

### 7.1 Experimental Conditions

We describe the choice and rational for the following three experimental factors, **Visualization Designs, Tasks, and Latency Profiles**, and their corresponding conditions:

**Visualization Designs**: In addition to the baseline design from the pilot experiments, we introduce two further asynchronous rendering behaviors for the experiment as treatment: **Small Multiples** and **Overlay**. The baseline (control) design uses blocking rendering behavior, as described in the first pilot study, which we refer to as **Blocking**. **Small Multiples** is inspired by the success of the Cumulative Asynchronous Rendering design studied in the second pilot experiment. **Overlay** is based on the first pilot, but with the added consideration of a stable external anchor. In particular, the user’s past interaction results are not immediately erased, which provides the necessary anchor for the user to see the effects of their new interactions in the context of the past ones.

- **Baseline (Control)**: We replicated the design used in the first pilot as the baseline condition, using “blocking” rendering, where only the most recent interaction result is displayed, and all previous concurrent interactions are canceled/rejected.
- **Small Multiples**: As discussed previously, when a new interaction response is received, instead of replacing the existing visualization, a scaled-down version of the response is rendered on the side, as shown in Figure 12. In our experiment, we limit the maximum number of multiples that are shown on screen to 4.
- **Overlay**: As discussed previously, similar to Small Multiples, this design also renders past responses, however rather than rendering scaled-down responses side-by-side, new responses are overlaid on top of the existing visualization. To support the overlay design, this experiment uses a line chart visualization of stock price data rather than the bar chart used in the pilot studies.

**Tasks**: In both the pilot studies, the participants were asked to “detect outliers” in a visualization. We refer to this task as **threshold** because the participants were given a threshold value and asked to identify whether any data element crosses over the threshold.

The **threshold** task was chosen for the pilots because it is relatively easy to complete, even with asynchronous rendering—identifying data elements above a threshold value does not require correct sequencing between a user’s interactions and the system’s responses. For our experiment, we added two treatments to test tasks that require increasing consideration for sequencing, **maximum** and **trend**. We describe all three below in the context of the experiment setup:

- **Threshold**: We include the original threshold task from the pilot studies as our baseline (control) condition. As noted, this task requires the least amount of consideration for sequencing. Specifically, we asked the participants, “Does any month have a stock price higher than 80 for the year 2010?”.
- **Maximum**: For this task, the participants are asked to identify the point with the maximum value in the visualization (“Which month had the highest stock price for the year 2010?”). This requires the participants to remember the largest value seen so far. Some consideration for sequencing is necessary because the participant would need to recognize the largest value and identify the corresponding interaction that results in that value.
- **Trend**: This task requires the participants to identify a trending pattern across multiple interactions (“What is the trend in stock price from Jan to Dec for the year 2010?”). This task is arguably the most difficult because it requires the participant to perform three actions: (1) read the data value, (2) identify the corresponding interaction, and (3) remember the sequencing order to identify if there is a trend.
Latency Profiles: Since users cannot cognitive offload past results to the screen due to unexpected asynchronous rendering, they will need to retain some information in their memory. Working set memory decays quickly [16], so the length of the latencies should have an impact on the user’s interaction patterns. Varying latencies thus could help explore to what degree users can make sense of asynchronous rendering—it is possible that a delay that is too long or too short would both discourage asynchronous interactions.

In our pilot, we found that beyond 5 seconds, the task becomes “painful” and “frustrating” for the participants with the baseline design. Hence we chose 5 seconds as an upper bound in the choices of the delay. It is plausible that there exists a number less or more than 5 seconds that could serve as the bound, but the focus of the experiment is to make an initial investigation into asynchronous interactions.

The latency profiles are random distributions to simulate high variance in network delays based on similar reports from Tableau [59], [60].

To this end, we tested 3 different latency conditions: (1) no latency, as baseline (control) (2) uniformly at random between 0 and 1 seconds, which we call low latency, as treatment, and (3) uniformly at random between 0 and 5 seconds, which we call high latency as another treatment.

The high latency condition is much higher than the upper bound of usable latency of 1 second in Liu and Heer’s visualization system [1], because our visual analytic tasks are simpler. The latency condition is about the same or shorter than that used in Zgraggen et al.’s experiment, which had 6 seconds and 12 seconds [8]. While we could also attempt a 12-second latency, it would be unnecessarily difficult for the participants for the tasks chosen. We discuss this further in the Limitations section.

