THE EFFECTS OF LATENCY ON 3D INTERACTIVE DATA VISUALIZATIONS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Allen Korenevsky

June 2016

© 2016
Allen Korenevsky
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: The Effects of Latency on 3D Interactive
Data Visualizations

AUTHOR: Allen Korenevsky

DATE SUBMITTED: June 2016

COMMITTEE CHAIR: Zoë Wood, Ph.D.
Associate Professor of Computer Science

COMMITTEE MEMBER: Alex Dekhtyar, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Franz Kurfess, Ph.D.
Professor of Computer Science

ABSTRACT

The Effects of Latency on 3D Interactive Data Visualizations

Allen Korenevsky

Interactive data visualizations must respond fluidly to user input to be effective, or so we assume. In fact it is unknown exactly how fast a visualization must run to present every facet within a dataset. An engineering team with limited resources is left with intuition and estimates to determine if their application performs sufficiently well.

This thesis studies how latency affects users' comprehension of data visualizations, specifically 3D geospatial visualizations with large data sets. Subjects used a climate visualization showing temperatures spanning from the 19th to the 21st century to answer multiple choice questions. Metrics like their eye movements, time per question, and test score were recorded. Unbeknownst to the participants the latency was toggled between questions, subjugating frame rendering times to intervals between $33\frac{1}{3}$ and 200 milliseconds.

Analysis of eye movements and question completion time and accuracy fail to show that latency has an impact on how users explore the visualization or comprehend the data presented. User fixation times on overlaid 2D visualization tools however are impacted by latency, although the fixation times do not significantly differ over 3D elements. The finding speaks to how resilient users are in navigating and understanding virtual 3D environments — a conclusion supported by previous studies about video game latency.

ACKNOWLEDGMENTS

Thank you to those who helped this thesis become a reality. Dr. Zoë Wood for guiding this project, and me, for close to a year. The anonymous participants who donated half an hour of time to the experiment. Dr. Andrew Schaffner for help with the statistical analysis. Finally, my family and Shoshana for support and encouragement throughout.

TABLE OF CONTENTS

	Page
LIST OF TABLES	ix
LIST OF FIGURES	x
CHAPTER	
1 Introduction	1
2 Background	4
2.1 Real-Time Data Visualizations	4
2.2 Latency	5
2.3 Latency vs. Frames Rate	5
2.4 Gaze Tracking	6
3 Related Work	8
3.1 Search Engines	8
3.2 Video Games	10
3.3 Data Visualizations	13
3.4 Web Page Loading	14
3.5 Fitts' Law	14
4 Climate Visualization	16
4.1 Temperature and Weather Station Data	18
4.2 Visualization Tools	19
4.2.1 3D Navigation	19
4.2.2 Timeline	19
4.2.2.1 Timeline Settings	20
4.2.2.2 Additions and Changes	20
4.2.3 Legend	22
4.2.4 Histogram and Station Selector	23
4.2.4.1 Bug Fix for Station Selector	23
4.3 Build Tools	24
4.3.1 Webpack	24
4.3.2 Node Package Manager	25

4.4	Speed Optimizations	25
4.4.1	Memory Management	26
4.4.2	Spatial Data Structure	27
4.4.3	Throttling Graphics Updates	29
4.4.4	Asynchronous Data Loading	30
4.4.5	Cesium API Settings	31
5	Latency Experiments	32
5.1	Latency Conditions	32
5.2	Hypothesis	32
5.3	Equipment	33
5.3.1	Hardware	33
5.3.2	Software	34
5.4	Visualization Modifications	35
5.4.1	Toggling Animation Latency	36
5.4.2	Experiment Control Panel	36
5.4.3	Server	37
5.5	Procedure	38
5.6	Sample Pool	39
5.7	Experiment Variables	39
5.8	Experiment Errors	40
6	Analysis	42
6.1	Data Parsing	42
6.2	Question Analysis	43
6.2.1	Duration of Question Completion Analysis	44
6.2.2	Comprehension Test Score Analysis	45
6.3	Gaze Analysis	46
6.3.1	Heat Map	47
6.3.1.1	Heat Map Over The Globe	48
6.3.1.2	Heat map Over 2D Widgets	49
6.3.2	Fixations	49
6.3.2.1	Globe Fixations	50
6.3.2.2	Widget Fixations	50

7 Conclusion and Future Work	60
BIBLIOGRAPHY	62
APPENDICES	
A Comprehension Test Questions	65

LIST OF TABLES

Table		Page
5.1	The hardware specifications for the two computers used in the study. Only the Windows computer actually ran the visualization during the experiments.	33
5.2	The software versions used in the experiment to test users and capture metrics for later analysis.	35
6.1	Summary statistics for the question duration by latency level. . . .	45

LIST OF FIGURES

Figure	Page
2.1 How frames per second change in response to latency.	6
3.1 The impact latency has on daily searches over time, as found by a Google study.	9
3.2 The results from Bing latency experiments. Dashes indicate that there was no statistically significant change.	10
3.3 The negative correlation between latency and target hit ratio in the first-person shooter study.	11
3.4 Exploring time increases as latency gets higher in Warcraft III, a real-time strategy video game.	12
3.5 The data visualization that Liu and Heer had participants use. They added up to half a second of latency for all operations. Participants were given the open ended exploration task with geospatial check-in data.	13
4.1 An overview of the historical climate visualization. The image at the top shows a zoomed out view, giving a general sense of temperatures in the United States. The bottom left image shows the histogram selector tool being used, producing a distribution of temperates. The final image shows the result for searching “Germany” in the search widget.	17
4.2 Cesium’s default time dial on the left compared with the modified version on the right. The time ring and the button to reset the time to the computer’s clock were removed. The text with the time and animation speed is likewise hidden. Some other visual edits to the play, pause, and rewind buttons were made as well.	21
4.3 The timeline tooltip, which shows the month and year depending on the cursor’s position. The tooltip only appears when the cursor is hovering over the timeline.	22
4.4 Selecting just part of the earth that is visible with the view frustum. The calculated rectangle’s coordinates are used to select all weather stations that lie within the highlighted region on Earth.	28

5.1	A typical experiment setup. The photo on the left shows the subject's view before the experiment. The mouse, keyboard, gaze tracker, and monitor are connected to the black Windows laptop. The right image shows the experimenter's view. The Mac has the experiment control panel while the Windows PC is running the Mirametrix gaze tracking software.	34
6.1	The spread of question completion durations before and after the outlier was filtered out.	43
6.2	Question completion durations before and after logarithmic transform to make it symmetrical, a prerequisite for ANOVA p-values to be accurate.	44
6.3	A logistic regression plot, along with an integrated histogram. The line estimates the test score based on the frames per second. While the score slightly increases as the latency is reduced, this slope is too minor to be significant. The histogram bars give a better sense of data quantities than a traditional logistic regression plot [20]. . . .	46
6.4	A comparison of every latency level. Only items with a confidence interval all positive or negative are significant results. Here, 5-20 and 10-20 are significant. The Tukey test makes sure that the overall significance level is 95%, making each individual confidence interval wider to compensate.	52
6.5	Histograms showing that the logarithmically transformed fixation duration data is roughly symmetrical. The geometric mean is a good approximation of the median because of this.	53
6.6	Global heat map for questions about weather stations appearing and disappearing over time.	53
6.7	Global heat map for questions about temperature changes over a decade.	54
6.8	Global fixation map for questions about temperatures at specific dates.	54
6.9	Visualization widget heat map for questions about weather stations appearing and disappearing over time.	55
6.10	Visualization widget heat map for questions about decade temperature changes.	55
6.11	Visualization widget heat map for questions about temperatures at specific dates.	56
6.12	Global fixation map for questions about weather stations appearing and disappearing over time.	56

6.13	Global fixation map for questions about temperature changes over a decade.	57
6.14	Global fixation map for questions about temperatures at specific dates.	57
6.15	Visualization widget fixation map for questions about weather stations appearing and disappearing over time.	58
6.16	Visualization widget fixation map for questions about temperature changes over a decade.	58
6.17	Visualization widget fixation map for questions about temperatures at specific dates.	59

Chapter 1

INTRODUCTION

Real-time 3D data visualizations present information ranging from medical data [7] to Twitter trends [14]. Visualizations are regarded as the next step for Big Data [4], specifically, the development of tools to analyze readings produced by the myriad of sensors monitoring the world.

For these multifaceted and complex interactive data visualizations, the primary challenge is keeping the response times to a minimum. The visualizations simulate motion by showing dozens of frames per second, much like how a film in a movie theater is projected. An interactive data visualization however must respond to stochastic user input. Rendering each view at 60 frames per second gives visualizations a mere ten milliseconds of computation time [11]. There are many ways that software engineers can optimize visualizations for speed, but programming talent is expensive. The question becomes: how slow can a data visualization run while still communicating its data to users?

