FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

HistoryTime

A Chrome history visualization using WebGL

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Roman Püngüntzky
Registration Number 1125593

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Dipl.-Ing. Dr. Ivan Viola
Assistance: Dr. Manuela Waldner M.Sc.

Vienna, 30th October, 2014

Roman Püngüntzky | Ivan Viola

Erklärung zur Verfassung der Arbeit

Roman Püngüntzky
Franz Liszt Gasse 5, 2353 Guntramsdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. Oktober 2014

Roman Püngüntzky

Acknowledgements

I would like to thank my supervisor, Manuela Waldner, for accepting me as one of her bachelor students, for her patience and her time, as well as her generous support throughout the project.

I thank my parents for their constant support and care.

I also thank the participants of the user evaluation for their time, cooperation and honest feedback.

The following people must also be mentioned for their help and inspiration: Michael, Andreas, Alexandra and David.

This thesis was conducted in the course of the Vienna Science and Technology Fund (WWTF) project "Visual Computing: Illustrative Visualization" (VRG11-010).

Abstract

Even though modern web browsers offer history functionalities, only few people use it to re-visit previously visited websites. In this thesis we present HistoryTime, a 3D visualization of the Google Chrome browser history. The goal of this project was to visualize the content of a user's web browsing history in an aesthetic way, as well as to increase the general motivation to use it. We developed a 3-dimensional, visually appealing extension for Google Chrome that offers various possibilities, sorting-modes and browsing-functionalities which should make exploring and searching for websites in the history more pleasant to use. The data is retrieved via the Chrome history API exclusively, and visualized in a WebGL environment using the three.js JavaScript 3D library.

The prototype of HistoryTime was tested and compared to the standard Google Chrome browser history in the scope of a small user study. The results indicated that our extension offered a better usability overall, and also allowed to solve certain tasks that were not possible with the standard history.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

While most modern web browsers offer a history functionality, only few people use it to revisit web pages that were previously visited. A study by Tauscher and Greenberg [25] indicated that revisiting web pages made up for 58% of all browsing done on the Internet. However, some browser features that are intended to aid revisitation are rarely used. The browser history, for example, is only used to initiate 0.2% of all web page requests, as Weinreich et al. [27] found in their study.

One reason for this is that a browser's history function is kind of a "hidden" feature that is not as prominent as the back button or the address bar [28]. According to past work [27, 11], many users were not aware that a browsing history even existed. Another reason could be that, in most web browsers, the browsing history is represented as a textual list of visited web pages, sorted by date or popularity. The human visual system can only perceive textual input sequentially. However, graphical information, such as pictures, videos or charts, can be perceived in parallel [19]. A logical step to make browser history data more accessible and usable is to represent them visually. In their paper, Nadeem and Killam further showed that visual aids in history mechanisms are far more effective than the use of just textual data [22]. Additionally, the Google Chrome standard history does not provide sufficient query possibilities for an interactive exploration, such as time-based queries, or the possibility to sort the elements in a different manner.

In this thesis we present the concept, implementation and evaluation of HistoryTime, a 3D history visualization whose goal is to improve the aesthetics, usability and utility of the history data in Google Chrome. We wanted to motivate people to explore their history by developing an extension for Google's browser that was visually appealing, and replaces its standard history. Through multiple prototypes and a small user study, we found a combination of ideas and concepts to support the exploration of a user's browser history, while extending the rudimentary amount of functionality for visualizing the data.

The structure of this thesis is as follows: related systems that have been developed in the field of browsing history visualization will be introduced in Chapter 2. The design principles and methods of HistoryTime are discussed in Chapter 3. Chapter 4

covers the the most important parts of the implementation of our developed system in detail. An outline of the user study designed to test the usability, aesthetics and efficiency of HistoryTime compared to the standard Google Chrome history, as well as the corresponding data analysis and discussion of results is presented in Chapter 5. Finally, our findings and conclusions are summarized and possible future work is discussed in Chapter 6.

CHAPTER 2

Related Work

Over the last years, researchers have tried various forms of visualizations to simplify searching within a browsing history. For the related work that will be introduced in this chapter, we will distinguish between different representations (e.g. graph-based, timeline-based and others) and dimensions (2D, 3D). Table 2.1 shows a comparison of all mentioned systems:

System	Representation	Dimensionality
PadPrints [20]	graph	2D
MosaicG [12]	graph	2D
Trails [30]	graph	2D
Webpath [18]	graph	3D
VISVIP [17]	graph	3D
WebBook [15]	other	3D
BrowseLine [21]	timeline	2D
WebComets [16]	timeline	2D
HCB [24]	timeline	2D
Yamaguchi [29]	other	3D
HistoryTime	**timeline**	**3D**

Table 2.1: Comparison of the presented related work.

PadPrints [20] and ***MosaicG*** [12] are similar 2D browser companions that were proposed to aid web navigation. They show a graph-based hierarchical history of web pages visited during a browsing session. When a users accesses a page from the browser, that page is added to the display within the web browser window. In both systems, a node in the hierarchy displays the title and thumbnail of a web page. The hierarchy is constructed as users traverse links from one page to another. In contrast, HistoryTime uses a date-domain hierarchy where visited websites are pooled by their domain and the

day they were visited. We do not update this hierarchy as soon as the user continues to browse the web, unless the extension is reloaded. Both the PadPrints and MosaicG visualizations adapt to changes in realtime. The ***Trails*** [30] system is an interactive web history visualization and tagging tool that was built as an aesthetic, functional Mozilla Firefox add-on. The authors showed that, through the aid of visual elements, users were able to retrace their life on the Internet and understand their habits better, compared to the standard Firefox history. Also, they wanted to make the searching and organizing of the web content faster and more fun, which was one of the main goals of HistoryTime as well.

WEBPATH [18] is a 3D visualization of a user's browsing history, developed by Frecon and Smith back in 1998. It uses the information stored in the HTML description of each website to produce a graph representation in 3D space. Each visited page is displayed as a cube, labeled with the page title on its top. The surface of the cube shows an extracted image from the HTML description. This image can either be the background image, the first image in-lined in the document or simply the background color, depending on the user's preference. ***VISVIP*** [17] by Cugini et al. is a tool for website developers and usability engineers that visualizes the paths taken through websites by the users. A 3D graph is generated, where visited websites are entries connected by edges, which represent transitions. The third dimension shows the time spent on each page visit.

All of the systems introduced so far use graphs to organize and visualize browser history data. While graphs are effective way to show the paths of web browsing activities, due to the connected nature of the Web, it is not clear whether following these paths provides sufficient aid for re-finding previously visited web pages. Thus, HistoryTime used a timeline-based approach for the main representation instead, as we wanted users to be able to travel back and forth in time while exploring their visited websites.

Card et al. proposed ***WebBook*** [15], another interactive 3D visualization. It uses book metaphors to aggregate web pages in virtual 3D books, where each page in a book represents a visited website. The aim of that project was to allow rapid interaction with objects at a higher level of aggregation. While the navigation in such a book might be intuitive, this design might not perform well in terms of search speed, due to the need for visiting most of the book's pages that precede the desired page. However, it offers a number of interesting applications, one being *relative-URL books*. This application clusters visited websites by recursively finding all relative URLs of a given page and displaying the results in a book.

The ***BrowseLine*** [21] 2D timeline visualization also uses domain clustering in their application. A hour-based timeline visualizes visited web pages, clustering them per domain per hour. The clustering used in both WebBook and BrowseLine is similar to the one we used in HistoryTime, as we cluster per domain per day. One advantage of clustering is that it more closely matches the user's mental model of their browsing activities [21]. In addition, it reduces visual clutter and provides overviews of the presented information. ***WebComets*** [16] is a timeline-based 2D visualization that also displays when a new window or tab has been opened, and visually encodes the time spent on a specific website, just like VISVIP [17], but through a comet tail. The ***HCB***

[24] system by Shirai et al. presents different types of information about them in terms of the currently displayed page, such as temporal sequence, URL-based proximity and content similarity. It analyses and compares the entries that have been saved in its own database, which goes beyond the visualization-only aspect of HistoryTime.

(a) Book mode (b) Circle mode (c) Cube mode

Figure 2.1: Different 3D layout modes of [29].

Yamaguchi et al. [29] proposed a bookmark system that additionally offers a 3D browsing history function, a marker function and a look-ahead loading function. Various layout modes like book, circle and cube mode can be used to arrange the elements, as Figure 2.1 shows. Users can also arrange the order of the displayed web pages. The simple presentation of an image plane, as used in the circle mode, was an inspiration for our visualization. We do not encode information on our domain-elements that uses the third dimension, such as the time that was spent on a cluster, due to API limitations. Thus, we decided to use planes instead of cubes, as opposed to WEBPATH. The slider that is shown in Figure 2.1 lets the user control the general transparency of the displayed elements. In HistoryTime, we use transparency to enhance the visual experience by fading out elements that are further away from the viewer, and elements that do not share a connection to a selected one.