7.2 Procedure

Similar to the pilot studies, the experiments were conducted online through Amazon Mechanical Turk. Participants were allowed a maximum of 60 minutes to complete the tasks. 50 participants were recruited for each combination of between-group parameters, for a total of 450 participants. Participants were 39% female and 61% male, 57% had a college degree or higher, and the average age was 35 years old.

Each experiment consisted of the participant going through the following procedure in order: training, real assignments, and survey, as explained below. We collected the same measures as in the pilots: accuracy, task completion time, and concurrency of interactions.

Training: Participants were first instructed on how to read and interact with the baseline visualization with no latency, then with low latency, and then high latency with one of Overlay or Small Multiples. The same dataset was used to ensure that participants focus on the change in the conditions.

Afterwards, participants were presented with a task question. The correct answers were shown after submission for comparison. Participants were shown three training assignments: first with blocking render design and no latency; then latency was introduced, then one of the two treatment designs was introduced with and without latency. Participants watched two short videos demonstrating contrasting interaction behaviors: one self-serializing, and one asynchronous. Participants were recommended to try interacting asynchronously.

Assignments: Each participant was randomly assigned to a specific task type and one of the two asynchronous rendering visualization design (Small Multiples or Overlay) as treatment. Each participant completed one of the combinations of design (treatment and baseline) and latency (2 by 3). The assignments were shuffled, so participants were not expecting at any point in the experiment what the conditions are for the next assignment. Participants did not know beforehand what the latency profile was for a task. No time limit was imposed per individual task, though participants were advised to complete tasks as quickly as they could. Unlike in the training setup, participants were not shown the correct answer before moving on to the next task.

After the main experiment, participants completed a survey asking whether they preferred the cumulative asynchronous rendering design or blocking rendering design (in simpler terminology), and to rate the task difficulty with the two designs. Responses were scored on a 5-point Likert scale, with space left for open-ended comments.

8 Results

User 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 than the parametric version while being almost as efficient when the underlying distribution is normal. For the survey results, we use the one-sample version of Wilcoxon test. We report z-value and p-value for the test, along with the medians of the two groups (C for controlled baseline, and T for treatment). When comparing across groups, the unsigned version of the test is ran and we report the U statistic. Non-statistically-significant p-values are reported with a parenthesis. We use notation \( N = \) to describe sample size when relevant. The Holm-Bonferroni correction is used to adjust the significance levels when considering multiple hypotheses. For simplicity, we take a conservative upper bound on the number of possible comparisons, and take an adjusted \( \alpha \) of 0.05/27 = 0.0019. As a data cleaning step, we removed all responses by participants who got the majority of the assignments wrong because they are suspected not to have been trying to complete the tasks in earnest (10%), assignments that took longer than two minutes (< 1%), and responses where the participant did not interact at all with the visualization (< 1%).

Consistent with the pilots, accuracy across the factors are high—average accuracies over all the latency, task, design groups were all above 95%. We ran the Wilcoxon signed rank sum with the Holm-Bonferroni correction to compare the accuracy of tasks completed across all the conditions, and found no statistically significant
difference. As a result, we do not report accuracy in the remainder of this section. However, completion times varied greatly, and are shown in Figure 14. We report statistics around this figure.

8.1 The Effect of Asynchronous Visualization Designs

Under high latency, users completed all three task types faster with the two asynchronous rendering designs (Small Multiples and Overlay) compared to baseline (Blocking). We report the statistics below, where the medians compare the treatment design (T) to baseline Blocking (C). Under low latency, Small Multiples and Overlay both show task completion time improvements, but the differences are not statistically significant after the Holm-Bonferroni adjustment is applied.

Under no latency, there are three pairs of “task” and “visualization design” conditions that show statistically significant differences between the treatment and the control (see table below). Interestingly, as opposed to the “high latency” condition above, the use of Small Multiples increase task completion time when compared to the control.