It turns out that latency does not affect all application types equally. For tasks like a web search, users show less engagement with millisecond differences in response time [3] [2] [17]. For video games however, latency does not have significant effects on user performance [1] [19]. Previous work studying latency in data visualizations only reviewed 2D visualizations, and used a fixed latency rather than attempting to pinpoint a threshold [12]. It is unknown if there is a performance mark developers can target for making interactive 3D data visualizations effective enough.

In an effort to find performance thresholds, this project explored the effect of latency on a user's experience while exploring a visualization of three-dimensional

data. Our goal was to quantify whether a user’s ability to answer questions about this data was affected by latency. Participants completed a comprehension test using a 3D geospatial data visualization of the Earth’s climate from the late 19th century to 2013. The visualization has a number of 2D tools overlaid on a 3D globe to help users navigate the environment and find information about temperatures in different regions of the globe. The visualization’s frame rate was toggled between 30, 20, 10, and 5 frames per second between certain questions. Along with gaze positions captured by an eye tracking device, the question performance was analyzed to see how responses varied and how visualization tool use changed depending on the amount of latency users faced.

This study fails to show latency having any significant impact on test score, test completion time, or gazing patterns. An analysis of gaze fixation times shows there is a statistically significant difference in median fixation durations when users are looking at overlaid 2D visualization widgets. For gazes in the 3D environment however the fixation durations do not significantly differ.

These results, in combination with the previous latency work, indicate that user performance is robust in three dimensional environments under latency. The fixation analysis along with other papers studying search engine and web page latency show that user behavior is more susceptible to change when using two dimensional tools and products experiencing high latency. We attribute this to the different nature of these visualization types. The 2D visualisations are presenting information in response to a single request, lacking a prior frame of reference that 3D spatial visualizations provide.

For teams optimizing a spatial visualization’s performance, the results indicate that the 3D environment can experience a surprisingly large amount of lag — as high as 200 milliseconds — and keep user performance relatively constant. The findings

do not reveal how user sentiment or exploratory data analysis changes in response to latency.

This thesis presents the following contributions:

- Presents evidence as to which types of applications need to minimize latency to ensure user comprehension
- Details ways to analyze gaze tracking data for a 3D application
- Outlines performance enhancing techniques for a 3D data visualization when concerned with latency

Chapter 2

BACKGROUND

This thesis builds on previous work in data visualizations and real-time interaction. We present background material related to both areas and discuss the important correlation between latency and frame rate. We also present how gaze tracking can reliably capture where users are looking on a computer monitor.

2.1 Real-Time Data Visualizations

Data visualizations are tools to analyze information as a graphic. There are hundreds of cataloged visualizations, including bar charts, pie charts, line plots, and histograms. Commonplace tools like maps and historical timelines are also data visualizations [6].

This thesis deals with two general types of visualizations: information and spatial visualizations. The information visualizations are two-dimensional and typically service a single request. Features like the timeline tool and search in the visualization created for this thesis are both examples of this type.

A spatial visualization is usually three-dimensional and contains data that has some inherent spatial mapping, for example temperatures measured at a particular location on the globe or tissue measurements at specific points on the human body. Spatial visualizations render a constant stream of information to the user, making the engagement a continuous series of requests rather than single separate requests. The graphics rendering is calculated in real-time to service a change in the view of the data (i.e. camera movement). Section 2.3 discusses rendering thresholds to make visualizations appear real-time.

2.2 Latency

Latency is the round trip time between an input and a response. The term is often used in networks as the time between a data and an acknowledgement packet between two communicating computers. In the video game world latency is often referred to as “lag”.

2.3 Latency vs. Frames Rate

In this thesis the visualization’s latency is changed by setting the number of frames per second that are rendered. Calculating the latency between frames is as simple as $\frac{1}{FPS}$. If the visualization is showing 30 frames each second, than there are $33\frac{1}{3}$ milliseconds of latency between each frame. Because of the inverse relation, only about 17 milliseconds of additional latency change a frame rate from 60 to 30 frames per second. However, 100 milliseconds of latency are needed to reduce the frame rate from 10 to 5 frames per second. The relationship is shown in figure 2.1.

The ideal frame rate for any application matches the screen refresh rate on the device. Most computer monitors and phones refresh the screen buffer 60 times a second, which means that the ideal frame rate should be 60 frames per second [11].

For applications that cannot maintain 60 frames per second, there is a risk of not appearing real-time. The human eye can detect up to 12 separate images shown a second [16], so a visualization must exceed this to appear fluid. When a person sees separate images rather than continuous animation, the visualization is no longer real-time. It is likely in this scenario that a visualization can no longer present its data effectively to a user.

Figure 2.1: How frames per second change in response to latency.

2.4 Gaze Tracking

The goal of gaze tracking is to glean what a user is thinking [5]. By knowing where a user looked, how frequently, and for how long, we can interpolate their thoughts. Gazing at 2D tools more frequently under low latency conditions rather than high ones for example tells us that usage declines when latency gets higher.

Gaze tracking uses a user's pupil position to determine where they are looking. The device shines an ultraviolet light that creates a glint on a subject's eye. The glint can be reliably tracked, and the pupil is found nearby the glint. The gaze vector is extrapolated and the intersection point with a perpendicular plane representing the monitor is calculated. The calculation can produce accurate results, however even the best systems have some margin of error in measurements.

A known limitation of gaze tracking is that many gazes are not intentional. By using heat maps, the gazes are aggregated by density so the most looked at features stand out in later analysis. An example of this is testing how effective a website's layout is by generating a heat map of where users looked most frequently [5].

Chapter 3

RELATED WORK

There have been several studies of how delays affect users at different tasks, conducted both by corporations and universities. The goal is to see if and how latency impacts a product’s usage or the user’s performance.

The previous work is overall inconclusive. Latency has a measurable impact on how much subjects use search engines [3] [2] [17], but for video games it does not significantly affect 3D environment navigation or competitive match performance [1] [19]. None of these studies analyzed 3D data visualizations in depth, or recorded metrics like gaze position. Theoretically, latency should inhibit task completion. If costs are added to each step of movement, then exploration and navigation is hindered. This phenomenon is described by Fitts’ Law; a human-computer interaction concept that quantifies the difficulty of human movement [13].

3.1 Search Engines

Google has published several articles detailing users’ sensitivity to latency when loading search results. The first study had users compare two mock search engines which differed only in loading speed of search results. The “control” search engine had a fixed latency of 250 milliseconds while the “test” search engine had a latency of either 2, 3, 4, or 5 seconds. The study found that when latency was greater than three seconds, users chose the faster search engine 1.5 times more often. However, users’ stated preference was inconclusive. The authors acknowledged a shortcoming with the study — users could perceive sub-second latency, so delays lasting several seconds did not impact users in different ways [3].

A second Google study imposed more granular delays to precisely measure how users react to sub-second latency changes. Conducted on the live Google Search page, a small subset of users had their search results delayed anywhere from 50 to 400 milliseconds. Adding 100 milliseconds of latency measurably reduced the number of queries users issued. Over the six week duration of the experiment, the number of searches continuously declined, as figure 3.1 shows. This paper highlights the measurable performance impacts of slight latency differences [2].

Figure 3.1: The impact latency has on daily searches over time, as found by a Google study.

Latency cost was also observed in the Bing search engine. Microsoft performed a study similar to Google's on their live page, albeit with more extensive testing.

The artificial latency ranged from 50 to 2000 milliseconds. Figure 3.2 shows the experiments results. Latency at 200 milliseconds had a measurable impact, while delaying by two seconds significantly changed users' behavior. For example, with two seconds of latency it took a user an extra three seconds to select a search result. Bing's revenue only began to decline with half a second of latency however. This paper reinforces Google's findings users' sensitivity to latency while using web search engines [17].

	Distinct Queries/User	Query Refinement	Revenue/User	Any Clicks	Satisfaction	Time to Click (increase in ms)
50ms	-	-	-	-	-	-
200ms	-	-	-	-0.3%	-0.4%	500
500ms	-	-0.6%	-1.2%	-1.0%	-0.9%	1200
1000ms	-0.7%	-0.9%	-2.8%	-1.9%	-1.6%	1900
2000ms	-1.8%	-2.1%	-4.3%	-4.4%	-3.8%	3100

Figure 3.2: The results from Bing latency experiments. Dashes indicate that there was no statistically significant change.

3.2 Video Games

In a video game setting, unlike with search engines, users appeared to be more resilient to latency. This was true across two different playing paradigms: first-person shooter and real time strategy games.

A first-person shooter game places users in control of an avatar. This avatar navigates and interacts with a three dimensional world. A player's objective in the studied game, Unreal Tournament 2003, was to eliminate the enemy opponents using different weapons. The paper found that latency's effect was attenuated. Navigation

Figure 3.3: The negative correlation between latency and target hit ratio in the first-person shooter study.

skill in the three dimensional world and match performance were independent of latency experienced by the players.