Unlike HistoryTime, all of the browser history solutions mentioned above save content onto the user's hard drive by logging various information, such as thumbnails, and the time spent on a certain website. HCB [24], for example, saves the the entire source of all web pages visited. We use the unmodified Chrome history data without any additionally logged information, so that our extension can always visualize the present history data, no matter when it was installed.

CHAPTER 3

HistoryTime

The standard history application of Google Chrome offers a simple list-view for all recently visited websites. We wanted to enhance the user experience of history browsing by providing an aesthetic visual representation of the data, making the history itself more pleasant to use. The most important requirements were as follows:

- **Aesthetics**
  This was the main requirement of our project. We wanted to make users have fun exploring their histories by offering an intuitive, motivating design.
- **Usability**
  The extension developed with all its functions had to be at least as usable as the standard Chrome history.
- **Data**
  The history data for our visualization had to be retrieved via the chrome.history API only.

We planned to cover the basic user scenarios that were possible with the standard Chrome history, and added a few more scenarios that seemed plausible when revisiting previous web pages, such as the selection of a specific date and the possibility to order elements by certain criteria.

3.1 Data

The data for the visualization had to be exclusively retrieved via the *chrome.history* API [2], as we wanted to keep the extension as ligthweight as possible. HistoryTime does neither log nor save anything on its host system. This way, the extension can be installed on an arbitrary desktop system that supports the Google Chrome browser, and can also instantly visualize history activities that took place beforehand. Google's extension API

is very well documented and offers a wide range of functionalities that were sufficient for our needs.

The API basically revolves around two kinds of objects:

- **HistoryItem**
  This object encapsulates one result of a history query. It is returned once per URL and offers additional information, such as the title, the visit count and the latest visit time of one particular item.
- **VisitItem**
  For each HistoryItem another search can be performed in order to retrieve every single visit of a particular URL. A VisitItem encapsulates one visit to a URL, with information like the visit time, the transition-type to this URL and also the referring website.

A fair amount of data is available via the Chrome API. Unfortunately, certain information could only be visualized and retrieved by specialized logging, such as the time spent on a particular URL or screenshots of the websites visited. The API also supports the manipulation of the history recorded. However, this is not utilized in the current prototype of this extension, as the goal was the visualization of the browsing history, not its manipulation.

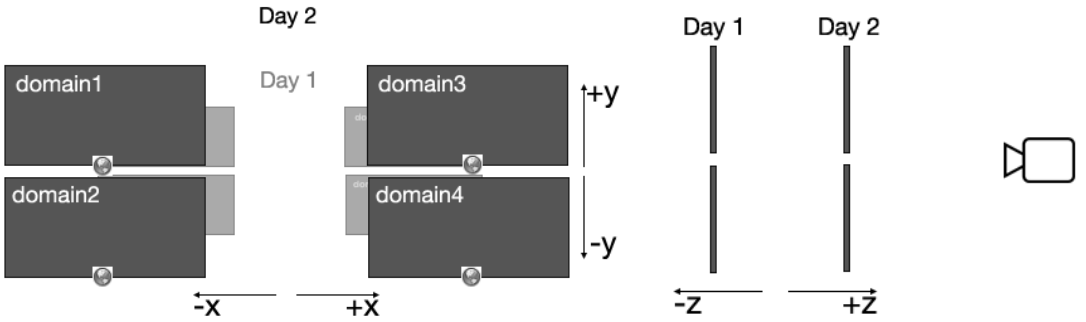
3.2 Visualization

For the visualization part of our Chrome extension, we wanted users to be able to *"dive into the scene"*. The exploration aspect played a vital role for the design decisions in the beginning of the project. The setting of the current version of our prototype is a space environment, which should contribute to the aesthetics and exploration part we want to achieve. The inspiration for the space environment, as well as the zoom-in and zoom-out part of the navigation, was drawn from the interactive Flash animation *The Scale of the Universe* [1].

The history data that is retrieved via the Chrome history API had to be visualized in the described environment. Figure 3.1 shows a mockup of the main view of our scene. HistoryTime clusters the retrieved data per domain per day. One element of a specific domain in the scene contains all visits made to that particular domain on a specific day. As the mockup shows, the x-axis and y-axis map the object layer, which is used for the placement of the elements, whereas the z-axis maps the time. This sort of mapping results in one layer of domain-elements for each day that has been retrieved by the API.

The object layers are placed in discrete intervals along the z-axis, where each interval represents one day in the visualization. The greater the distance of an object layer to the camera in -z direction is, the older are the entries of that layer. Analogous to this, the greater distance to the camera in +z direction is, the newer are the entries.

[1] `htwins.net/scale2/`



(a) Front view (b) Side view

Figure 3.1: Mockups of the mapping and structure of the scene.

Initially, we wanted to place one row of elements on the object layer. The scene felt empty with only one row present on each z-interval, and it felt too crowded with three rows of elements. Thus, we decided to use two rows of elements in the y-direction to compress the space used by the elements and to reduce the amount of panning needed when navigating through an object plane. The space that is left open in the middle of the scene is for immersion and orientation reasons only. We decided to go with domain clustering to reduce visual clutter in our scene. This helps to keep the scene clean and still provide the users with more informations about one domain on demand. A screenshot of how this mapping actually looks like in the prototype is shown in Figure 3.2.

In order to make the different layers of domain-elements more distinguishable, we changed their alpha value depending on their distance to the camera, as Figures 3.1 and 3.2 also show. The currently selected day is always fully visible, where the transparency of the layers behind it is sequentially divided by half. To further improve the rendering performance and reduce visual clutter, only a maximum of four days are visible at a time. Days that are not visible, but lie within the currently loaded date range, will become visible as the user zooms in or out in the scene.

Figure 3.3 shows the representation of domain-elements in the scene. We display the domain name, the total amount of visits, the amount of visits on a certain day and the favicon on each element, utilizing a HTML5 canvas for creating a texture. In our case, the total amount of visits is summarized per domain over the entire history available on the host system. The domain-elements' shape and size serve as a placeholder in the current version of the prototype, for an eventual display of thumbnails on each element in the future, as we discuss in Section 4.6.

To visually encode additional information on our domain-elements, we decided to use color values to highlight certain elements. We have tried two types of color encodings that are shown in Figure 3.3. We use the RGBA color space and modify the CSS property of an element's texture canvas to alter its background color. The elements in Figure 3.3a are coded by the amount of visits of a particular day by using thresholds of 1, 5, 10

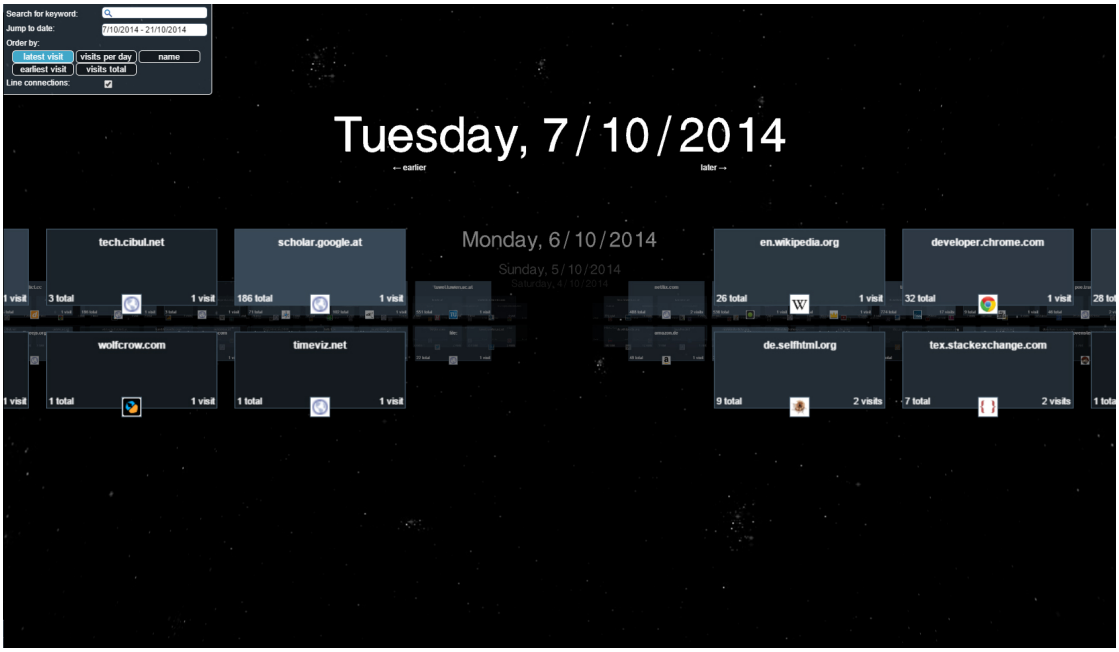


Figure 3.2: Screenshot of the main view of HistoryTime.

