TÉCNICO LISBOA

TimeWarp: Animated transitions in big data streaming visualizations

Miguel António Oliveira Rocha

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Daniel Jorge Viegas Gonçalves

Examination Committee

Chairperson: Prof. João António Madeiras Pereira
Supervisor: Prof. Daniel Jorge Viegas Gonçalves
Member of the Committee: Prof. Sandra Pereira Gama

October 2021

This work was created using LaTeX typesetting language
in the Overleaf environment (www.overleaf.com).

Acknowledgments

Um obrigado ao professor Daniel Gonçalves, ao professor Daniel Mendes e ao professor João Moreira por toda a sua ajuda e pela enorme paciência para comigo. Um obrigado ao Mark e à Carolina por me ajudarem a controlar a minha ansiedade. Um abraço aos colegas de trabalho do DHA que não me deixaram desistir do percurso e à SINFO por me ter recebido de braços abertos.

Ao António, ao Tó, ao Artur, ao Afonso, à Carlota, à Luísa, ao Rodrigo, ao João Patrício, ao Bernardo, à Margarida, à Carolina, ao João, ao Duarte, ao Daniel e à Catarina por todo o seu apoio. Ao Pedro, um obrigado por todas as vezes que me acalmou e assegurou que ia conseguir. Ao Miguel e à Mariana por todo o carinho, pelas risadas noitadas dentro e por estarem sempre lá que precisei. A todes a que me acompanharem neste percurso, pelo bem e pelo mal, um sincero obrigado.

Acima de tudo, um obrigado de tamanho infinito à minha família. À mãe Carla, ao pai António, à avó Natália e à irmã Diana por todos os vossos sacrifícios e por terem feito isto possível. Sem vocês, não estava aqui. Um obrigado do fundo do coração. Conseguimos.

Abstract

Nowadays, millions of data are produced and transmitted between different points on the planet. These massive amounts of data are explored in big data, one of the most important and researched areas of computer science of today. Research into big data visualizations has increased in the last few years, as technology has also progressed, especially when we start discussing big data streaming visualizations. With streaming, the data arrives to the visualization continuously, without interruption and in real time. This specification, combined with the sheer volume and information that exists in big data, means that traditional visualization techniques are not suitable to represent this type of visualization. In this document, we present TimeWarp, a big data streaming visualization which research focus is achieving a visualization able to display data across several visual idioms with minimal loss of context and with minimal loss of information across different time spans while maintaining a consistent and linear performance for different data flow rates.

Keywords

Visual Analytics; Information Visualization; Performance Visualization; Performance; Big Data; Streaming; Animated Transitions; Visual Idioms; Context; Knowledge Retrieval

Resumo

Atualmente, são produzidos e transmitidos milhões de dados entre diferentes pontos do planeta. Estas enormes quantidades de dados são exploradas na área de big data, uma das mais importantes e pesquisadas da informática no mundo de hoje. A investigação sobre visualizações de big data tem aumentado nos últimos anos, à medida que a tecnologia também tem progredido, especialmente quando começamos a discutir visualizações que combinam big data com streaming. Com o streaming, os dados chegam à visualização de forma contínua, sem interrupção e em tempo real. Esta especificação, combinada com o enorme volume e informação que existe em big data, significa que as técnicas tradicionais de visualização não são adequadas para representar este tipo de visualização. Neste documento, apresentamos o TimeWarp, uma visualização de big data e streaming cujo foco de pesquisa é conseguir uma visualização capaz de exibir dados através de várias expressões visuais com perda mínima de contexto e com perda mínima de informação através de diferentes períodos de tempo, mantendo um desempenho consistente e linear para diferentes taxas de fluxo de dados.

Palavras Chave

Análise Visual; Visualização de Informação; Visualização de Performance; Big Data; Streaming; Transições Animadas; Idiomas Visuais; Contexto; Obtenção de Conhecimento

Contents

List of Figures

List of Tables

Acronyms

CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
JS	Javascript
SVG	Scalable Vector Graphics
2D	Two Dimensions
3D	Three Dimensions
API	Application Programming Interface
IDC	International Data Corporation
ZB	Zetabytes
MAP	Multi-dimensional Aggregation Pyramid
VR	Virtual Reality
HTM	Horizontal Transitions Module
HTV	Horizontal Transition Visualization
FPS	Frames per Second
VTV	Vertical Transition Visualization
VTM	Vertical Transition Module
UI	User Interface

1

Introduction

Contents

Nowadays, millions of data are produced and transmitted between different points on the planet. These massive amounts of data are explored in **Big Data**, one of the most important and researched areas of computer science of today. A more concrete definition of big data describes it as "large growing data sets that include heterogeneous formats: structured, unstructured and semi-structured data" [14].