The latency was controlled by network speed, and the client program had the ability to interpolate movements while server communication was strained. Unfortunately, the frame rate during the tests was not published. Imposed network latency never exceeded 400 milliseconds.

Latency degraded players' precision in using weapons, however. The results of this performance decline are diagrammed in figure 3.3. It shows that 300 milliseconds of additional latency reduced a player's hit rate by approximately 30%. This study indicates that in 3D environments, only operations that require a high level of

dexterity are impacted by the delays [1].

In real-time strategy games latency minimally affected player behavior as well. This game genre focuses on long term planning, where players build small cities, empires, or ecosystems and aim to control the maximum amount of territory needed to win the game. In a study of Warcraft III, the authors separated interaction types into building, exploring, and attacking. Only exploring was hindered by added latency, as shown in figure 3.4. The correlation was only modest — close to 40% of the variation in exploration time could not be attributed to latency.

Figure 3.4: Exploring time increases as latency gets higher in Warcraft III, a real-time strategy video game.

Players’ resilience to latency, the study explained, is due to traits of real-time strategy games. Since victory requires long-term planning, latency does not affect prolonged decision making. However, similar results from the first-person shooter study seem to indicate that resilience to latency is not specific to a game type [19].

3.3 Data Visualizations

The work by Liu and Heer finds that latency of 500 milliseconds has measurable costs on user exploration in a 2D interactive data visualization. The work divides operations a user can perform in a typical data visualization into four categories: selecting, panning, zooming, and brushing and linking. The paper found brushing and linking to be the most sensitive to added latency. Brushing and linking is when a change in one view of the data affects a separate view. For the visualization, drawing a rectangle on the map caused the histograms to update with selected items. Overall, latency negatively impacted the entire user’s analysis.

Figure 3.5: The data visualization that Liu and Heer had participants use. They added up to half a second of latency for all operations. Participants were given the open ended exploration task with geospatial check-in data.

This study is the only one found focused on interactive data visualizations, and inspired this thesis, however the fixed 500 millisecond latency it employed leaves the reader wondering what is the actual latency threshold for visualization tools to remain effective. The custom data visualization used in the study is shown in figure 3.5 [12].

3.4 Web Page Loading

The importance of graphical responsiveness was shown in a study of web page loading speed. Sears et al. show that web pages with text and graphics get more favorable reviews from subjects when they are loaded quicker. This study was conducted in the 1990s, so its results may be somewhat out of date. Yet, it does find an actual difference in preference due to latency, rather than user performance impacts [18].

3.5 Fitts' Law

Latency theoretically increases the time it takes to navigate from source to target node in an environment, since it adds a cost to each step of the overall movement. This is described by a formula known as Fitts' Law:

$$ID = a + b \log_2 \frac{2D}{W}$$

where ID stands for index of difficulty, D is the distance to a target and W is the target width. The a and b terms are model parameters that change how difficult the task is. The b term is dependent on the user's mobility skills and on how difficult a device or software makes it to navigate to a target. The a term is a constant depending on the problem domain.

Consider an example of a user moving a computer mouse from one end of the screen to a colored target on the other end. Normally, the user guesses the best path to move the mouse, and continuously corrects errors using the device until the target is reached. High latency delays feedback the user receives from their movements, making them operate on out of date information. The user either has to slow down to wait for movement updates or guess how to move the mouse based on old information. This adds time to complete the task.

For data visualizations, latency should increase the time it takes to discover insights and may even prevent the user from discovering any meaning in a data set. Fitts' law has been used to analyze what effect input latency has on completing simple tasks, and has been found to affect eye movements as well as mouse movements [13] [22].

Chapter 4

CLIMATE VISUALIZATION

To explore the effects of latency in 3D data visualizations, a custom visualization was developed so different visualization components, both spatial and informational, can be tested at once. The software system developed for this thesis combines 3D and 2D visualization primitives to fully exercise interaction methods and separate out behavior patterns as the latency changes.

The visualization plots over 6,000 weather stations on a 3D cartographic globe. Each station's color varies depending on the recorded monthly average temperature. A timeline tool allows users to change the month and year, which in turn updates the weather stations' color. The timeline spans over a century, so weather stations appear and disappear over time on Earth as they were installed and dismantled. For active stations, some months had invalid data. The visualization hides these stations until the temperatures are accurate again.

The visualization is built with Cesium, a web rendered globe with an API for adding geometry, changing the time, and navigating the world. The Cesium platform was picked based on the recommendation from a study on popular 3D geospatial visualization tools [10]. Cesium is a client side library — it is designed to run in a web browser. It is written in JavaScript and uses modern web technologies like WebGL and Web Workers to keep the interface as fast as possible.

Weather station data was downloaded from a public NASA database and is stored in a JSON encoded file. When the visualization is launched, it parses the data and then draws the weather stations on the globe. Only stations with recorded temperatures on the start date in the timeline widget are drawn. The weather stations

are drawn as billboards, which means they always face the camera and change size depending on the camera's zoom level. They grow larger when the camera zooms out and shrink when zooming in. The weather station color palette is based on NASA's climate data visualizations, ranging from purple, blue, white, and red. Some screen shots of the visualization are available in figure 4.1.

Figure 4.1: An overview of the historical climate visualization. The image at the top shows a zoomed out view, giving a general sense of temperatures in the United States. The bottom left image shows the histogram selector tool being used, producing a distribution of temperatures. The final image shows the result for searching “Germany” in the search widget.

Most of the development time went into improving the program performance and enhancing the provided tools. The most notable effort is a bug fix in the Cesium library itself, described in section 4.2.4, which was merged into a new Cesium release.

The visualization does not effectively show global warming. This is because climate change visualizations show relative temperature differences from a base period, often an average from 1959 to 1980. The color scale reflects margins of a few degrees Celsius, rather than absolute temperatures shown in this visualization. The scientific visualizations also do more pre-processing to the data, including temperature averaging by bins and various normalization stages [21].

Although it uses standardized web technologies, this visualization is only tested against Google Chrome. Since the experimental software is entirely controlled, the browser is a fixed variable. The visualization will likely work in other browsers but there is no guarantee for performance.

4.1 Temperature and Weather Station Data

The data for this visualization is made available courtesy of NASA’s Goddard Institute for Space Studies (GISS) specifically the Earth Sciences Division. Data was downloaded from the GISS website using a Python script, which scraped temperature readings for each weather station and saved it into a JSON file. Metadata about each weather station — the cartographic coordinates, city, country, and station name — was saved into a separate JSON encoded file, following the GeoJSON schema. Cesium has a built in tool to automatically convert GeoJSON files into Cesium geometry.

The city and country names were downloaded in the Python script as well using a reverse geocoding service run by Microsoft’s Bing, which returns location names when given latitude and longitude coordinates. When the search failed to find a name, the weather station’s name provided in the GISS data was used.

4.2 Visualization Tools

The visualization has a collection of tools that allow users to dissect the climate data. Cesium came with many visualization tools needed to interact with the data. Several tools were augmented in style and functionality, while some others were built with third party libraries.

4.2.1 3D Navigation

To navigate the globe, users can zoom and pan a virtual camera, as well as move it to a searched location. The visualization's starting view shows the entire globe, so zooming allows a user to focus on continents, countries, states, and cities. The imagery mapped on the globe is downloaded as tiles, so as the camera is zoomed in the tiles are downloaded and placed over the proper location.

The camera is panned by clicking and dragging on the globe. This changes where on the globe the camera is focusing. The panning orbits the globe, so the zoom level remains constant. This allows users to manually navigate to different regions around the world.

The visualization also provides a search widget to automatically send the camera to a geographic region. A search for any destination has the camera automatically pan and zoom so the whole region is visible. This is useful for the experiment comprehension test, since several questions ask about temperatures for lesser known countries like Gabon.

4.2.2 Timeline

The timeline widget allows a user to browse through historical temperature readings. The time can be played, paused, reversed, or set to specific periods. This updates

the colors of each weather station, which represents the monthly average temperature the station recorded. It also shows and hides weather stations depending on if it has any readings for the selected month and year.

4.2.2.1 Timeline Settings

The timeline by default starts and follows the computer's time. The time can be set by the user to thousands of years in the past or future. Since the weather data only spans a century, the time is clamped between January 1880 and December 2013. The timeline is also set to pause the time once either the start or end time is reached rather than loop back to the other end of the timeline.

The time progression is also sped up 31,557,600 times. This is the number of seconds in a year, so the timeline progresses one year a second. This allows the seasons to smoothly animate — in the northern hemisphere the cold and warm temperatures animate across latitude lines. It also allows many experiment questions to be answered in a timely fashion, since some ask about temperature trends over a decade.

4.2.2.2 Additions and Changes

While the Cesium platform allows many changes to the timeline through an API, additional changes were needed to reduce the possibility for user error during experiments. These include appearance and functionality changes to timeline elements.