and 25 sub-elements to determine the color. The domain-element of "facebook.com" is displayed darker than the one of "reddit.com" because it was visited less often that day. Figure 3.3b shows the coding of total visits, how many times a URL of a given domain has been visited in the entire browsing history. To achieve this, we used the formula $value = (min_v + (max_v - min_v)/log(max_{visits})) * log(visits)$ to compute the color value for each channel, where min_v and max_v define the range for each channel, max_{visits} is the highest amount of total visits overall, and $visits$ is the amount of visits of the element to be colored. The "facebook.com" and "reddit.com" elements nearly have the same color, as their total visit counts are nearly equal. The consequence of this coding is that elements of a certain domain have a constant color in the whole visualization, as opposed to the first coding we presented. In the current version of the prototype, we chose to go with the second option, because it was the preferred option by the people we interviewed for our small user study, which will be discussed further in Chapter 5 of this thesis.

When a user selects a domain-element by clicking on it, all elements in the scene that contain the same domain are visually highlighted, where all other elements have their transparency reduced. In addition, a list of all sub-elements is displayed. Sub-elements are individual visits made to URLs of the selected domain on the currently active day. In this list, the entries are ordered descending by their last visit time. A click on such an entry either opens its URL in a new window or a new tab in the browser, depending on the user's browser settings. Figure 3.4 shows how the selection and the list look like in the current prototype. It also shows arrows on the left and right hand side of the

(a) Coding of visits per day (b) Coding of total visits

Figure 3.3: Screenshots of different color codings for the domain-elements.

domain-element. Those arrows can be used to navigate to the next or previous entries of the selected domain or search query. Implementation details regarding this part of the navigation are discussed in Section 4.5.

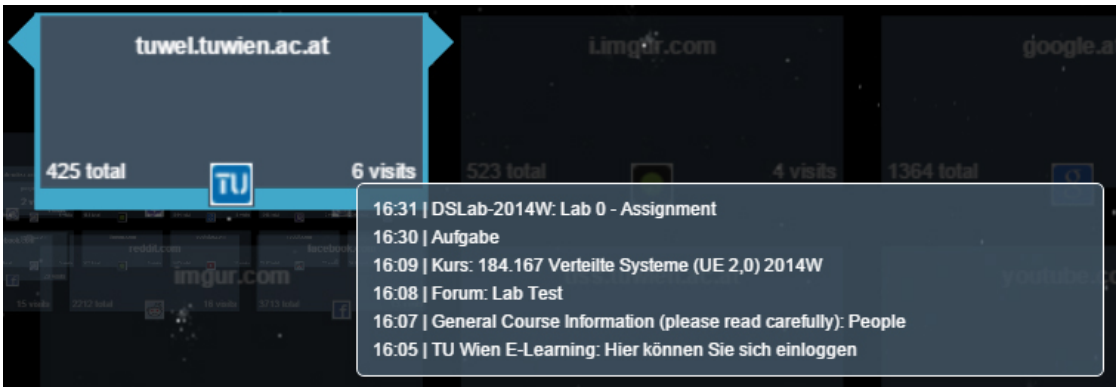


Figure 3.4: Selection of a domain-element.

3.3 Interaction

HistoryTime offers multiple ways for users to interact with the extension and to alter the view. Even though the visualization takes place in a 3D environment, we have restricted

the navigation for conceptual reasons. A free fly-through of the scene is not possible, as we do not encode information anywhere but on the front side of the displayed elements. Also, panning is only available to the left and to the right. The whole scene is constructed in two rows in the x-y-plane, so there is no benefit of panning up and down. Table 3.1 shows the keyboard and mouse controls that have been implemented.

Key	**Function**
Esc	Reset selections
Space	Reset camera position to the center of the current day
Left mouse	Select an element
Right mouse (hold)	Pan the view
Mouse wheel up	Previous day (-z direction)
Mouse wheel down	Next day (+z direction)
Up arrow	Previous day (-z direction)
Down arrow	Next day (+z direction)
Left arrow	Previous element of the selected domain
Right arrow	Next element of the selected domain

Table 3.1: Mouse and keyboard controls of HistoryTime.

A simple user interface (UI) contains further possibilities to interact with the visualization. A ***search-bar element*** can be used to find elements of a certain domain. The user can enter a keyword into the provided text-box. After pressing the *return* key, all elements which contain the provided keyword in their domain are highlighted in the scene and the camera moves to the first result found. The highlighted elements can then be browsed by using the left and right arrow keys, as well as the corresponding arrow-shaped UI that were shown in Figure 3.4.

A ***date-picker element*** makes sure that a user can load history entries by selecting one specific date or a specific date range. When a user clicks the text-box of this date-picker, a calendar widget is shown. On this calendar widget, one month is displayed at a time, with the possibility to switch to the next or previous months. A left mouse click on a day-entry in this widget loads and shows the data for the selected day. When clicking on a day-entry and holding down the left mouse button, a user can select a whole timespan for the extension to visualize.

The ***order of the elements*** that are rendered in the scene can be altered through different sorting algorithms. We felt that the standard Chrome history lacked alternate possibilities for sorting the displayed elements, other than the standard last visit order. Therefore, we implemented sorting modes to order the history elements by their time, amount of visits and names. The following sorting modes are possible:

- **Sort by last visit**
  The domain-elements are ordered by their last visited sub-element. On the right side of the visualization, the most recent visited webpages are displayed. This is

the standard ordering of both the Google Chrome history and HistoryTime.
This mode can be used to explore the elements in the order they were most recently visited.

- **Sort by earliest visit**
  The domain-elements are ordered by the earliest visited sub-element. Thus, it is basically the inverse version of last visit sorting. On the furthermost left side, the earliest visited element is displayed.
  In this mode, users can explore the elements in the order they were visited first on a selected day.
- **Sort by visits per day**
  The domain-elements with high amount of visits are moved to the right side. Those with a low amount of visits are moved to the left side.
  In situations where users are looking for a website they accidentally stumbled upon, thus having few visits, this sorting mode could be helpful to re-find it.
- **Sort by total visits**
  This sorting algorithm works in the same way as the previous one, but sorts the domain-elements according to their total visit count.
  It could be used for the same reasons as the visits per day one.
- **Sort by name**
  This action performs an alphabetic sort, where A is left and Z is right.
  This sorting could be useful in a scenario where users remember the first few letters of a website's name, and therefore want to browse through the entries in an alphabetically sorted manner.

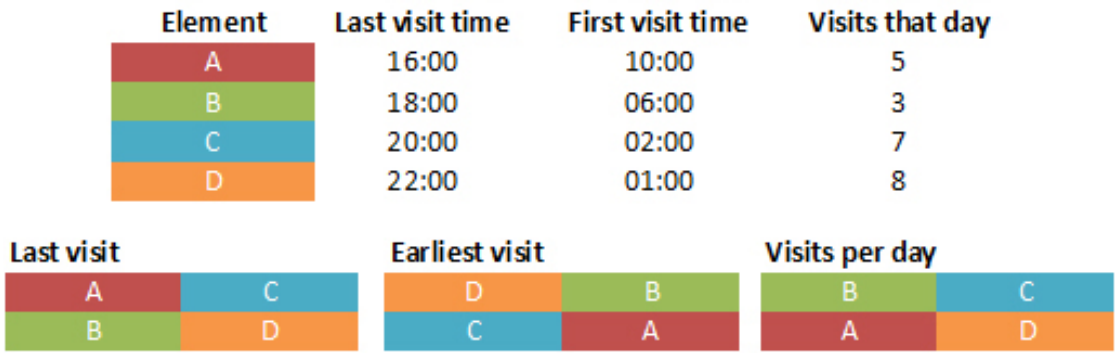


Figure 3.5: Mockup of the element's arrangement using the example of three sorting modes.