Big data has very specific characteristics, mostly evidenced by the **5V**: variety, volume, velocity, value, and veracity [15]. Variety addresses the multiple sources that originate the data; volume describes the large amount of data generated every second; velocity talks about how fast the data is generated; value talks about the information and knowledge that can be extracted from big data; veracity relates to the correctness and accuracy of the information. Because of the required computational resources to handle the characteristics of big data, **traditional visualization techniques cannot handle big data**.

The processing and analysis of big data require significant computational resources to guarantee flexibility, scalability, and consistent performance - all characteristics of a good visualization. A visualization allows for the exploration of data and information, with the goal of obtaining knowledge pertaining to a certain context. This concept can be better explained as the idea of mapping data in its raw form into graphics, symbols, colors or textures can make it very hard to extract knowledge for the end user [16]. A good visualization allows for a simpler interpretation of the information on behalf of the end user with minimal effort required to extract valuable knowledge from it.

Most visualizations are applied to previously known static data sets. Research into big data visualizations has increased in the last few years, with the concept of **Streaming** getting coupled to big data visualizations to create big data streaming visualizations. With streaming, data arrives to the visualization in continuous fashion, without interruption and in real time, requiring processing, storing and profiling as it arrives. Applications working with data streams will always require two main functions: storage and processing. Storage must be able to record large streams of data in a way that is sequential and consistent. Processing must be able to interact with storage, consume, analyze and run computation on the data, requiring computational resources for these operations to be run in an efficient manner.

When one combines big data with streaming into a visualization - two elements that require a big amount of computational resources - traditional visualization techniques are deemed unsuitable, as they are not prepared to handle the type of heavy loads big data and streaming will place on the visualization. Therefore, we must investigate the best way to map data in big data streaming visualizations while searching on how to obtain the best performance out of a visualization. For that, our research will look into the context of the visualization and into the performance of the visualization.

The context of the visualization includes an analysis of its goals, tasks and data. In big data streaming visualizations, several factors, such as goals, tasks, state changes (of the data or the whole visualization), and actions from external sources [17], can cause changes to what the visualization displays to the end user. When a change like that occurs, they need to be detected and presented to the end user in a

way that is comprehensible for the end user to perceive the changes occurring to how the visualization is displayed to them, while doing those operations in a way it causes the least possible impact to the performance of the visualization.

The performance impact connects directly to the concept of performance visualization, a type of software visualization that includes aspects such as hardware performance [18]. Performance visualizations are used to evaluate performance, verify correctness, diagnose problems, and gain insight into structure and execution behavior [18]. Therefore, performance visualizations can be connected to the analysis of a big data streaming visualization to find if performance of the system is stable enough during execution. In the analysis of a big data streaming visualization from the point of view of a performance visualization, one needs to make sure context loss and information loss are as minimal as possible. A solution for that is to employ animation techniques between each visual idiom. Animation techniques employed between different visual idioms are called **Transition**s - a particular moment when we switch from one visual idiom to a different one, to suit the new representation, helping with maintaining context in a visualization and proving a more understandable visualization to the end user.

An example of this is the approach chosen for VisBig [8], a big data visualization. In VisBig [8], an aggregation technique called graceful degradation is employed to improve maintenance of context within the visualization. Graceful degradation involves aggregating data as it gets older into visualizations with progressive fewer level of details. In other words, as data gets older, it progressively gets aggregated into a visual idiom that is not as detailed as the idiom for fresh data. To help with maintaining context between the different visual idioms, VisBig [8] employs transitions between visual idioms to guarantee context is not lost across the visualization and information loss is minimal.

1.1 Objectives

Our objective is to create a big data streaming visualization – TimeWarp - able to display data across several visual idioms with minimal loss of context and with minimal loss of information across different time spans while maintaining a consistent and linear performance for different data flow rates.