8.2 The Effect of Latency

Overall, even small amounts of latency introduced a noticeable increase in task completion time. However, it was smaller for the control conditions. Further, when we consider the condition Small Multiples in the third table below, we find that low latency does not have a significant effect on the completion of threshold and maximum.

We report significant statistics below, where the medians compare the low latency completion times (T) to that under the no latency (C) (Figure 14).

8.3 The Effect Of Tasks

While all the tasks are responsive to asynchronous rendering, there are meaningful differences. First, maximum has longer task completion time and lower concurrency in general. For example, the median concurrency of maximum is 0.51 (N=42) while threshold is 0.67 (N=44), under high latency, Overlay (U = 642, p<0.0075). Similar statistics are seen for Small Multiples.

Additionally, we were surprised by the results for trend. First, we had anticipated trend to be more difficult than maximum (i.e., longer completion time and more errors), but that was not the case, as users mostly completed it faster than maximum, and highly accurately (similar to threshold, with the exception being the first preceding table, when both maximum and threshold were not affected by latency for Small Multiples, trend was (with a significant p-value).

We discuss tasks more in the next section. Figure 15 combines across the designs and visualizes the behaviors discussed.

8.4 Usability Survey

We found a significant preference for the two treatment designs across different latencies and tasks. When asked to rate “How much did you prefer viewing one month of data at once versus multiple months of data at once?” from “Strongly prefer one month at once” to “Strongly prefer multiple months at once” at 5 (with “neutral” at 3), users responded positively to the asynchronous rendering designs (pseudo-median: 4.5, p<0.00001 against neutral null hypothesis).

Participants found both of the asynchronous rendering designs to help offset the task difficulty introduced by latency. When asked “For visualizations that allowed viewing only one month of data at once, how much did loading delay affect the difficulty of using the visualization?”, the estimated pseudo-median is 3.5 (“Large difficulty”), and for multiple months, the pseudo-median was 2.0 (“Slight difficulty”). For both statistics, p<0.00001, against the null hypothesis of “some difficulty” (3).

9 Discussion

As expected, the three factors tested in the experiment (visualization design, task, and latency profile) significantly affect the usability of asynchronous visualizations. Although accuracy is high across all conditions (similar to that of the pilots due to “self-serialization”), the differences in completion time provide a measure of how challenging the conditions are.
The Effects of the Three Factors: In general, as we hypothesized, for the factor “visualization design”, Small Multiples outperform Overlay, which outperforms the baseline (blocking render) condition. For the factor “latency profile”, not surprisingly, longer delays make the task more difficult.

One unexpected outcome is the effect of “tasks”. While we hypothesized that trend should be the most difficult task, followed by maximum and threshold, our results indicate that maximum takes the most amount of time (followed by trend). We observe that the reason is that when completing the trend task, participants do not always search through all the data before making a decision. For example, if the participant sees an upward trend between January, February, and March, they would declare that the trend is “increasing.” Conversely, for the maximum task, the participants needed to examine all data points before being able to submit an answer.

Beyond the unexpected effect of “tasks”, we also observe a few interesting findings.

First, the results suggest that the more difficult a condition (e.g., longer latency), the more asynchronous rendering could help alleviate the effect of latency. However, when the condition is too easy, asynchronous rendering can be detrimental. For example, as shown in Figure 14 with the maximum task but with no latency (red bars), the baseline condition outperforms Small Multiples. However, when latency increases (blue bars), both Small Multiples and Overlay outperform the baseline.

Second, although asynchronous rendering can improve task completion times, there is still room for better design. For example, in Figure 14, the use of Small Multiples make the completion times of the maximum task at low latency the same as when there’s no latency. This suggests that the participants are highly effective at utilizing asynchrony. However, this effect diminishes when latency increases. In all three tasks, and with both Small Multiples and Overlay outperform the baseline.

Third, our analysis of the relationship between concurrency and task completion rates (Figure 15) highlights the value of careful interface design. In our initial pilot, we found that despite an asynchronous interface, users “self-serialized” by waiting for the previous request to complete before triggering the next request. However, changing the design of the visualization, both encouraged users to make use of asynchrony to trigger concurrent requests and resulted in improved task completion times.