An illustration of how the elements are arranged when sorted is provided in Figure 3.5. When the *last visit sorting mode* is selected, the sorting starts with item that has the earliest last visit time on the top left side (e.g. A), and ends with the last visited item on the bottom right side (e.g. D). The *earliest visit sorting mode* works in the same

manner, but this time the first visit time is used for the actual sorting. The domain cluster that contains the first visited website of a day is arranged on the top left side (e.g. D), and also ends on the bottom right side (e.g. A). The *total visits sorting mode* works just like the *visits per day sorting mode*, which is shown in the mockup. The top left side is reserved for the element with the least amount of visits (e.g. B), whereas the lower right side contains the element with the highest amount of visits (e.g. D). The *name sorting mode* works the same way.

CHAPTER 4

Implementation

In addition to the Chrome.history API, HistoryTime utilizes a number of libraries.

The visualization part has been implemented using the ***three.js*** JavaScript 3D library [8]. Three.js is a lightweight 3D library that allows to create complex 3D scenes with minimal effort. Even though such wrappers for WebGL take away some of the manual control one has on the whole rendering process, like, for instance, on-demand texture-binding and memory management, it was more than sufficient for our needs.

HistoryTime uses animations to make the navigation smoother. To achieve this, the tweening engine ***tween.js*** [10] for JavaScript is used in our project. Tween.js is a simple, fast and easy to use tweening engine that uses optimized Robert Penner's easing functions for smooth transitions [23]. It offers great customization and callback functionality. In HistoryTime, we use it for moving the objects and the camera in the scene.

As JavaScript only supports objects and arrays to store data, HistoryTime uses the ***buckets*** [1] library. This library contains many different data structures and is well documented. HistoryTime mainly uses the *buckets.MultiDictionary*, *buckets.Dictionary* and sorting functions for arrays that are offered by *buckets.arrays*.

CibulCalendar [3] is the date picker library that has been used in our UI. It enables users to pick single dates or date ranges. It is also customizable and has a simplistic design that fitted perfectly to our user interface.

4.1 Data structures

HistoryTime displays domain-elements on the screen that contain all sub-visits of one specific date. This feature requires a data structure where one key element can hold more sub-elements. JavaScript itself only offers a limited amount of built-in datastructures: objects and arrays.

In our extension the *buckets library* is used to hold the information needed by the visualization and to make it efficiently accessible. The main data structure is a

buckets.MultiDictionary, which allows the use of multiple elements for each key. A JavaScript array is used to save such a multi-dictionary for each day.

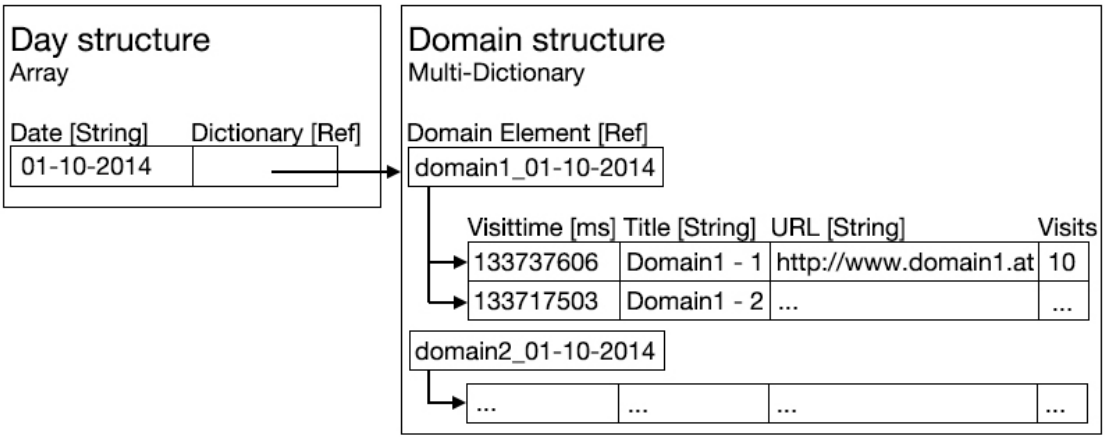


Figure 4.1: Mockup of the main data structure used to save the history data.

As 4.1 shows, one date, represented by a String, maps to one multi-dictionary. In this dictionary, scene objects that represent certain domain clusters count as the key to all visited websites (sub-elements) of the respective domains on a specific day, and is identified by its *.name* property (e.g. "domain1_01-10-2014"). For each sub-element, an array is constructed, which stores the last visit time, the title, the URL and the total visit count of each element.

Other less complex information needed by the extension, such as scene objects or favicons, are saved in either a buckets.Dictionary or a simple JavaScript array. The favicons of each domain are are loaded and stored in a buckets.Dictionary in the HistoryManager class at the start of the extension. This ensures that those images can be used for multiple elements in the scene, and only have to be loaded once, as the loading process is time consuming.

4.2 Scene objects

HistoryTime uses a variety of three.js objects to render the scene and navigate through it. A *THREE.Scene* object is used to store all elements that are to be rendered at some point during the visualization. In order to render the scene, a *THREE.WebGLRenderer* is used. This renderer has been chosen because it is a lot faster than the *THREE.CanvasRenderer*, as it uses the hardware-accelerated WebGL to render the objects on the scene, instead of the slower HTML Canvas 2D Context API [6]. A *THREE.Camera object* is used to alter the view.

Each individual domain-element on the screen is represented by a *THREE.Plane* object. On each of these objects a *THREE.Texture* is mapped onto, which is created

through a HTML5 Canvas. This canvas displays the name, the favorite icon, the total visit count and today's visit count of a certain domain.

For each day-level along the z-axis, a *THREE.TextMesh* object with the current date is created and placed. On each z-level, the specified text moves in sync with the camera, so the user always knows the currently selected date, no matter where he scrolls to reach the previous or next day.

For performance reasons, all objects in the scene have frustum culling enabled, so that all vertices that cannot be seen by the camera are not rendered. To further improve the performance, the automatic matrix updates are disabled after each animation in the scene, so the position, rotation and scaling of the objects is not automatically re-calculated every frame. This means that, if the position, rotation or scaling of an object changes in a frame, due to an animation for example, it does not appear in the scene, unless the matrix of the particular object is updated manually again. In order for our animations to work, we manually update the objects' matrices when needed.

4.3 Loading the history

HistoryTime retrieves the browsing history via the Google Chrome history API. In order to encapsule all communication and with this API, a JavaScript class named *HistoryManager* has been implemented. This class reads the browsing history and also retrieves and saves favicons for each domain.

Algorithm 4.1 shows the algorithm for loading the history in pseudocode. At first, by calling the asynchronous function *chrome.history.search*, a history query with a startTime is sent to the API. The API then returns an array of historyItems, which is further processed by a call of another asynchronous function, *chrome.history.getVisits*. As the getVisits function returns all visits to a given URL, the results have to be filtered by checking their visitTime. If the visitTime is inside the provided timespan, which is specified by a startTime and an endTime, it is added to the array. The default timespan is defined as today minus 2 weeks at the start of the application, but can be altered by the user through the date picker element in the UI. After all items were loaded, a callback is invoked, which returns the historyData array to the main class of HistoryTime.

For each domain that has been parsed by the API, the favorite icon is loaded and saved in a *buckets.Dictionary* data structure. To get the domain of a given element, its URL is matched with the regular expression $/(?:https?:\/\/)?(?:www\.)?(.*?)\//$ to filter out the "http://", "www." sequences and every character after the third slash, leaving only the domain for further processing. After that, we create a new Image object, enable crossOrigin because the image will later be displayed on a HTML5 Canvas, and provide the URL of a favicon service, concatenated with the element's domain for its *.src* property. The *onload* function then loads and saves the favicon in our dictionary. We use the free service of *getFavicon* [4] to retrieve the icons of a given domain, because it also allows cross-origin access. Cross-Origin Resource Sharing (CORS) allows us to request the favicon as a resource from another domain, outside our extension. Such cross-domain requests would otherwise be forbidden by Google Chrome, per the same-origin security

Algorithm 4.1: Pseudocode for the history loading via the Chrome API.

```
Data: HistoryItems and VisitItems
Result: Array of all visited websites of a certain timespan
historyItems ← chrome.history.search;
for i ← 0 to historyItems.length do
    visitItems ← chrome.history.getVisits(historyItems[i]);
    for j ← 0 to visitItems.length do
        currentVisit ← visitItems[j];
        if currentVisit inside timespan then
            historyData.add(currentVisit);
        end
    end
end
return historyData;
```

policy. The Google S2 Favicon Service [5], for example, does not support CORS and therefore prevented the received favicons to be displayed on our canvas, therefore not fitting our needs.