Considering the characteristics that big data possesses, it is very important that TimeWarp is a visualization that stays relatively consistent in performance throughout its time span, but that can keep the visual idioms of each of its modules with minimal information loss. Minimal information loss can be achieved with smooth transitions between different stages of the visualization with every different stage corresponding to a different module and time span.

In order to combine our objective with the exploration of the computational resources required to handle a big data visualization, the major focus of this research will be given to **obtaining the best possible performance possible** for TimeWarp with the migration VisMillion and Change [13] to Three.js,

a WebGL[1] based Javascript (JS) library. WebGL is a JS Application Programming Interface (API) for rendering interactive Two Dimensions (2D) and Three Dimensions (3D) graphics within any compatible web browser without the use of plug-ins..

To perform the migration from D3.js to Three.js, the architecture from VisMillion and Change [13] will be maintained, including its visual idioms and the results from its study of animated transitions between those visual idioms. To understand the success of the migration process, performance tests will be used to compare the performance of VisMillion and Change [13] with our visualization. In our implementation, it will also be studied the impact of the implementation of the object Instaced Mesh on the performance of the visualization to understand if its performance gains compensate the drawbacks it causes to the manipulation of visual elements of the visualization.

[1] https://get.webgl.org/

2

State of the Art

Contents

In this section, we will discuss the state of the art in **Visual Analytics**, with a major focus on big data, streaming, animated transition and performance, and its intertwining concepts.

In the last couple of years, the amount of work developed in big data has increased. A study developed in 2017 claims that this comes from several organizations have become increasingly dependent on the knowledge extracted from big data [14].

Despite the increase, there are still few works related to displaying the (big) data and changes it suffers in real time. Each consequent sub-section will analyze several works related to big data visualizations, data streaming visualizations and animated transitions in visualizations.

2.1 Big Data Visualizations

The goal of information visualization is to aid end users in extracting knowledge from data represented within a certain context using graphical elements. As technology progresses, more data is getting generated. A study developed by the International Data Corporation (IDC) [19] suggests the data volume of the world will reach 163 Zetabytes (ZB) by the year 2025.

When we discuss data sets with large amounts of data, we are discussing **big data**. To obtain information and knowledge from any type of data, big or not, it must be represented in a comprehensible way to the human mind. This is where a good visualization comes in, although it is not an easy road to create one. The creation of a single, global (big) data visualization tool is very hard to come across, as multiple challenges arise, mainly connected to the 5V and how each domain is different and inserted into its own specific context.

In spite of that, it is very important to develop multiple alternatives that can eventually lead us to a fully automated data analysis process (as long as that process remains ethically correct). This will be crucial in the future, as, according to [20], extensive analysis of a (big) data set is needed before proper and useful information is extracted from big data. Today, the process of extracting knowledge from big data visualizations still requires user input, although research on the automation of that process has increased in the last few years.

A good example of the type of user input still required today is present on [21]. Users can interact with the proposed interface, named SkyViz [21], by defining the entire visualization context through seven coordinates. After the manual input is performed, SkyViz [21] translates the coordinates into a set of suitable visual idioms. This translation process is automatic and done through skyline-based techniques. Skyline-based techniques filter interesting points from a larger data set, making them a good candidate to become part of the automatic process for data analysis.

In spite of the recent technology to implement the automatic portion of the visualization, SkyViz [21] is a good showcasing of today's visualization paradigm - one where context is very important for the success of a visualization and one where changes can occur over time, as the visualization and its data

set evolve. As we have mentioned before, although, context only provides suitable guidelines for the direction we pretend on giving the visualization. Context alone cannot create a sustainable visualization.

MAP-Vis [1] is another good example of a visualization for the paradigm described above. MAP-Vis [1] allows for the visualization of spatio-temporal big data and its interface has the goal of enabling users to explore, in an interactive way, the big data set presented to them. It provides a variety of correlated visualization views: a heat map, a time series and an attribute histogram - Figure 2.1.

As mentioned before, when we are dealing with big data – in this case, the chosen data sets contain millions to tens of billions of records - simply using these visual idioms is not enough. Some pre-processing is required to be applied to the data before it can be visualized by the end user.