Cost of Asynchronous Rendering: As noted above, under the no latency condition, participants showed slightly higher task completion time when using the asynchronous rendering designs compared to the baseline design. This may be due to the extra cognitive burden of interacting with an unfamiliar interface. However, the user experience did not seem to deteriorate as evidenced by higher user preference for the asynchronous rendering designs in the survey responses. Using the concept of “cognitive flow” [61], we speculate that spending more time on a task makes it more challenging in a way that engages with the participants, yet waiting due to latency is disengaging and causes a worse experience.

This is consistent with comments participants shared. For the baseline, participants often used negative words such as “painful”, “frustrating”, “tedious”, and “awful” to describe assignments with high latency. Participants expressed 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 asynchronous rendering designs when there was 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 change as well—“Some of the tasks loaded really slow, [using baseline condition] got irritating waiting. Most of the [asynchronous rendering ] loaded fairly quickly.”.

These feedbacks suggest that when designed appropriately, asynchronous rendering can improve task completion time and increase user’s satisfaction. However, the cost of bad design can be high. If designed poorly, asynchronous rendering can be frustrating to use, even if it improves performance.

9.1 Limits of the Experiment
The visualization and tasks chosen are simple. It is unclear how asynchronous rendering technique will generalize to more complex visualizations, e.g., dashboards with multiple linked visualizations. In fact, to generalize this approach to multiple linked interactions would require further specification of the model and design, and is relevant future work. Also, if the tasks were exploratory and required more cognitive effort, we do not know how asynchronous rendering will affect user motivation and effectiveness. We plan to apply the design ideas formulated in this paper to more complex interactive visualizations, such as cross-filter, and evaluate more complex visual analytics tasks.

It is yet unclear how intuitive asynchronous rendering is to the user, especially when the visualization design is complex. However, the issue of additional complexity is a common problem faced by novel interaction designs, for instances error bars and animations in PVA designs [9], and the benefits may justifiy the costs.

9.2 Limits of the Design
In designing asynchronous visualizations, once the correspondence is mapped to any of Bertin’s visual channels, we can further examine the limits of design.

For example, in the case of Small Multiples, the limitation is simply the size of the canvas. With limited visual real estate, a system might not be able to show as many past interactions as the designer might like. On the other hand, the limitation of using Overlay is a little more nuanced. When the “recency” of a user’s interaction is encoded using intensity, hue, shape, size, orientation, or texture, the limiting factor is the user’s perceptual ability to effectively discriminate similar representations.

From a design standpoint, it becomes necessary to consider the number of past interactions that need to be shown. This number can be somewhat informed by the amount of latency in the system. For example, for a system with high latency where the system can be slow to respond to a user’s interaction, it might be necessary to show a large number of past user interactions. With such a number in mind, the designer can determine which visual channel can afford the needed perceptual discriminability. “Position” (i.e. the use of Small Multiples) can afford the highest number but comes at the cost of requiring visual real estate, whereas intensity allows for a smaller set of discriminable values but does not have the same constraint.

We plan to apply the design ideas formulated in this paper to more complex interactive visualizations, such as cross-filter, and evaluate more complex visual analytics tasks.
We also do not know how well asynchronous rendering would fare under longer latencies. We expect the technique to break down when the latency is much longer than the experiment conditions, e.g., 5 minutes. However, by allowing for a latency up to 5 seconds, we explored designs with an order-of-magnitude increase over interactive speeds, traditionally considered under 500 ms. In practice, these interactive speeds are quite challenging to deliver reliably, as the 95th percentile network latency exceeds 300 ms for WiFi networks even if data processing time is ignored. This requires a challenge for traditional vis tools when applied in Cloud and Big Data environments. Our approach can offer significant benefits in those increasingly common settings. We will add this discussion to experiment setup section. Also, our approach can be combined with other approaches, such as progressive visualization to reduce the latency for some initial results, or faster systems, to reduce the latency to a viable range.

10 Conclusions and Future Work

Interactive visualization research has traditionally focused on ways to minimize interaction response times, or otherwise assumed that response times are instantaneous. As data sizes continue to increase, and more data processing moves to cloud environments, network and data processing latencies will continue to become a reality and is important to be taken into consideration when designing future interactive visualization interfaces. Recent work highlights how latency can negatively affect visual exploration, and the need to study this aspect.