The timeline by default shows the entire date details, from the year to the seconds of the day. The temperature data gets only as granular as by month, so the excess data is distracting. Built in methods that convert the date to a string are attached to JavaScript object prototypes, so they were overwritten to only display the abbreviated month and year for the selected time. This change affects data labels in the time animation dial and the timeline tick labels.

The timeline by default also allows users to zoom in and expand the bottom bar ticks for precision timeline control. This functionality was frequently triggered when trying to zoom the virtual camera. It also was not straightforward to restore the timeline to the full century view. A Cesium JavaScript method was overwritten, again attached to the prototype, with a no-operation function to disable this feature.

The time dial had several visual and functional changes done, all with Cascading Style Sheets (CSS). The dial previously had a ring around it that allowed a user to set the magnitude of the time animation. The maximum speed a user can set it to however is only 604,800 times faster, much slower than one year a second. Instead of modifying the Cesium source code to change the time ring's maximum magnitude, the feature was disabled by hiding the ring with CSS. Additionally, a button that changes the time to the computer's time is also hidden with CSS.

The ring previously displayed the animation speed and the full selected date including the time. CSS was again used to hide everything but the month and year. A few other visual changes were made to simplify the time dial's appearance. A before and after comparison is shown in figure 4.2.

Figure 4.2: Cesium's default time dial on the left compared with the modified version on the right. The time ring and the button to reset the time to the computer's clock were removed. The text with the time and animation speed is likewise hidden. Some other visual edits to the play, pause, and rewind buttons were made as well.

During experiment dry runs, several volunteers found setting the time was too difficult when looking at the scrubber position and to the time dial at the side of the screen. To remedy this a tooltip was built that shows the time above the cursor when hovering over the timeline.

The tooltip is a third party tool outside of Cesium. It relies on a jQuery event handler on the timeline HTML element to detect when the mouse is moving over it. Then it uses an algorithm from the Cesium source code to convert the pixel offset on the page into the number of seconds offset from the timeline start date. This is then converted into a Gregorian date, where just the month and year are displayed. The tooltip in action is seen in figure 4.3. Care was taken to ensure the tooltip could be read even when pushed into the corners of the screen.

Figure 4.3: The timeline tooltip, which shows the month and year depending on the cursor’s position. The tooltip only appears when the cursor is hovering over the timeline.

4.2.3 Legend

The legend gives a simple way to see what temperatures the weather station colors correspond to. It is built with D3 and uses the exact same function to convert temperatures to colors that the weather stations use. This guarantees continuity between the legend and the actual data. The legend is positioned in the bottom right corner of the visualization.

4.2.4 Histogram and Station Selector

The station colors give a general sense of the temperatures in a region. For a more quantitative look at station temperatures, users have the histogram widget. It is completely separate from the Cesium library. It is built with D3, a SVG manipulation library, and overlaid onto Cesium using CSS absolute positioning. The background is transparent so that it blends in with the overall visualization.

When the user selects a group of stations, a histogram forms showing the distribution of temperatures. The y-axis is dynamically rescaled depending on the number of weather stations selected. The histogram distribution gives a new perspective into selected temperatures and aids in answering questions about differences in temperature over time.

The temperatures are selected by clicking and dragging over an area with the shift key pressed. This draws a rectangle with corners pinned at the start and end of the drag. While selecting, both the rectangle and the histogram update so the selection has quick feedback. The histogram selects temperatures quickly using a spatial hash data structure, which is detailed in section 4.4.

4.2.4.1 Bug Fix for Station Selector

The selector rectangle is drawn above the Earth's surface so it appears on top of the weather stations. It has to be some distance off of the ground so the resizing weather stations never grow to intersect it. The Cesium API supports adding maximum and minimum zoom limits to the camera, however these limits were not honored when using the search tool. For instance, searching for Iceland in the visualisation would move the camera past an active selector, making it impossible to see when changing the selection area. The code controlling this was not attached to any JavaScript pro-

totypes, so a source code modification was submitted to the Cesium Github project. The fix was merged and released before the experiment phase of the thesis began.

4.3 Build Tools

Several automated tools removed much of the redundant work of installing libraries, packaging the code, and other maintenance tasks.

4.3.1 Webpack

Webpack bundles different modules together to form a cohesive project. Given a set of files to include, the build tool recursively includes every needed component and bundles it into compressed code and stylesheets. Advances in the JavaScript language, brought on by ECMAScript 2015, provide a specification for module loading that automatically prunes unused code. Webpack did not support these new features during the visualization development, so files had to be explicitly included.

The Webpack building scripts allowed the project to be reliably recompiled and deployed on Heroku, an ephemeral server platform. Every time the server is launched, it downloads all needed dependencies, bundles the included code, and serves it to new clients visiting the site. The deployments must therefore be easily reproducible.

Webpack also helped alleviate some memory issues with the visualization. Early in development, the visualization was routinely paused by JavaScript's garbage collector since memory use was high. One reason for this was that all of Cesium's code was loaded into memory, despite only a subset of the modules being used. This would additionally pose a problem during the experiments, since the browser would need to download megabytes of unused code. The download time is already long because of the temperature data, so every megabyte shaved off counts. Webpack's dependency

management reduced the amount of code in memory and was a big first step towards the visualization running without rendering hiccups.

4.3.2 Node Package Manager

The project overall uses 34 different libraries, so downloading and updating them would normally be an ordeal. Node Package Manager (NPM) however provides a way to deal with dependencies in a structured way. A simple `package.json` file contains the list of packages the project uses. This is part of the reason the project can be seamlessly rebuilt on different platforms like Heroku. The file also allows install scripts to be added, which are needed to copy some Cesium static assets into a public directory before the project is loaded.

Every third party library used is maintained through NPM, including Webpack and Cesium itself. It also manages D3, jQuery, moment for formatting times and lodash as a functional tool helper. The only exception is the spatial hash data structure detailed in section 4.4.2, which was not available through NPM. Several source code changes were made to it though, so it could not be used as a third party package in any matter.

4.4 Speed Optimizations

Roughly half of the development time for the visualization was research and implementation of speed optimizations to ensure it could reliably run at 30 frames per second. This became the control latency, which other latencies are later compared to. The first naive implementation, which only showed weather stations changing color with time, ran slower than one frame a second due to constant garbage collection.

The final visualization holds 60 frames per second with no camera movement and

the timeline paused, and dips below 40 frames per second when animating the timeline, using the histogram widget, and moving the camera. Many of the optimizations were found using the Google Chrome profiling tools, which can record performance and memory metrics to illustrate which code segments contribute to computation time.

4.4.1 Memory Management

The largest drain on performance was the garbage collector pausing code execution while it freed unreferenced memory. This came from creating new color objects every time the weather station temperature changed. The Cesium API luckily has a work around. For many methods, including creating a color object, an allocated buffer can be given as an optional final argument. The method writes over the preallocated memory rather than instantiating more memory. Several other methods like the timestamp conversion — converting a Gregorian date to a month and year string — also utilize objects allocated once and reset numerous times.

The D3 color function was also reimplemented for memory saving benefits. The method previously returned a hex string representing the color. Since JavaScript strings are immutable, parsing substrings created three new objects for each red, green, and blue color component. For the thousands of weather stations, memory churn added up quickly. To reduce the number of objects allocated, the D3 interpolation function was lifted from the source code and rewritten to return a simple D3 color object, with fields for each primary color. This required no source code changes in D3, since the interpolation function is an argument to the function that maps temperatures to colors.

As detailed in section 4.3.1, the in code memory was reduced by loading only needed modules in Cesium using Webpack. This freed up heap space, making major

garbage collection events less frequent. The visualization never noticeably stop or stuttered from garbage collection events during the experiments.

4.4.2 Spatial Data Structure

To quickly query weather station locations, their cartographic coordinates were stored in a spatial hash. This allowed culling weather stations outside of the camera's view frustum and quickly selecting temperatures under the histogram selector. The cartographic coordinates were linearly offset so the values range from 0 to 360 for longitude and 0 to 180 for latitude.

A spatial hash is a two dimensional hash map, where items are grouped into different buckets. The hashing function does not aim to reduce bucket collisions however. Items are stored however their position indicates. It is similar to a uniform spatial data structure, except key ranges can easily be queried.

The spatial hash reduces rendering time by speeding weather station culling. Due to Cesium implementation details, the geometry that has color properties changing on demand is re-rendered every frame, regardless of a color change or not. Drawing only visible weather stations gives a modest performance boost, depending on how much geometry is being culled. There are three dimensional spatial hashes available, but two dimensions suffice despite the globe being a three dimensional element. The stations are all on the Earth's surface, so the z axis can be ignored. The latitude and longitude represent the y and x coordinate respectively for each station.

The culling is quickly done by querying the spatial hash using the camera's view frustum height at the earth's surface. In other words, the width and height of the Earth visible to the camera is calculated. This is at most half of the Earth at a given time. The coordinates of the visible region are used to query the spatial hash. A visual example of what is being calculated is shown in figure 4.4.