4.4 Building the scene

When the array of history data has been returned to the main class, we iterate through it and prepare the visualization. Algorithm 4.2 shows the most important steps in pseudocode. For each element in the returned array, we check if there is already an entry in our day structure present. If not, we add the date of the current element to our day structure and create a multi-dictionary to store our domains and sub-elements for this date. In addition, a *THREE.TextMesh* with the same date is created and saved in the textsArray for later use. After that, we check if there is already a domain-element saved for this date. If not, we first create a HTML5 Canvas using the Document Object Model (DOM) that will display the information and colors we discussed in Section 3.2. With the finished Canvas, we invoke the *THREE.Texture* constructor to convert it to a texture. As we need both a texture and a *THREE.Geometry* object to display a textured element in the scene, we create a global geometry object on startup of the extension, which contains the vertices for our plane. This object is then reused for all domain-elements for performance reasons, as all of them share the same size and shape and we do not encode any information in either of those attributes. We then create a *THREE.Mesh* with the global geometry and map the texture onto it. Finally, the information of each element is copied into the multi-dictionary, using the previously created mesh as its key.

After the history data has been processed, the main data structure is traversed, so that each day is processed separately. The algorithm for this is shown in Algorithm 4.3 in pseudocode. While iterating through the elements of a certain day, we check if there has been any non-default sorting mode selected in the UI. If so, we sort the elements

Algorithm 4.2: Pseudocode for preparing the scene after receiving the history data.

```
Data: Array of history data [visitTime, title, URL, totalVisits]
Result: Prepared dayStructure and scene objects
for i ← 0 to historyData.length do
    current ← historyData[i];
    if !(current.date in dayStructure) then
        dayStructure.add(date);
        create multi-dictionary for current.date;
        create THREE.TextMesh for current.date;
        textsArray.add(textMesh);
    end
    if No mesh for domain exists for current.date then
        create HTML5 Canvas;
        create THREE.Texture;
        create THREE.Mesh(globalGeometry, texture);
    end
    multi-dictionary.add(current);
end
```

accordingly. If not, nothing sort-related happens, as the history data itself is already sorted by the last visit time by API definition. We then count the number of elements that have to be arranged for this day, and compute a translation vector starting point. For each element, its position is translated by the translation vector. The vector itself is updated for each element processed (i.e. the y-coordinate is flipped and the x-coordinate is increased every two elements), and the element is added to the scene. The text mesh that has been previously created and saved for each day is also added to the scene after a day has been fully processed. Finally, after all elements of all days have been arranged, a tween.js animation is started, which results in a nice effect of the domain clusters "zooming in".

4.5 Navigation and Queries

Three.js offers free examples for many common scenarios. In our extension we used the ***OrbitControls.js*** [9] example which offers a fully functional camera control system. We modified the original file so that the orbiting and zoom function were disabled, because our own customized versions were already implemented. We ended up using its pan function only and restricted it by a maximum-range parameter in x direction, which limited the panning to the utilised part of the scene, and also disabled panning in y-direction. This maximum-range was calculated based on the position of the furthermost elements in both +x and -x direction.

Algorithm 4.3: Pseudocode for building the scene.

```
Data: Array of scene objects
Result: Renderable scene
for i ← 0 to dayStructure.length do
    currentDay ← dayStructure[i];
    if sorting selected then
        currentDay.sort();
    end
    translationVector.x ← currentDay.length/4;
    translationVector.y ← 1;
    for j ← 0 to currentDay.keys do
        currentElement ← currentDay.keys[j];
        currentElement.translate(translationVector);
        scene.add(currentElement);
        if j%2==1 then
            translationVector.x− = 1;
        end
        translationVector.y∗ = 1;
    end
    scene.add(currentDay.textMesh);
end
animation.start();
```

Every time a user zooms in, the camera is moved in -z direction. When a user zooms out, it is moved in +z direction. This movement is also restricted, so that the user cannot move past the first and last days in the scene. Additionally, the transparency of the elements is adjusted as described in Section 3.2.

When a user clicks on an element in the scene, ray casting is performed via the *THREE.Raycaster* class. We first unproject the screen position of the mouse cursor with a *THREE.Projector*, to convert it to scene coordinates. We then send a ray from the camera's position in the direction of the resulted mouse vector and determine the intersections. Whenever an intersection occurs, we always select the nearest element to the camera, as we only allow elements of the currently active day to be selected.

After the raycasting returned a result, a list containing all saved visited websites of that domain-element is created and displayed via the Document Object Model (DOM). This list is positioned by first projecting the object's scene coordinates to screen coordinates, using the *THREE.Projector* class, and applying the result via CSS style properties. After that, the entries of this list are looked up in the main data structure, as the raycasting returns the nescessary key-object to retrieve the data. In addition, all elements with the same domain name are looked up in the data structure. The objects with the same domain are then highlighted in the scene, by creating *THREE.Plane* objects in a bright blue color, and placing those objects slightly behind the corresponding domain-elements.

All highlighted elements are then connected with *THREE.Line*s, so that the user can see the position and connection of the related elements better. The elements which are not related to a selected domain have their alpha value reduced by 80%, so that the context is not lost entirely upon selection, but the similar elements stand out more. The result has been shown in Figure 3.4 of Section 3.2.

At the same time the list of visited websites is displayed, ***arrow-shaped elements*** appear on the left and right hand side of the selected domain-element, as shown in Figure 3.4 of Section 3.2. Those arrows are essentially DIV elements with modified CSS properties, which make them look like arrows. The placement of these arrows works in the same manner as the placement of the list does. When a user clicks on one of those arrow-elements, the next or previous highlighted element is looked up in the main data structure. The camera then moves to the position of the next or previous element by applying a tween.js animation that translates the camera's current position to the desired element's position. After that animation is finished, a list containing the saved visited websites and the arrow-shaped elements appear again.

The ***search bar*** lets users find history entries by typing a keyword in the corresponding text box. When the *return* key is pressed, the application searches every multi-dictionary in the main data structure and matches the *.name* properties of the key elements with the keyword provided. If the domain contains the keyword, the same highlighting we described in the previous paragraph is applied. After the whole data structure has been processed and the elements have been highlighted, the camera is moved to the first element that was found in the currently active day or previous days. This movement is realized by applying a tween.js animation, which starts at the camera's current position and ends at the first element's position.

The ***date picker*** allows users to select one date or a range of dates in the appearing widget. When a date is picked, the value of the corresponding text box is changed to the selected date. To grab the selected date or date range, the *CibulCalendar* API [3] is used, which changes the value of the text box in the format *"DD/MM/YYYY - DD/MM/YYYY"*. If only one date was selected, the text box will only show this date. As soon as the text box changes its value, a listener picks it up, performs a validity-check, and begins to reset the scene. When the scene is reset, all domain-elements, highlighted planes, connection lines and text-objects are removed from the scene and deleted from their respective data structures. Their textures are unloaded from the GPU to free memory. In addition, the camera is reset to its origin, and the main data structure is also cleared. After the scene reset, essentially the process showed in Algorithm 4.1 is invoked with the specified startTime and endTime that was read from the text box, and the Algorithms 4.2 and 4.3 are invoked again to re-build the scene. As soon as the whole re-load process is finished, the scene only shows elements of the specified date range.

When a ***sorting mode*** is selected, the process of Algorithm 4.3 is applied. The pre-sorting ensures that the elements will be placed in the correct order. Instead of adding the elements to the scene, they are translated with a tween.js animation, which starts at the element's current position and ends at the translation vector's position.

4.6 Possible Improvements

Many history visualizations that were mentioned in Chapter 2 include thumbnails of the viewed pages in their browsing history representations. This feature allows the users to visually identifiy pages easier by taking advantage of the human capacity to recognize previously viewed images [26]. Unfortunately, regarding the HistoryTime concept, using only the Google Chrome history API also comes with certain limitations, as there is no possibility to retrieve a screenshot of a visited website via the API. A logging functionality could save a thumbnail for each website visited in the background, which could then be used in the representation of the domain-elements in the scene. At the time of this thesis, there exist free thumbnail generation services on the Web, but unfortunately they were too slow to make them work with the extension. There are many paid services available, which provide this exact service for a monthly fee and claim to be faster. The problem with retrieving website thumbnails via such a service is that it first has to be generated on-demand, and therefore depends on multiple factors, such as the Internet speed, one cannot influence directly. HistoryTime would need a high amount of such images, and thus it might be more efficient to save them locally via the extension as soon as a user visits a website.

Another possibility would be to set up a server for HistoryTime that utilizes the *Open Graph protocol* [7], which Facebook also uses to create the previews of websites or Youtube videos, for example. We would need a server for this, as we cannot retrieve a cross-domain HTML page in our extension, due to the *same-origin policy* that prevents such requests for security reasons. Essentially, the server itself would act as a client and ask another server for a specific website, exactly the same way that a browser client would do. A script that would be triggered by a request from the extension, acting as the front end, loads the specified page, scans its metadata for the *og:image* tag and returns its URL to our extension. Cross-origin resource sharing would then allow the extension to communicate with the server, because we would have control over both the server and the extension.