In this work, we have performed initial studies on the role of employing asynchrony in interactive visualizations when request latency is non-trivial. We have found that changing the UX to cumulatively render asynchronous results can support users’ utilization of asynchronous rendering, improving the perceived speed and usability of interactive visualizations. In addition, we propose an analytical framework for asynchronously rendered visualizations based on three factors—the visualization design, the user task, and the latency profile—and discuss the effect of the factors.

There are a lot more to be specified in the design space of asynchronous rendering, such as the size of the buffer, whether the results are visually ordered, and other encodings used for the history dimension. We also plan to perform follow up experiments described previously in the Discussion Section.

11 Acknowledgments

This work was supported by the National Science Foundation under Grant No. III-1564351.

References

[1] Z. Liu and J. Heer, “The effects of interactive latency on exploratory visual analysis,” IEEE transactions on visualization and computer graphics, vol. 20, no. 12, pp. 2122–2131, 2014.
[2] MapD. “Platform for lightning-fast sql, visualization and machine learning,” 2017. [Online]. Available: https://www.mapd.com/
[3] Graphistry. “Supercharge your investigations,” 2017. [Online]. Available: https://www.graphistry.com/
[4] J. M. Hellerstein, P. J. Haas, and H. J. Wang, “Online aggregation,” in ACM SIGMOD Record, vol. 26, no. 2. ACM, 1997, pp. 171–182.
[5] D. Fisher, I. Popov, S. 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.
[6] C. D. Stolper, A. Perer, and D. Gotz, “Progressive visual analytics: User-driven visual exploration of in-progress analytics,” IEEE Transactions on Visualization and Computer Graphics, vol. 20, no. 12, pp. 1653–1662, 2014.
[7] J.-D. Fekete, “Progressivis: A toolkit for steerable progressive analytics and visualization,” in 1st Workshop on Data Systems for Interactive Analysis, 2015, p. 5.
[8] E. Zograf, A. Galakatos, A. Crotty, J.-D. Fekete, and T. Kraska, “How progressive visualizations affect exploratory analysis,” IEEE Transactions on Visualization and Computer Graphics, 2016.
[9] D. Moritz, D. Fisher, B. Ding, and C. Wang, “Trust, but verify: Optimistic visualizations of approximate queries for exploring big data,” in Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. ACM, 2017, pp. 2904–2915.
[10] J.-D. Fekete, J. J. Van Wijk, J. T. Stasko, and C. North, “The value of information visualization,” in Information visualization. Springer, 2008, pp. 1–18.
[11] Z. Liu and J. Stasko, “Memory models, visual reasoning and interaction in information visualization: A top-down perspective,” IEEE transactions on visualization and computer graphics, vol. 16, no. 6, pp. 999–1008, 2010.
[12] S. K. Card, A. Newell, and T. P. Moran, “The psychology of human-computer interaction,” 1983.
[13] S. M. Kosslyn, “Understanding charts and graphs,” Applied cognitive psychology, vol. 3, no. 3, pp. 185–225, 1989.
[14] N. Cowan, “The magical mystery four: How is working memory capacity limited, and why?” Current directions in psychological science, vol. 19, no. 1, pp. 51–57, 2010.
[15] J. Brown, “Some tests of the decay theory of immediate memory.” Quarterly Journal of Experimental Psychology, vol. 10, no. 1, pp. 12–21, 1958.
[16] D. H. Ballard, M. M. Hayhoe, and J. B. Pelz, “Memory representations in natural tasks,” Journal of Cognitive Neuroscience, vol. 7, no. 1, pp. 66–80, 1995.
[17] H. R. Lipford, F. Stukes, W. Dou, M. E. Hawkins, and R. Chang, “Helping users recall their reasoning process,” in Visual Analytics Science and Technology (VAST), 2010 IEEE Symposium on. IEEE, 2010, pp. 187–194.
[18] M. A. Borkin, A. A. Vo, Z. Bylinskii, P. Isola, S. Sunkavalli, A. Oliva, and H. Pfister, “What makes a visualization memorable?” IEEE Transactions on Visualization and Computer Graphics, vol. 19, no. 12, pp. 2306–2315, 2013.
[19] E. L. Hutchins, J. D. Hollan, and D. A. Norman, “Direct manipulation interfaces,” Human–Computer Interaction, vol. 1, no. 4, pp. 311–338, 1985.
[20] S. C. Seow, Designing and engineering time: The psychology of time perception in software. Addison-Wesley Professional, 2008.
[21] D. H. Maister et al., The psychology of waiting lines. Harvard Business School Boston, MA, 1984.
[22] K. L. Katz, B. M. Larson, and R. C. Larson, “Prescription for the waiting-in-line blues: Entertain, enlighten, and engage,” MIT Sloan Management Review, vol. 32, no. 2, p. 44, 1991.
[23] J. Johnson, GUI bloopers 2.0: common user interface design don’ts and don’ts. Morgan Kaufmann, 2007.
[24] B. A. Myers, “The importance of percent-done progress indicators for computer-human interfaces,” in ACM SIGCHI Bulletin, vol. 16, no. 4. ACM, 1985, pp. 11–17.
[25] C. Harrison, Z. Yeo, and S. E. Hudson, “Faster progress bars: manipulating perceived duration with visual augmentations,” in Proceedings of the SIGCHI conference on human factors in computing systems. ACM, 2010, pp. 1545–1548.
[26] M. R. Ebling, B. E. John, and M. Satyanarayanan, “The importance of translucence in mobile computing systems,” ACM Transactions on Computer-Human Interaction (TOCHI), vol. 9, no. 1, pp. 42–67, 2002.
[27] H. Lam, “A framework of interaction costs in information visualization,” IEEE transactions on visualization and computer graphics, vol. 14, no. 6, 2008.
[28] S. Greenberg and D. Marwood, “Real time groupware as a distributed system: concurrency control and its effect on the interface,” in Proceedings of the 1994 ACM conference on Computer supported cooperative work. ACM, 1994, pp. 207–217.
[29] Y. Wu, J. M. Hellerstein, and E. Wu, “A devilish approach to inconsistency in interactive visualizations.” in HILDA® SIGMOD, 2016, p. 15.
[30] W. K. Edwards, E. D. Myatt, K. Petersen, M. J. Spreitzer, D. B. Terry, and M. M. Theimer, “Designing and implementing asynchronous collaborative applications with bayou,” in Proceedings of the 10th annual ACM symposium on User interface software and technology. ACM, 1997, pp. 119–128.
[31] P. Dourish and S. Bly, “Portholes: Supporting awareness in a distributed work group,” in Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, 1992, pp. 541–547.