The frustum height is specifically calculated with the following formula:

$$fh = 2 * height * \tan(fov * \frac{1}{2})$$

where fh is the frustum height, height is the distance the camera is off the ground and fov is the field of view angle of the camera. This is basic trigonometry that, given an angle and height of a triangle, calculates the length of the triangle's base. The frustum width is calculated by multiplying the height by the camera's aspect ratio [23]. The camera height is in meters, so the result is divided by 111,111 to approximately convert it into degrees.

Figure 4.4: Selecting just part of the earth that is visible with the view frustum. The calculated rectangle's coordinates are used to select all weather stations that lie within the highlighted region on Earth.

Since the hash has only approximate weather station locations, stations just outside the view frustum are likely selected as well for drawing. A simple test across

all of the selected stations checks that they are contained in the view frustum. This second filtering step was found to improve rendering performance further.

The two dimensional spatial hash led to a specific problem though. The approach essentially wraps a rectangle around a globe, but the seam where the rectangle ends meet is a special case. This seam for latitude and longitude is called the anti-meridian, located in the Pacific Ocean. It is also the line where timezones shift from positive to negative, heading east.

The culling failed to show weather stations on the other side of the anti-meridian line, depending on which side the camera was centered. To fix this, the culling performs two queries to the spatial hash if it detects the view frustum crosses over the anti-meridian.

Spatial hash speedups are used as well when selecting temperatures for the histogram widget. The rectangle being drawn over the Earth is used to query the spatial hash, and the results are filtered with the same rectangle enclosure check the station culling uses. The anti-meridian handling is different for the drawn rectangle however. The Cesium engine by default wraps geometry on the other side of the world when it crosses the anti-meridian line. To avoid more source code changes, the experiment questions were designed to avoid areas around anti-meridian line, such as Australia.

An open source JavaScript spatial hash was used, albeit with changes to the querying to be compatible with the Cesium rectangle object.

4.4.3 Throttling Graphics Updates

Despite using the spatial hash, geometry updating was too intensive under certain scenarios. Culling updates were only triggered when the camera was moving, but they were happening every frame while the camera was in motion. Histogram updates too slowed the visualization when occurring in rapid succession. For a holistic fix

independent of a camera movement delta or selector size delta, the lodash library's throttle method is used. The helper ensures that a given function is called at most once per interval specified. This allows a simple trade-off between performance and rendering accuracy. In some cases it takes a few hundred millisecond to show new weather stations over a newly visible region, but this is an acceptable sacrifice for consistent performance.

4.4.4 Asynchronous Data Loading

Given the single-threaded nature of a web page, downloading and parsing over 60 megabytes of JSON data halts all rendering for five to ten seconds. Asynchronous tricks and rendering sleight of hand are used to convince the user the page is responding properly. All of the weather data needs to be loaded into memory prior to starting the visualization tasks, so that hundreds of thousands of weather queries a second return quickly.

To download and parse JSON asynchronously, the visualization uses the fetch API to get the data. A polyfill is used to ensure compatibility on older browsers. Once the parsing begins, the page still freezes, since the browser chooses to keep the parsing on a single JavaScript thread. A web worker cannot be used to parse JSON since all communication is done through strings. Any parsing work done would be undone by converting the data to a string.

To mask the main JavaScript thread being frozen, a style sheet animation thread is used instead. CSS3 animation is done on a separate thread from the JavaScript, so while the main thread is hung with processing, a simple loading animation is shown to the user that never halts. All visualization input is disabled while loading, so the behavior appears controlled.

4.4.5 Cesium API Settings

Despite many optimizations outside of the Cesium framework, several performance gains came from fine-tuning Cesium settings.

A small but useful way to reduce memory usage was setting a flag to only store the three dimensional geometry for the scene. Cesium by default supports animating the globe into a flat map, and likewise morphs the geometry into a two-dimensional representation. This functionality is disabled in the weather station visualization, so a good portion of memory is saved.

Another performance saving technique is batch processing the weather stations. The Cesium geometry entities are stored in Entity Collections with callbacks fired as items are added or removed. The callbacks can be paused and resumed before modifying many stations at once. This makes updating the histogram happen once the set of weather stations is finalized, rather than every time weather stations are added or removed from the selected set.

Chapter 5

LATENCY EXPERIMENTS

To test how latency affects user performance with 3D interactive data visualizations, a sample of 30 people used the climate visualization to complete a comprehension test. This was a single blind study — the visualization’s rendering speed was secretly changed between questions by the experimenter. The experiments recorded the time each question took to answer, the test score, and the gaze location. All of the experiments were conducted on the Cal Poly campus with registered students. This experiment was approved by the Cal Poly Human Subjects Committee.

5.1 Latency Conditions

The latency levels are picked so that they will capture when user performance degrades. The past study into data visualization latency used a half second delay, which is two frames per second [12]. In figure 2.1, this is at the flat part of the curve, where large latency changes lead to small changes in frame rate. The point where user performance drops off is likely at a higher frame rate. The frame rates forming the curve’s bend in figure 2.1 are therefore selected: 30, 20, 10, and 5 frames per second.

5.2 Hypothesis

User performance should measurably degrade at 5 frames per second. This should be seen in test scores, the time to complete a test, and it should affect gaze patterns to discourage exploration and movement in general. This prediction is based on previous studies, that found decreased tool usage as soon as 200 millisecond latency (equivalent to 5 frames per second) [2] [17] [12].

5.3 Equipment

The experiment used two laptops, a gaze tracker, and any available set of monitor, mouse, and keyboard peripherals. Several programs were simultaneously running on the different computers, making sure that gaze and question timing data was captured.

5.3.1 Hardware

Both a Mac and Windows machine were used during the experiments. Hardware specifications are in table 5.1. The Mac was used to operate an experiment control panel, discussed in section 5.4.2. The Windows computer powered the data visualization, the comprehension test, gaze capture software, and logging of question completing timing. A dual-monitor setup had the data capture programs on the laptop screen, turned away from the user, while an external monitor showed the visualization and the test questions.

Table 5.1: The hardware specifications for the two computers used in the study. Only the Windows computer actually ran the visualization during the experiments.

Computer	2013 13-inch MacBook Air	HP Pavilion dm4
Operating System	OS X El Capitan	Windows 7
CPU	1.7 GHz Intel Core i7	2.8 GHz Intel Core i7
GPU	Intel HD Graphics 5000	ATI Mobility Radeon HD 5470

The monitor, mouse, and keyboard used were all from on campus workstations. These computer peripherals were connected to the Windows laptop, allowing the user to focus on the experiment while the gaze capture quality was monitored.

Gaze positions were captured with the MiraMetrix S2, a high precision gaze

tracker. Readings are accurate within one degree of the visual angle and approximately 60 recordings are taken every second [15].

A typical hardware setup used in experiments is shown in figure 5.1.

Figure 5.1: A typical experiment setup. The photo on the left shows the subject’s view before the experiment. The mouse, keyboard, gaze tracker, and monitor are connected to the black Windows laptop. The right image shows the experimenter’s view. The Mac has the experiment control panel while the Windows PC is running the Mirametrix gaze tracking software.

5.3.2 Software

The gaze tracker comes with two pieces of software: a server and a client. The client receives the gaze data from the server and saves it as an XML encoded file. The client also streams the gaze tracker output from the device, giving feedback if a subject’s eyes are still in frame. These programs were only compatible with the Windows computer.

A custom python server program ran on the Windows machine as well, which received TCP packets from the visualization server. The data included a timestamp when the data was sent and the frames per second the visualization was running. The program logged this information into a text file along with the output of the Windows `QueryPerformanceCounter`, which is a tool to get high resolution timestamps. The gaze log included the `QueryPerformanceCounter` output as well; thus, the counter is used to correlate the two data streams. The details of merging together data logs are

discussed in section 6.1.

The python server was able to accept signals from a remote server because of ngrok; a tool which exposes a local port to the internet. The exact external port number changes every time the program is launched, so the port number was set in the experiment console before each trial. Version details for all of the software used is presented in table 5.2.

The visualization ran in a Google Chrome window. The comprehension test was created and administered using Google Forms. It ran in a separate Google Chrome window, filling the other half of the screen.

Table 5.2: The software versions used in the experiment to test users and capture metrics for later analysis.

Software	Version
Google Chrome	49.0.2623
Cesium	1.19.0
Miramatrix Server	2.5.2.152
Miramatrix Client	2.5.2.152
Miramatrix API	1.1
ngrok	2.0.25
Python	2.7.11

5.4 Visualization Modifications

The largest change the visualization underwent was enabling the frame rate to be set remotely. The visualization was transformed from a client only program (only needing an HTTP server to send the files) into more of a traditional client server architecture, where the client and server software regularly communicated frame rate changes. A server was also added since the virtual camera position needed to be

logged so it could later be interpolated with the gaze data. Web browsers do not provide convenient ways to create local log files, so the data was sent to the server for recording.

None of the participants suspected that the visualization’s latency was being artificially changed, ensuring that the study remained a blind study.