CHAPTER 5

Evaluation

To evaluate the usability and aesthetics of HistoryTime, a small user study was performed. For this, our extension was compared to the Google Chrome standard history (CSH) in various aspects. The aim was to investigate if users found a 3D, WebGL-enhanced history more usable and aesthetically more appealing than the CSH. Thus, we designed a comparative study where the participants were asked to solve five tasks, followed by a short questionnaire.

Six volunteers, 4 male and 2 female, were recruited from three universities in Vienna and Wr. Neustadt. The ages ranged from 20 to 25. Three participants were computer science students, two participants were technical physics students and one participant was a technical mathematics student. Five of the volunteers were experienced computer users, but only one of them used their browser history frequently in his working environment. Each participant worked on the provided tasks and questionnaires individually.

5.1 Design

All participants used the same computer with the same versions of Google Chrome, HistoryTime and the browsing histories respectively to complete the procedure. This resulted in a better comparability, but also caused an unnatural environment as the users did not get to use their own browsing history during the process. The host system had Microsoft Windows 7 installed and the monitor had a screen size of 27 inches with a resolution of 1980x1080.

To evaluate the two history types, the following tasks were prepared:

1. **Find the first and last visited websites of today/yesterday.**
   This task covers a rudimentary feature of browsing histories, as it is even possible with the simplest browser standard histories.

2. **Find the least and most visited websites of today/yesterday.**
   The visit count of websites can be useful to explore browsing habits (most visited websites), or to re-find websites that users just stumbled upon (least visited websites).
3. **Find a specific website using the search bar.**
   Searching for a text query counts as a rudimentary feature of history mechanisms and has therefore been covered by this task.
4. **Find all visited websites from one specific date.**
   This task covers a basic scenario where a user remembers an approximate visit time of a website and wants to search for a history entry in a certain time range.
5. **Find the most/least visited website of last week.**
   This is a complex task. It is designed to find out whether the CSH or HistoryTime functionalities allow to solve a problem through a combination of their features, or not.

The users had to perform these tasks with both histories. Each of those tasks had to be rated between 1 and 5, where 1 stood for *very hard to solve* and 5 for *very easy to solve*.

In addition to the tasks listed above, the System Usability Scale (SUS) questionnaire [14] was prepared. This questionnaire provides a reliable, low-cost usability scale that can be used to evaluate and compare the usability of a system to others. It contains the following questions, which were to be rated between 1 and 5, where 1 stood for *strongly disagree* and 5 for *strongly agree*:

1. I think that I would like to use this system frequently.
2. I found the system unnescessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system

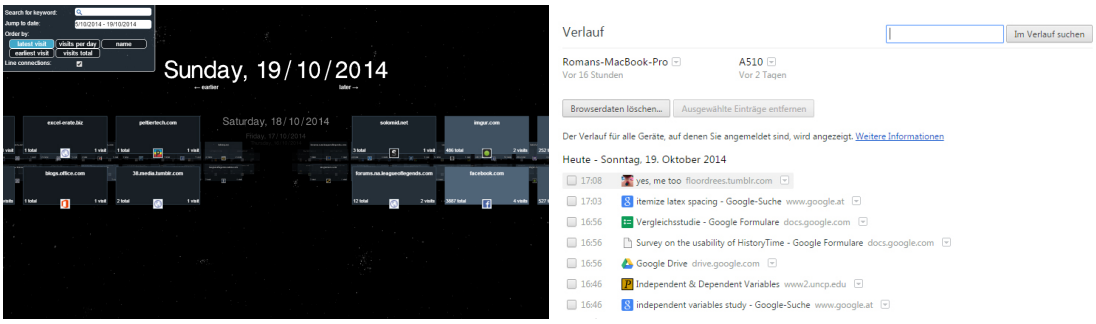
This questionnaire was issued after the tasks, together with two specific questions, to evaluate and compare the usability and aesthetics of the two systems. The following questions were to be rated between 1 and 5 as well, where 1 stood for *strongly disagree* and 5 for *strongly agree*:

1. The presentation of the history entries is good.
2. The positioning of the displayed elements is intuitive.

The independent variables for this study were the two types of histories (e.g. HistoryTime and CSH) and the tasks we provided. The history data was equivalent but slightly varied to accommodate for the learning effect, and the questionnaire was completely identical for each participant.

All participants were divided in two groups of three users each. We used within-subjects design for the tasks, where all users had to solve the same tasks with slight variations (e.g. today/yesterday, least/most). One group performed the tasks for the HistoryTime extension first, followed by the tasks for the CSH. The second group followed the same procedure the other way around (i.e. CSH first, HistoryTime second).

Initially, a separate history was loaded into Chrome by switching to a second user account in the browser. The volunteers were then asked to freely play around and explore the various functions to make themselves familiar with the first extension. After a few minutes, the history was replaced by a different one, and the users were asked to solve the tasks, one task at a time. For each task, the difficulty rating, the time and interaction steps needed to solve the tasks and any form of verbal feedback were noted by an observer. One step was defined as one click, pan, scroll and an entered keystroke, with the pan and scroll interactions being counted on release of the finger or button. Typing inside one of the text boxes was also counted as one step. In the beginning of each task, the scene was re-set to the standard start screen, which was identical for each participant. Figure 5.1 shows the starting positions for both HistoryTime and the CSH. After all tasks were completed, the participants answered the questionnaire. This procedure was then repeated with the second extension.



(a) HistoryTime (b) Chrome standard history

Figure 5.1: Starting positions for each task during the user study.

5.2 Results

Each participant was able to complete all five tasks with HistoryTime. However, they had problems to complete some of them with the CSH. Figure 5.2 shows the average

simplicity rating for each task, for both history types. As the chart shows, three of the five tasks had an average simplicity rating of 1 for CSH, which means that it was very hard, or in these cases, impossible to solve within a reasonable amount of time. The tasks *"most visited website of last week"* and *"least and most visited websites of today"*, which were about finding a website with a certain visit count, could not be solved with CSH. The users started to count the entries, but gave up rather soon. The *"most visited website of last week"* task was not even tried. In summary, all of the six participants gave up on those two tasks, and were only able to guess the answer, but could not tell for sure.

The task, which was about finding the visited websites from a specific date, was also not solvable in CSH within a reasonable amount of time. In HistoryTime, users seemed to have no problem loading the entries of a specific date, as it was rated to be *very easy*.

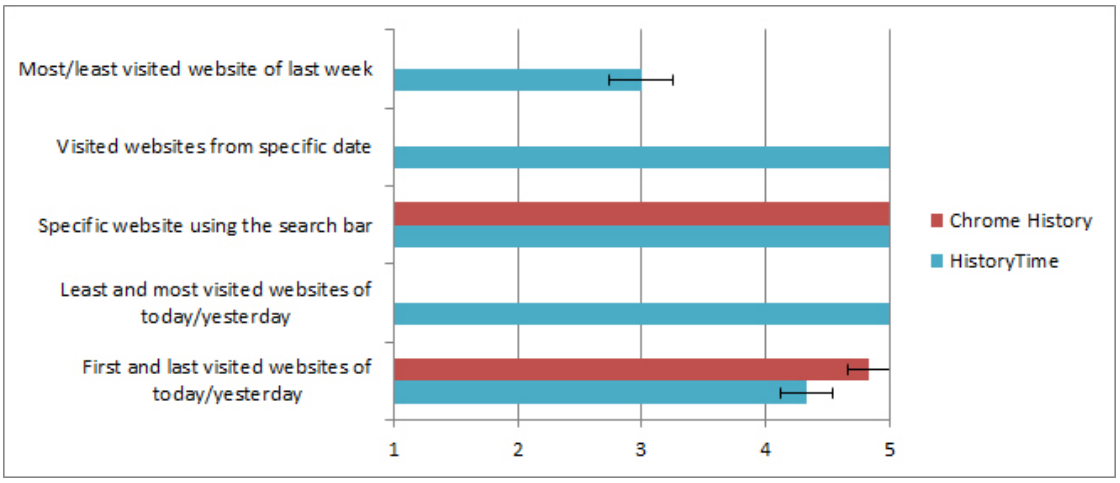


Figure 5.2: Result of the average simplicity rating for each task. Error bars show one standard error above and below the mean.

As the main reason for this study was to compare HistoryTime to CSH, we decided to only evaluate the intersection of tasks both extensions were able to successfully solve. Figure 5.3 shows the average steps and time needed to complete the *"find a specific website using the search bar"* and *"first and last visited websites of today/yesterday"* tasks. The chart shows that the participants needed slightly more steps, but equal or less time to complete those tasks with HistoryTime, compared to CSH.