[32] C. Gutwin, “Traces: Visualizing the immediate past to support group navigation cues,” in Graphics interfaces, 2002, pp. 43–50.

[33] C. Gutwin, T. Graham, C. Wolfe, N. Wong, and B. De Alwis, “Gone but not forgotten: designing for disconnection in synchronous groupware,” in Proceedings of the 2010 ACM conference on Computer supported cooperative work. ACM, 2010, pp. 179–188.

[34] C. Savery and T. Graham, “It’s about time: confronting latency in the development of groupware systems,” in Proceedings of the ACM 2011 conference on Computer supported cooperative work. ACM, 2011, pp. 177–186.

[35] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, “Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals,” Data mining and knowledge discovery, vol. 1, no. 1, pp. 29–53, 1997.

[36] C. Stoltte, D. Tang, and P. Hanrahan, “Multiscale visualization using data cubes,” IEEE Transactions on Visualization and Computer Graphics, vol. 9, no. 2, pp. 176–187, 2003.

[37] L. Lins, J. T. Klosowski, and C. Scheidegger, “Nanocubes for real-time exploration of spatiotemporal datasets,” TVCG, 2013.

[38] Z. Liu, B. Jiang, and J. Heer, “immens: Real-time visual querying of big data,” in Computer Graphics Forum, vol. 32, no. 3pt4. Wiley Online Library, 2013, pp. 421–430.

[39] M. El-Hindi, Z. Zhao, C. Binnig, and T. Kraska, “Vistrees: fast indexes for interactive data exploration,” in Proceedings of the Workshop on Human-In-the-Loop Data Analytics. ACM, 2016, p. 5.

[40] C. E. Weaver and M. Livny, “Improving visualization interactivity in java,” in SPIE Human-Computer Interaction, 2000, pp. 62–72.