5.4.1 Toggling Animation Latency

Fortunately, the Cesium API supports dynamically changing the engine’s frame rate. It’s a simple API call that instantly responds when the frame rate cap is set. The server sent frame rate changes to the client via WebSocket.

The main challenge however was changing the histogram animation speed in accordance with the frame rate. The histogram is a D3 component, so it is not under the Cesium API’s control. The D3 animations use interpolation functions to calculate animation patterns. A custom interpolation function was written to make animations progress in a stepwise fashion. If the animation time change was smaller than $\frac{1}{FPS}$, where FPS is frames per second, then the previous value was returned.

The visualization was also modified to display a loading wheel until an initial frame rate message was received. This ensured that participants did not begin the experiment until everything was properly configured.

5.4.2 Experiment Control Panel

The control panel is a password protected web page that orchestrates server communication to the client. The panel uses a REST interface to communicate with the server.

The panel is opened on the Mac machine for the experiments, where data like

the camera log name, the ngrok port number, the frame rate, and an end signal are controlled. The end signal is used to mark in the logs when the experiment is over. The panel also contains detailed step by step instructions to make sure each experiment participant gets the same tutorial and that all data is being captured properly.

5.4.3 Server

The client and control panel communicated with a Node.js Express server. It maintained a WebSocket connection with the visualization client to receive virtual camera positions and sent the current latency. Whenever a new frame rate was submitted, the server sent the value to the client through the WebSocket. It also logged it into the virtual camera log and sent it as a TCP packet to the Python server running on the Windows machine.

The server saved logs both as local files and into a data logging service called Papertrail. Heroku runs ephemeral servers, so if one of them crashed a completely new instance would replace it with no log files. The Papertrial backup was used to recover data for one trial.

Only one visualization client was allowed to open a WebSocket connection to simplify server logic. The first client to connect has the server's control until disconnect. Any other clients trying to connect cannot read the frame rate data. This prevents experiment errors like opening two clients accidentally. The second client would not even finish loading since it needs an initial frame rate to unlock the visualization.

The server also gave some feedback to the experiment control panel when data was submitted. When a frame rate was sent for example, the server returned an error if there were no active web socket connections. This functionality ensured that signals were resent if the client temporarily lost the WebSocket connection.

5.5 Procedure

Subjects met individually to answer 12 multiple choice questions using the climate visualization. They were told they were evaluating the effectiveness of data visualization tools. The experiments took between 20 and 30 minutes on average. Each volunteer first calibrated the gaze tracker by following a moving target with their eyes. The experiment proceeded when the calibration error score was less than 80 pixels of error, specified by the Mirametrix user guide [15].

Users were then given a short tutorial on how to use the visualization’s tools, including the search, timeline, camera, and histogram. They then began the test, with the option to ask for reminders on how to use different visualization features.

While participants were answering the questions, the gaze tracker’s video feed was monitored to make sure that the user’s eyes were being captured properly. In the case someone moved their head too much they were instructed to recenter. Simultaneously, the control panel on the second laptop sent frame rate messages whenever a question was answered. The latency only changed every third question, but to record question completion timing the same latency was often resent.

After a user finished the questions, they also answered whether they noticed the visualization’s speed change during the experiment. They then were then debriefed about the true purpose of the experiment. Early on in the experiment they were also asked to refer new participants to the study with while keeping the experiment’s true intent a secret.

The question order and latency order were randomized for every experiment. The question randomization was a stratified random sample however. The test has four types of questions that are repeated three times each. They were therefore selected so that each type appears once per latency level. This helped ensure that visualization

tools are evenly requested between different frame rates. The multiple choice options for each question also appeared in random orders for each participant to reduce possible bias in the option ordering.

5.6 Sample Pool

Statistics majors were originally tapped to volunteer for the study. The experiment was subsequently opened up to all Cal Poly students. Volunteers were sourced from the Mechanical Engineering department, the Computer Science department, and many first and second degree direct invitations. The sample quality therefore suffers to some degree from convenience sampling. No one was turned down from the study to minimize selection bias.

5.7 Experiment Variables

The independent variable in the experiment is the latency the visualization runs at. This is toggled throughout each experiment trial.

The dependent variables are the gaze patterns, question scores, and time answering each question. The gaze patterns include several facets like the fixation time and the density of gazes.

Controlled variables are the equipment each person used and the tutorial given at the start of the experiment.

Uncontrolled variables include the exact location the experiment took place, the internet speed, and the noise level in the room. Every experiment took place on campus and most were conducted in the Cal Poly Computer Science Lab. Some experiments however were conducted in the Human Computer Interaction Lab since volunteers were sourced from a nearby Computer Science class. In the Computer

Science Lab, whichever station that was available was used, so the exact location in the room varied from person to person.

Larger issues come from the internet speed and sound level. The internet speed became noticeably slow during some experiments. This caused the frame rate signals from the server to be delayed and the global tile imagery loading to slow down noticeably at times. In addition, the sound level sometimes went above what is normally expected in a test taking environment.

5.8 Experiment Errors

Of 31 attempted trials, only 30 were completed and 28 from those were selected. One participant had to abandon the experiment halfway through due to a scheduling conflict. For another trial, the gaze recording was started late, after the experiment began. Making matters worse, the user encountered a bug while answering the final question, causing the visualization to crash. The bug was fixed for future participants. The second discarded trial was deemed unsuitable since the gaze tracker's infrared lights had trouble staying on, rendering the gaze data incomplete. The gaze tracker was swapped for a working backup model for the next participants.

In the trials kept, some of the data is imprecise due to experimental error. There were a handful of times that the frame rate signal was sent either too early or too late, which changes how gazes are grouped by question and the timing between questions.

Additionally, for one trial the virtual camera log file was deleted from the Heroku server before it could be downloaded. The backup log on Papertrail was retrieved, however the timestamps did not have millisecond precision. Later merging with the precise gaze measurements is therefore not as accurate as with other trials.

The unreliable internet speeds also had an impact on a survey question. When

questioned if there were noticeable latency changes in the visualization, some users asked if this includes the slow tile loading. It is safe to assume that other people could not distinguish the difference between internet speed and rendering performance when answering this question.

Chapter 6

ANALYSIS

The multiple choice tests and the gaze data sets were both analyzed to see if latency had any affect on cognition and performance while using the visualization. The comprehension test analysis considers the time it took to answer questions and the resulting test score. The gaze analysis is split into a heat map visual comparison and a statistical analysis of fixations times. The fixation analysis is inspired by a prior gaze tracking study [8].

The comprehension test analysis fails to find statistically significant differences between the response time or resulting score based on latency. The gaze analysis, too, fails to show consistent patterns in the way user navigation is impacted. However, the fixation time analysis does show that in the case of 2D widgets the gaze fixation time is in fact being affected by latency.

6.1 Data Parsing

The three separate logs generated by each experiment had to be merged together to create a new set of aggregate logs. This included the gaze logs, virtual camera logs, and the test result logs. Python scripts merged and processed these logs into comma separated value (CSV) files. These files were then given as input to the R script for analysis.

The main challenge in preparing the data for analysis was synchronizing the virtual camera with the gaze logs, since the gaze log generated by the Mirametrix gaze tracker lacked an absolute timestamp. The gaze log did however have the Windows QueryPerformanceCounter value. The local Python server logged the incoming frame

rate and timestamp along with the `QueryPerformanceCounter` value, making it possible to correlate the two logs.

Parsing question scores also posed a challenge. The responses were recorded in a Google Forms document with a randomized question order. The randomization order had to be recorded since the Google Forms interface does not provide a way to track the actual question order. The final CSV file contains records for all users indicating whether they answered the question correctly, what latency they experienced while answering the question, and the time it took to answer the question.

6.2 Question Analysis

This analysis considered the test answers given during the experiments. The goal was to determine if latency had any effect on test scores or question response times.

During the test, participants were asked whether they noticed any latency during the experiment. Approximately 43% of the users indicated that they did notice latency. The number may well be lower though. As detailed in section 5.8, the internet speed variability occasionally caused slow tile loading times for the the globe, which could have been confused with the artificial induced latency.

Figure 6.1: The spread of question completion durations before and after the outlier was filtered out.

For the question completion duration analysis, one outlier was filtered out. One user took over 15 minutes to answer a single question, over double the next highest time of six minutes. Figure 6.1 shows the question completion duration for each latency as a box plot, before and after filtering the outlier out from the data set. The outlier’s magnitude is definitely an anomaly.

6.2.1 Duration of Question Completion Analysis

An Analysis of Variance (ANOVA) test provides insufficient evidence that latency affects question completion duration. This means that users answered questions in approximately the same amount of time, regardless of whether the visualization was running at 30, 20, 10, or 5 frames per second. The ANOVA test analyzed latency’s effect after adjusting for the specific questions and different users.

The data was originally skewed right, since there were several outliers who took a longer time to answer questions. A logarithmic transformation made the distribution roughly symmetric, allowing the ANOVA analysis to proceed. The results of the transform are plotted in figure 6.2.