To evaluate if there was a significant difference between the results of HistoryTime and CSH, we performed paired T-tests to compare the equality of the step- and completion time means. Table 5.1 shows the means of time and steps needed to complete the tasks, and the resulting p-value. For a given *significance level of 95%*, we could not find any significant differences, as $p > 0.05$ was valid for each test, but the sample size was also very low.

The evaluation of the SUS and specific questionnaires are plotted in Figure 5.4. We have added adjective ratings of the SUS score (e.g. ok, good and excellent) in the manner that is proposed by Aaron Bangor et al. [13].

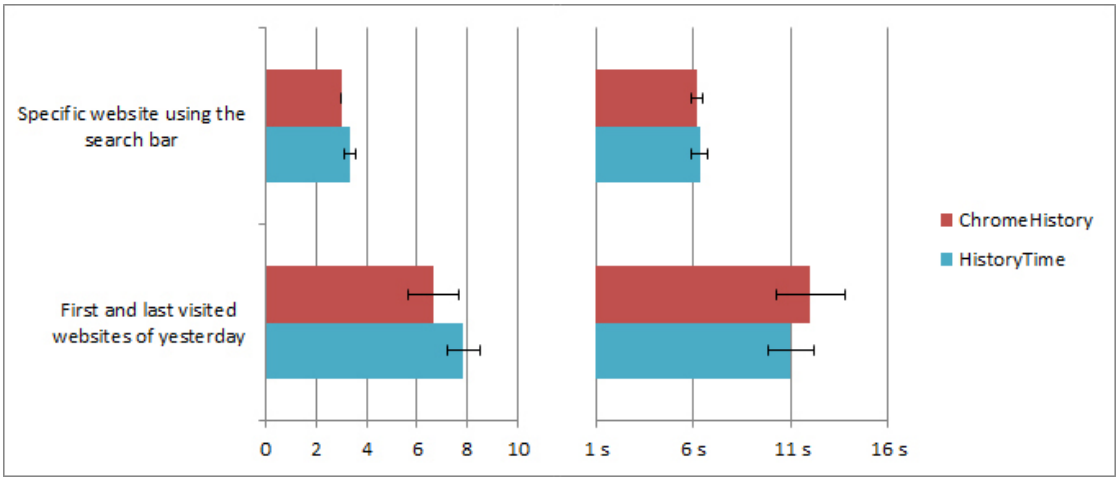


Figure 5.3: Plot of the resulting average steps time needed to complete the tasks. Error bars show one standard error above and below the mean.

Task	**HT time**	**CSH time**	**P-value**
Specific website using the search bar	6.00s	6.00s	0.6109
First and last visited websites of yesterday	11.00s	12.00s	0.1747
	HT steps	**CSH steps**	**P-value**
Specific website using the search bar	3.34	3.00	0.6250
First and last visited websites of yesterday	7.84	6.67	0.4933

Table 5.1: Comparison of the two history methods, using the t-test to compare means.

We evaluated the ratings of every single SUS question according to the document of John Brooke [14] and found HistoryTime to score an average of *88.34*, which is an excellent result. The CSH scored an average of *82.92*, which is slightly worse but still considered good. In comparison, the study of Aaron Bangor et al. [13] shows that the SUS score of an average system is about 68-70.

The specific questions covered the presentation aspect of the study. The bar chart in Figure 5.4 showed that the visual presentation of the history entries was preferred by the participants compared to a simple text-based list. However, the positioning of the displayed elements was found to be more intuitive in the CSH than in HistoryTime. We have also compared the results of the two specific questions in a statistical manner, using a Wilcoxon signed-rank test. The results of this test are shown in Table 5.2. For both questions, we found no statistically significant difference between the two histories.

During the process, a lot of qualitative feedback was received. Five of six people had problems understanding the button labels of the different sorting modes without explanation. Also, the total visit count was not clear, because the participants did not know whether it shows the total amount of the current timespan or the overall total

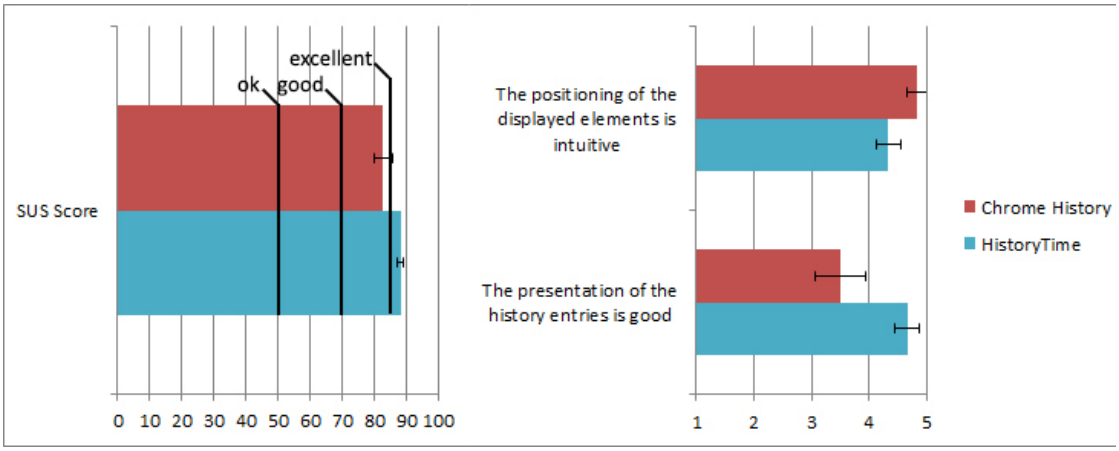


Figure 5.4: Result of the average SUS score and rating of the specific questions. Error bars show one standard error above and below the mean.

Task	**HT rating**	**CSH rating**	**U-value**	**P-value**
The positioning of the displayed elements is intuitive	4.83	4.33	-1.732	0.083
The presentation of the history entries is good	4.67	3.5	-1.841	0.066

Table 5.2: Results for the Wilcoxon signed-rank test for the specific questions.

amount of visits. The selection of a timespan through the date picker was denoted as a hidden feature, which was not to be found without a hint. At the time of the study, the prototype's color encoding on the domain-elements depended on the visits of a cluster per day. This was found to be inferior to the color encoding of the total visit count, so that regularly visited websites always had the same color. Other than that, the participants wished that the domain-elements had screenshots on them, to recognize them faster, but found the design very appealing. Finally, when searching for a keyword, three of the six participants wished that the camera moved to the first domain-element found and selected it.

5.3 Discussion

As the user study was conducted with only 6 participants, all with a solid computer science background, the results that were presented above can not be considered representative for web browser users. However, the feedback we received was very interesting and helped us to get a first estimate of HistoryTime's functionality, aesthetics and usability.

As much as 60% of the tasks provided could not be solved with the CSH within a reasonable amount of time, which could indicate that this history lacks functionality. The

tasks that involved determining some sort of visit count of a given website, i.e. *"find the least and most visited websites of today/yesterday"* and *"find the most/least visited website of last week"*, both failed to be completed by the users without counting the entries of each website themselves. We think that the visit count of websites could help people re-finding certain websites, especially those which were only visited once or twice and were forgotten afterwards. It could also help exploring and analysing a user's browsing behaviour, determining which websites were visited regularly.

The *"find the most/least visited website of last week"* task was designed to evaluate if the histories provided functionality in a way that users are able to combine some of them to reach a more complex goal. In CSH, users would have to count every single visit of the websites on each day, write them down and compare them afterwards. This is obviously not a reasonable way for participants to solve a task. In HistoryTime, the users sorted the elements by their *visits per day*, and then only had to compare the furthermost entries on one of the sides for each day. Additionally, the visit count is also provided on the displayed elements, so the users did not have to count themselves. It was encouraging to see that the possibilities HistoryTime offers were sufficient and obvious enough to solve this task, even though it was rated with a mediocre difficulty overall. We strive to improve HistoryTime's appearance and features to make complex tasks like this one easier to solve.

The task that was about finding websites of a specific date, namely *"find all visited websites from one specific date"*, also failed to be completed with the CSH by all participants. The structure of CSH, a textual list ordered by the last visit time of its entries, does not allow a fast selection or skipping of certain dates. Also, the search bar does not provide support for a time-based query. In a scenario where a user only remembers an approximate visit time of a website, and therefore wants to search for this entry in a specific time range, he/she will have a hard time using the CSH, if this time range is more than a few days away from the current date. In contrast, HistoryTime provides a date-picker where users can select one date or a range of dates they want to explore. This is probably why solving this task was found to be very easy, as the results in the previous section showed.