[41] S.-M. Chan, L. Xiao, J. Gerth, and P. Hanrahan, “Maintaining interactivity while exploring massive time series,” in Visual Analytics Science and Technology, 2008. VAST’08: IEEE Symposium on. IEEE, 2008, pp. 59–66.

[42] H. Piringer, C. Tominski, P. Muigg, and W. Berger, “A multi-threading architecture to support interactive visual exploration,” IEEE Transactions on Visualization and Computer Graphics, vol. 15, no. 6, pp. 1113–1120, 2009.

[43] E. Czaplicki and S. Chong, “Asynchronous functional reactive programming for gui’s,” in ACM SIGPLAN Notices, vol. 48, no. 6. ACM, 2013, pp. 411–422.

[44] E. G. Hetzler, V. L. Crow, D. A. Payne, and A. E. Turner, “Turning the bucket of text into a pipe,” in Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on. IEEE, 2005, pp. 89–94.

[45] J. C. Roberts, “State of the art: Coordinated & multiple views in exploratory visualization,” in Coordinated and Multiple Views in Exploratory Visualization. 2007. CMV’07. Fifth International Conference on. IEEE, 2007, pp. 61–71.

[46] D. Guse, S. Schuck, O. Hohlfeld, A. Raake, and S. Möller, “Subjective quality of webpage loading: The impact of delayed and missing elements on quality ratings and task completion time,” in Quality of Multimedia Experience (QoMEX), 2015 Seventh International Workshop on. IEEE, 2015, pp. 1–6.

[47] J. Hollan, E. Hutchins, and D. Kirsh, “Distributed cognition: toward a new foundation for human-computer interaction research,” ACM Transactions on Computer-Human Interaction (TOCHI), vol. 7, no. 2, pp. 174–196, 2000.

[48] M. Feng, C. Deng, E. M. Peck, and L. Harrison, “Hindsight: Encouraging exploration through direct encoding of personal interaction history,” IEEE Transactions on Visualization and Computer Graphics, vol. 23, no. 1, pp. 351–360, 2017.

[49] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala, “Graphical histories for visualization: Supporting analysis, communication, and evaluation,” IEEE transactions on visualization and computer graphics, vol. 14, no. 6, 2008.

[50] N. H. Riche, B. Lee, and C. Plaisant, “Understanding interactive legends: a comparative evaluation with standard widgets,” in Computer graphics forum, vol. 29, no. 3. Wiley Online Library, 2010, pp. 1193–1202.

[51] W. Willett, J. Heer, and M. Agrawala, “Scented widgets: Improving navigation cues with embedded visualizations,” IEEE Transactions on Visualization and Computer Graphics, vol. 13, no. 6, pp. 1129–1136, 2007.

[52] J.-D. Fekete and C. Plaisant, “Interactive information visualization of a million items,” in Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on. IEEE, 2002, pp. 117–124.

[53] J. Heer, N. Kong, and M. Agrawala, “Sizing the horizon: the effects of chart size and layering on the graphical perception of time series visualizations,” in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2009, pp. 1303–1312.

[54] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko, “Effectiveness of animation in trend visualization,” IEEE Transactions on Visualization and Computer Graphics, vol. 14, no. 6, 2008.

[55] J. Bertin, “Semiology of graphics: diagrams, networks, maps,” 1983.

[56] T. Munzner, Visualization analysis and design. CRC Press, 2014.

[57] B. Tversky, J. B. Morrison, and M. Betrancourt, “Animation: can it facilitate?” International journal of human-computer studies, vol. 57, no. 4, pp. 247–262, 2002.

[58] GapMinder, “Gapminder,” 2017. [Online]. Available: http://www.gapminder.org/

[59] Tableau, “Tableau server scalability - a technical deployment guide for server administrators,” 2017. [Online]. Available: https://www.tableau.com/learn/whitepapers/tableau-server-scalability-technical-deployment-guide-server-administrators

[60] ———, “Designing efficient workbooks,” 2017. [Online]. Available: https://www.tableau.com/learn/whitepapers/designing-efficient-workbooks

[61] J. Nakamura and M. Csikszentmihalyi, “The concept of flow,” in Flow and the foundations of positive psychology. Springer, 2014, pp. 239–263.