Figure 6.2: Question completion durations before and after logarithmic transform to make it symmetrical, a prerequisite for ANOVA p-values to be accurate.

The question duration does have noticeable differences in answering time, namely that questions at 10 and 5 frames a second took eight seconds shorter than questions at 30 and 20 frames per second. This is masked however by standard deviations ranging from 38 to 53 seconds between the latency levels. The full data is shown in table 6.1.

Table 6.1: Summary statistics for the question duration by latency level.

Frames Per Second	Mean (in seconds)	Standard Deviation (in seconds)
30	100.19	53.07
20	100.15	50.42
10	92.25	38.22
5	92.43	47.82

6.2.2 Comprehension Test Score Analysis

A logistic regression analysis found that, adjusting for the question and subject, there is poor evidence that test scores are affected by the visualization's latency. The test failed to conclude that latency has a significant effect on how well people performed on the comprehension exam. Figure 6.3 gives an illustration of the logistic regression analysis.

This result may be surprising, but the spread of the test scores is to blame. Mean scores are 10% lower for the slowest latency compared to the control. However, the standard deviation for the tests at each latency level is over 45%. Despite the difference in the average, the high variance makes it difficult to separate out test scores which may have been affected by latency.

Figure 6.3: A logistic regression plot, along with an integrated histogram. The line estimates the test score based on the frames per second. While the score slightly increases as the latency is reduced, this slope is too minor to be significant. The histogram bars give a better sense of data quantities than a traditional logistic regression plot [20].

6.3 Gaze Analysis

Both the raw gaze points and fixations were analyzed for trends in response to latency. The gaze points were combined with the virtual camera position to extrapolate positions on the globe where the user was looking at. The extrapolation worked by recreating the camera’s view frustum on the Earth’s surface, much like how it was calculated for the station geometry culling in section 4.4.2. Since some gazes were

over two-dimensional elements like the histogram or timeline tool, these gazes were filtered out before extrapolating the cartographic gaze coordinates. Another analysis was done for eye tracking on the two-dimensional elements of the climate visualization.

6.3.1 Heat Map

The heat map images vary from blue to red, where red represents the highest density of gazes. This allows just the most focused on areas to be analyzed, and different focal areas to be compared for trends. Due to experimental error, described in section 5.8, some of the question cutoffs are inexact. This results in gaze data from different questions occasionally being combined. A statistical method to compare two heat maps is beyond the scope of this thesis, so the images are subjectively compared for trends.

The heat maps are generated with an R script, which groups data points based on frequency in a region. The *stat_density2d* function is used to create the heat map with 32 bins set.

The twelve questions can be divided into three question groups that focus on different tool sets. The first four questions analyze weather station quantities, questions five through nine ask about temperature changes over time, and the last four questions ask about weather incidents in specific years. These question groups require different focus on interaction tools, which may affect gaze patterns as latency changes.

The first group relies on the timeline and spatial search. The second group needs the histogram and the legend to effectively answer questions, although some users neglected these extra tools. The third group requires extensive use of the search tool since it often compares separate regional groups. The expectation is that more complex tools like the histogram will decrease in use when the latency is high, given

the past work that showed that brushing and linking is very sensitive to delays [12].

The gaze maps were also normalized for each user. This overcomes a known limitation with heat maps. A hot spot can either represent many users briefly looking at an area or few users focusing their attention. To reduce the influence of any one particular experiment, the gaze frequencies were normalized for every question so each user contributed an equal number of randomly sampled gazes.

6.3.1.1 Heat Map Over The Globe

Figures 6.6 to 6.8 collectively show all of the gaze data on the globe. Each figure displays a separate group of questions. The expectation is that gaze patterns should be consistent within each figure, and different between the figures if latency had any effect. Each group was analyzed for an overall trend between high and low latency, to see if latency affected gaze patterns.

Analysis shows that none of the question types follow a clear pattern with regard to how latency affects gazes. In figure 6.6 for instance, half of the questions show gazes becoming more narrow, while the other two show gazes becoming more broad as latency increases.

Even subjectively grouping questions by gaze focusing or gaze broadening fails to reveal any patterns among the questions. A general trend of more narrow gazes is seen in questions 1, 2, 7, and 8 while gazes cover more area under questions 3, 4, 5, 9. The other four questions do not have a clear pattern. Each group of questions has different focus on tools, making the increasing and decreasing density patterns coincidental between these questions. The analysis is therefore inconclusive.

An additional heat map analysis selected only gazes contributed by users who answered a given question correctly. This removed about a third of the gazes overall, but the heat maps retained their basic shapes and still lacked patterns in response to

latency changes.

6.3.1.2 Heat map Over 2D Widgets

For gazes just over the widgets, figs. 6.9 to 6.11 fail to show any consistent pattern as well. The widgets that had been made available to the users were the time dial, timeline, and legend at the bottom, along with the histogram and search at the top. Similar to the cartographic gazes, there is no subjective pattern between latency levels per question type. While some questions demonstrate declining usage of the histogram, like questions 2, 6, and 10, others show the histogram increase in use as latency increases.

6.3.2 Fixations

The fixation analysis aggregates gazes into fixation points, each with a location and duration. The fixations are mapped as circles, where larger circles represent a longer fixation duration. Gaze aggregation was computed by the gaze tracker.

The tests generated thousands of fixations, which unfortunately makes the maps difficult to read. For the volume of data collected a heat map is one of the only effective ways to see both the map background and the relevant hot spots. Fixation duration lengths can be statistically analysed though, which gives useful insights.

Figures 6.12 to 6.17 show a similar lack of pattern as the gaze maps. The fixation map is in many ways similar to the heat map. Dark regions on the figures are caused by concentrations of many translucent fixation representations.

A statistical analysis of the fixation durations reveals interesting facts however. An ANOVA test of the logarithm of the duration time of fixations, adjusted for the question and the specific subject, gives strong evidence that latency affects fixation

times. Fixation time does not impact the time completing the comprehension exam as section 6.2.1 shows.

Separating fixation data by 2D and 3D elements shows that only 2D fixation durations are significantly affected by latency.

The fixation maps, unlike the heat maps, are not normalized per user. This is because the fixation aggregation limits how much any one user can mask other participants. This normalizes the gazes already.

6.3.2.1 Globe Fixations

For the subset ANOVA test, there is insufficient evidence to say that latency has an effect on fixation duration. With a p-value of 0.0616, this is above the alpha level of 0.05 set as a criterion for a significant result.

Fixation maps over the globe fail to show any pattern as well. As with the heat map, while some individual questions exhibit a fixation pattern, none of the groups of questions have any consistent patterns. The patterns are very similar to the heat map, since it is the same underlying data.

6.3.2.2 Widget Fixations

Tests of the 2D fixation data show strong evidence that the fixation durations are affected by latency, with a p-value of 0.022. A post hoc Tukey comparison test reveals exactly which latency levels differ from one another, illustrated in figure 6.4.

The figure shows a statistically significant difference between 20 and 10 frames per second, along with 20 and 5 frames per second.

The median of the fixation duration during 10 frames per second is 1.023 times longer than fixations at 20 frames per second (95% CI: 1.015 to 1.031 times higher).

Likewise for the second significant result, the median of the fixation duration during 5 frames per second is 1.021 times longer than fixations at 20 frames per second (95% CI: 1.012 to 1.029 times higher). Converting the Tukey differences from logarithms gives the geometric mean, but since the logarithmic fixation data is roughly symmetrical, the geometric mean is a good approximation for the median of the original data set. Figure 6.5 shows that the logarithmic transformation makes the fixation duration distribution roughly symmetrical.

This result is interesting since none of these tests include the control of 30 frames per second. Also, the confidence interval between 20 and 10 frames per second is higher than the 20 and 5 frame rate comparison. The interval magnitudes are only slightly different though, so the impacts from 20 frames to either 10 or 5 frames per second are very similar. The 30 frames per second level did not show a significant difference with any other level, despite it nominally being the control. This is likely because 30 frames per second appeared to be too fast for many users, forcing fixation times to increase as users were trying to catch up. Previous latency work alludes to how a visualization should operate at the speed a user is thinking [12]. Several users during the experiment commented how using the visualization at 20 frames per second was easier because it was slower. In retrospect setting the visualization timeline to move one year a second was too quick — the 20 frames per second slowed the visualization down to the pace most users’ were comfortable with.

One possible flaw with this analysis is that the second round of ANOVA tests are post hoc tests. Repeated post hoc tests typically control for the family error rate using a p-value correction. The p-values are reported here for the reader to decide if they indicate statistical significance. The widget fixation ANOVA test is still significant with a Bonferroni correction however, since $0.022 < \frac{0.05}{2} = 0.025$. Note however that setting the Tukey family-wise error rate to 0.975 makes only the 20 to 10 frames per second fixation difference significant.