Only 40% of the tasks provided could be solved with both histories. These two tasks, namely *"find the first and last visited websites of yesterday/today"* and *"find a specific website using the search bar"*, were further evaluated in terms of steps and time. As we presented in the previous section, users of HistoryTime needed slightly more steps, but only equal or less time on average to complete these two tasks. One possible reason for the result of the *"find the first and last visited websites of today/yesterday"* task could arise from the different starting positions for each task, as Figure 5.1 showed. In order to find the last visited website of today, for example, users only had to first scroll all the way to the right in HistoryTime, but immediately saw the answer in CSH, so they might have needed less steps for the task at hand. However, after finding either the first or last visited website with HistoryTime, users only had to pan all the way to the other side of the visualization, possibly resulting in less time needed to complete the task.

The SUS scores both systems received were very high. One reason for this might be

that five out of six participants of our study had a scientific background and are quite familiar with the learning of technical systems. This could have affected the results of the SUS questions 2, 3, 4, 8, 9 and 10. Anyway, the results and the participants' valuable feedback motivate us to present our extension to a larger community. In a possible future long-term study we would like to interview more people, covering a wider age-span and also less technical-affine people, using their own web browsing history during the process.

The positioning of the history elements was found to be more intuitive in CSH, compared to HistoryTime. One reason for this could be that a generic list, where the elements are sorted in a descending manner depending on their last visit time, is easier to understand than the alternating sideways sorting mechanism HistoryTime uses.

Finally, based on the qualitative feedback we received, we decided to improve the prototype of HistoryTime. We changed the button labels based on the suggestions of the participants. We have also encoded the total amount of visits to a domain cluster in its color, instead of the visits of a certain day, which we feel was an important change. Websites that were rarely visited always have the same color, the same goes for websites that were often visited. This ensures an easier, consistent visual distinction of the displayed elements. When searching for a certain keyword, the camera now moves to the first element that was found in the current or past object planes. Unfortunately, we were not able to include website thumbnails in the final prototype of this thesis, as we did not find a performant way of retrieving screenshots of webpages without having to log and save them ourselves, as we discussed in Section 4.6.

CHAPTER 6

Conclusion and Future Work

In this thesis, we introduced HistoryTime as a 3D browser history visualization that was developed to encourage people to explore their browsing history, with the focus on aesthetics and usability. The primary contribution of this work is the use of WebGL to represent Google Chrome's browsing history in a 3D environment. A timeline-based navigation and the display of domain-clustered elements on the scene provide a compact representation of the history data. The exploration possibilities of the standard Google Chrome history have been enhanced with a 3D environment, various sorting modes and visual highlighting of the displayed elements, as well as time-based queries.

The results of the first user study were encouraging, as all participants showed great interest to explore HistoryTime and were excited about what it had to offer. In the context of this study, some of the provided tasks could not be solved with the Google Chrome standard history, whereas HistoryTime users were able to solve all of them. In terms of usability, HistoryTime received a slightly higher usability score, compared to the standard history. The participants found the presentation of the history data in HistoryTime, which was one of the main goals of this project, superior to the standard history. The qualitative feedback we received allowed us to improve certain aspects of the extension.

For a possible release of HistoryTime on the Google Chrome store, we first have to address certain unresolved design issues. We did not implement the display of thumbnail previews on the elements in the scene, because we could not find a way to generate them without implementing a logging functionality. However, based on the results of the user study we conducted and our personal opinion, we think it is mandatory to have thumbnails of the viewed websites available, as it greatly increases the visual recognizability and further improves the look and feel of the extension. We also want to make the hidden control possibilities more obvious, such as the selection of a timespan in the date picker, or the arrow key navigation. A more comprehensive user evaluation is also under consideration, because we want to let a greater pool of people use our history for a few weeks, to further study the usability and aesthetic aspects of HistoryTime.

Bibliography

[1] buckets - a javascript data structure library. https://github.com/mauriciosantos/buckets/, 2014. [Online; accessed 30-October-2014].

[2] chrome.history. https://developer.chrome.com/extensions/history/, 2014. [Online; accessed 30-October-2014].

[3] Cibulcalendar - a javascript date range picker. https://github.com/kaore/CibulCalendar/, 2014. [Online; accessed 30-October-2014].

[4] getfavicon. http://g.etfv.co/, 2014. [Online; accessed 30-October-2014].

[5] Google s2 favicon service. http://www.google.com/s2/favicons?domain=, 2014. [Online; accessed 30-October-2014].

[6] Html canvas 2d context. http://www.w3.org/TR/2dcontext/, 2014. [Online; accessed 30-October-2014].

[7] The open graph protocol. http://ogp.me/, 2014. [Online; accessed 30-October-2014].

[8] three.js - javascript 3d library. https://github.com/mrdoob/three.js, 2014. [Online; accessed 30-October-2014].

[9] three.js - orbitcontrols. https://github.com/mrdoob/three.js/blob/master/examples/js/controls/OrbitControls.js, 2014. [Online; accessed 30-October-2014].

[10] tween.js - javascript tweening engine. https://github.com/sole/tween.js/, 2014. [Online; accessed 30-October-2014].

[11] Anne Aula, Natalie Jhaveri, and Mika Käki. Information search and re-access strategies of experienced web users. In *Proceedings of the 14th international conference on World Wide Web*, pages 583–592. ACM, 2005.

[12] Eric Z Ayers and John T Stasko. Using graphic history in browsing the world wide web. In *Proceedings of the Fourth International World-Wide Web Conference*, 1995.

[13] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

[14] John Brooke. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189:194, 1996.

[15] Stuart K Card, George G Robertson, and William York. The webbook and the web forager: an information workspace for the world-wide web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 111–ff. ACM, 1996.

[16] Daniel Cernea, Igor Truderung, Andreas Kerren, and Achim Ebert. An interactive visualization for tabbed browsing behavior analysis. 2013.

[17] John Cugini and Jean Scholtz. Visvip: 3d visualization of paths through web sites. In *Proceedings of the 10th International Workshop on Database and Expert Systems Applications*, pages 259–263. IEEE, 1999.

[18] Emmanuel Frécon and Gareth Smith. Webpath-a three dimensional web history. In *Proceedings. IEEE Symposium on Information Visualization, 1998.*, pages 3–10. IEEE, 1998.

[19] William R Hendee and Peter NT Wells. *The perception of visual information.* Springer, 1997.

[20] Ron R Hightower, Laura T Ring, Jonathan I Helfman, Benjamin B Bederson, and James D Hollan. Padprints: graphical multiscale web histories. In *Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 121–122. ACM, 1998.

[21] Orland Hoeber and Joshua Gorner. Browseline: 2d timeline visualization of web browsing histories. In *Information Visualisation, 2009 13th International Conference*, pages 156–161. IEEE, 2009.

[22] Tamer Nadeem and Bill Killam. A study of three browser history mechanisms for web navigation. In *Proceedings of the 5th International Conference on Information Visualisation*, pages 13–21. IEEE, 2001.

[23] Robert Penner. *Robert Penner's Programming Macromedia Flash MX*, chapter 7. McGraw-Hill, Inc., 2002.

[24] Yoshinari Shirai, Yasuhiro Yamamoto, and Kumiyo Nakakoji. A history-centric approach for enhancing web browsing experiences. In *CHI'06 Extended Abstracts on Human Factors in Computing Systems*, pages 1319–1324. ACM, 2006.

[25] Linda Tauscher and Saul Greenberg. How people revisit web pages: Empirical findings and implications for the design of history systems. *International Journal of Human-Computer Studies*, 47(1):97–137, 1997.

[26] Colin Ware. *Information visualization: perception for design*. Elsevier, 2013.

[27] Harald Weinreich, Hartmut Obendorf, Eelco Herder, and Matthias Mayer. Off the beaten tracks: exploring three aspects of web navigation. In *Proceedings of the 15th international conference on World Wide Web*, pages 133–142. ACM, 2006.

[28] Sungjoon Steve Won, Jing Jin, and Jason I Hong. Contextual web history: using visual and contextual cues to improve web browser history. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1457–1466. ACM, 2009.

[29] Toshihiro Yamaguchi, Hiromitsu Hattori, Takayuki Ito, and Toramatsu Shintani. On a web browsing support system with 3d visualization. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 316–317. ACM, 2004.

[30] Wenhui Yu and Todd Ingalls. Trails–an interactive web history visualization and tagging tool. In *Design, User Experience, and Usability. Theory, Methods, Tools and Practice*, pages 77–86. Springer, 2011.