Figure 6.4: A comparison of every latency level. Only items with a confidence interval all positive or negative are significant results. Here, 5-20 and 10-20 are significant. The Tukey test makes sure that the overall significance level is 95%, making each individual confidence interval wider to compensate.

Figure 6.5: Histograms showing that the logarithmically transformed fixation duration data is roughly symmetrical. The geometric mean is a good approximation of the median because of this.

Figure 6.6: Global heat map for questions about weather stations appearing and disappearing over time.

Figure 6.7: Global heat map for questions about temperature changes over a decade.

Figure 6.8: Global fixation map for questions about temperatures at specific dates.

Figure 6.9: Visualization widget heat map for questions about weather stations appearing and disappearing over time.

Figure 6.10: Visualization widget heat map for questions about decade temperature changes.

Figure 6.11: Visualization widget heat map for questions about temperatures at specific dates.

Figure 6.12: Global fixation map for questions about weather stations appearing and disappearing over time.

Figure 6.13: Global fixation map for questions about temperature changes over a decade.

Figure 6.14: Global fixation map for questions about temperatures at specific dates.

Figure 6.15: Visualization widget fixation map for questions about weather stations appearing and disappearing over time.

Figure 6.16: Visualization widget fixation map for questions about temperature changes over a decade.

(a) Question 9

(b) Question 10

(c) Question 11

(d) Question 12

Figure 6.17: Visualization widget fixation map for questions about temperatures at specific dates.

Chapter 7

CONCLUSION AND FUTURE WORK

The experiment failed to find that latency leads to sufficient loss of user effectiveness. This may be largely due to the latency level choices. The previous data visualization latency experiment, which found positive results, slowed operations by half a second [12], whereas this study induced a maximum of 200 milliseconds of lag. The point where the visualization's performance begins to affect a user is likely higher than 200 milliseconds latency. Future work should study higher latency levels.

Despite the lower limit, latency's effects were present in gaze fixation durations, although just with the two-dimensional elements on the screen. This shows evidence that 2D information visualizations are more susceptible to latency than 3D spatial visualizations.

The sample pool of users should be more homogeneous in a future study. Users who had experience with video games or had a solid grasp of geography had a testing advantage in this study. Selecting a more consistent group could reduce variability between test scores and reveal some significant comprehension differences which can be attributed to latency.

Another way to reduce differences between participants is to randomly generate the geography that the weather stations are mapped onto. This would put each user on an even playing field so geographic knowledge could not be used to quickly find locations. The visualization's interaction tools must be used to discover insights into the data.

The work here is focused on performance degradation; it does not consider user preference. If the study were redone then users would be asked which question was

their favorite at the end of the comprehension exam. The answers would be cross-referenced with the latency the question was answered under to see if users indicate a preference for lower latency environments. It is plausible that subjects dislike any tools with higher latency, spatial or informational, and will be less likely to use products they perceive to be slow or unresponsive.

BIBLIOGRAPHY

- [1] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. The effects of loss and latency on user performance in unreal tournament 2003®. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '04, pages 144–151, New York, NY, USA, 2004. ACM.
- [2] J. Brutlag. Speed matters for google web search. Technical report, Google, Inc., jun 2009. services.google.com/fh/files/blogs/google_delayexp.pdf.
- [3] J. Brutlag, H. Hutchinson, and M. Stone. User preference and search engine latency. In *JSM Proceedings, Quality and Productivity Research Section.*, Alexandria, VA, 2008.
- [4] D. Chavez. Is 3D visualization the next step for big data? Technical report, Wired, 2012. www.wired.com/insights/2012/11/3d-visualization-big-data.
- [5] H. Chennamma and X. Yuan. A survey on eye-gaze tracking techniques. *Indian Journal of Computer Science and Engineering*, 4:388–393, 2013.
- [6] M. Friendly and D. J. Denis. Milestones in the history of thematic cartography, statistical graphics, and data visualization. <http://www.datavis.ca/milestones>, 2008.
- [7] H. Fuchs, M. Levo, and S. M. Pizer. Interactive visualization of 3d medical data. Technical report, Computer Science Department University of North Carolina, 1989.
- [8] N. A. Giovinco, S. M. Sutton, J. D. Miller, T. M. Rankin, G. W. Gonzalez, B. Najafi, and D. G. Armstrong. A passing glance? differences in eye track-

- ing and gaze patterns between trainees and experts reading plain film bunion radiographs. *The Journal of Foot and Ankle Surgery*, 54(3):382–391, 2015.
- [9] J. Heer and B. Shneiderman. Interactive dynamics for visual analysis. *Commun. ACM*, 55(4):45–54, apr 2012.
- [10] M. Kramer and R. Gutbell. A case study on 3d geospatial applications in the web using state-of-the-art webgl frameworks. In *Proceedings of the 20th International Conference on 3D Web Technology*, Web3D ’15, pages 189–197, New York, NY, USA, 2015. ACM.
- [11] P. Lewis. Rendering performance. Technical report, Google, Inc., 2016. developers.google.com/web/fundamentals/performance/rendering/.
- [12] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2014.
- [13] I. S. MacKenzie. Fitts’ law as a research and design tool in human-computer interaction. *Hum.-Comput. Interact.*, 7(1):91–139, Mar. 1992.
- [14] D. MacLeod. The cnn ecosphere project. *The Inspiration Room*, dec 2012. theinspirationroom.com/daily/2012/the-cnn-ecosphere-project.
- [15] Mirametrix Inc., 200 - 245, Victoria Ave. Westmount, Quebec. *Mirametrix S2 Eye Tracker User Guide*, 2.5.0.152 edition, aug 2013.
- [16] P. Read and M.-P. Meyer. *Restoration of Motion Picture Film*. Conservation and Museology. Butterworth-Heinemann, aug 2000.
- [17] E. Schurman and J. Brutlag. The user and business impact of server delays, additional bytes, and http chunking in web search. Technical report, O’Reilly Radar, 2009. radar.oreilly.com/2009/06/bing-and-google-agree-slow-pag.html.

- [18] A. Sears, J. A. Jacko, and M. S. Borella. Internet delay effects: How users perceive quality, organization, and ease of use of information. In *CHI '97 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '97, pages 353–354, New York, NY, USA, 1997. ACM.
- [19] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu. The effect of latency on user performance in warcraft III. 2003.
- [20] J. Smart, W. J. Sutherland, A. R. Watkinson, and J. A. Gill. A new means of presenting the results of logistic regression. *Bulletin of the Ecological Society of America*, 85(3):100–102, 2004.
- [21] J. B. Smith and G. J. Pitts. Regional climate change scenarios for vulnerability and adaptation assessments. *Climatic Change*, 36(1):3–21, 1997.
- [22] R. J. Teather, A. Pavlovych, W. Stuerzlinger, and I. S. MacKenzie. Effects of tracking technology, latency, and spatial jitter on object movement. In *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*, pages 43–50, March 2009.
- [23] Unity Technologies. *The Size of the Frustum at a Given Distance from the Camera*, 5.3-M edition.

APPENDICES

APPENDIX A

COMPREHENSION TEST QUESTIONS

1. In the 1960s, which country in Western Asia added the most weather stations?
 - (a) Egypt
 - (b) Turkey
 - (c) Israel
 - (d) Lebanon
2. Around 1990, which Eastern Asian country lost the most weather stations?
 - (a) Japan
 - (b) South Korea
 - (c) China
 - (d) Mongolia
3. When did Canada lose a large amount of weather stations?
 - (a) 1980 - 1981
 - (b) 1893 - 1895
 - (c) 1989 - 1991
 - (d) 2009 - 2012
4. Which European country added the most weather stations around 1951?
 - (a) Poland

- (b) France
 - (c) Germany
 - (d) Spain
5. In the 1970s, which country had the most stable temperatures?
- (a) The Philippines
 - (b) Japan
 - (c) South Korea
 - (d) Thailand
6. What is the highest temperature range that Iceland experienced in the 1920s?
- (a) -10 to 0
 - (b) 0 to 10
 - (c) 10 to 20
 - (d) 20 to 30
7. Which country in the southern hemisphere had no weather stations record below freezing from 1985 to 1995?
- (a) Chile
 - (b) Argentina
 - (c) Uruguay
 - (d) Antarctica
8. From year 2000 to 2010, which North African country registered the coldest weather stations?
- (a) Algeria

- (b) Libya
 - (c) Egypt
 - (d) Morocco
9. Around January 2011, which single state in the US had no snow?
- (a) Florida
 - (b) California
 - (c) Texas
 - (d) Arizona
10. Which was the coldest year in Colorado?
- (a) 1956
 - (b) 2013
 - (c) 1890
 - (d) 1912
11. Which year had the highest temperatures in the United Kingdom?
- (a) 1900
 - (b) 1911
 - (c) 1921
 - (d) 1931
12. Around June 1960, which African country was hotter?
- (a) Somalia
 - (b) Kenya
 - (c) Ethiopia

(d) Gabon