On MAP-Vis [1], this is done by applying an aggregation model called Multi-dimensional Aggregation Pyramid (MAP). MAP can support simultaneous hierarchical aggregation of time, space and attributes, and later, it can transform the derived attributes into discrete key-value pairs for scalable storage and efficient retrieval. Data aggregation is one of the most important techniques employed in big data analysis, as it allows for a more efficient visualization and a faster computational speed for each visual idiom to represent its corresponding data. In spite of those advantages, data aggregation has the disadvantage of increasing the possibility of data loss, as it conceals differences between and among important subgroup categories and might hide oddities in the data set that might be important for analysis of the data set.

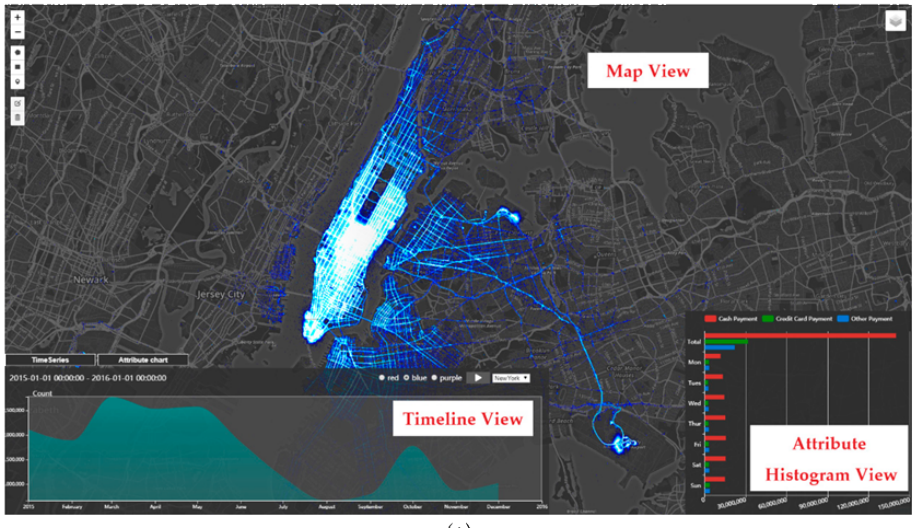
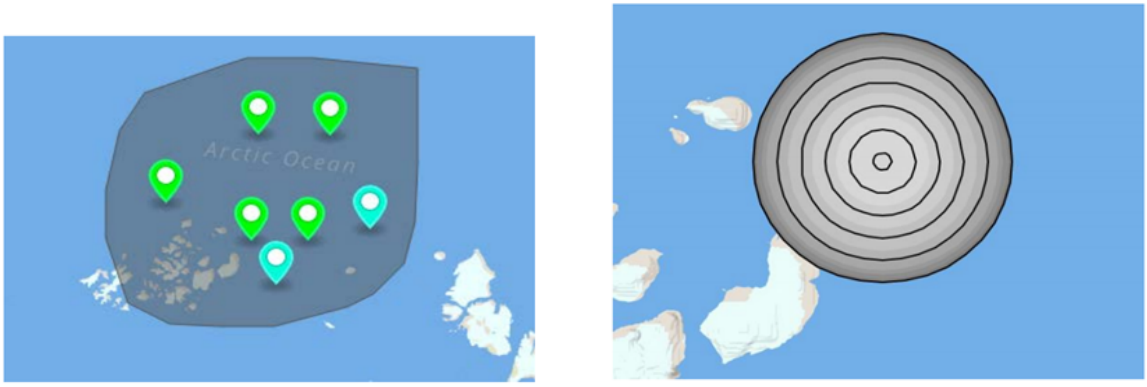


Figure 2.1: MAP-Vis web visualization interface. It includes three correlated views: map view, timeline view and attributes histogram view [1].

Representation of spatio-temporal data is just one of the many uses for big data visualizations today. As investigation into big data visualizations increased, it has become a staple in multiple fields that require big data sets to be represented in an efficient - when discussing computational performance - and comprehensible manner to the end user so it eases the process of knowledge extraction. One of the fields that has embraced the use of big data visualizations is the field of oceanography. An example of this is the work displayed in Figure 2.2 [2], where visualization techniques are used to analyze the acidification of the oceans at different fields of water and depths.



(a) General overview of the visualization.

(b) Site-specific visualization.

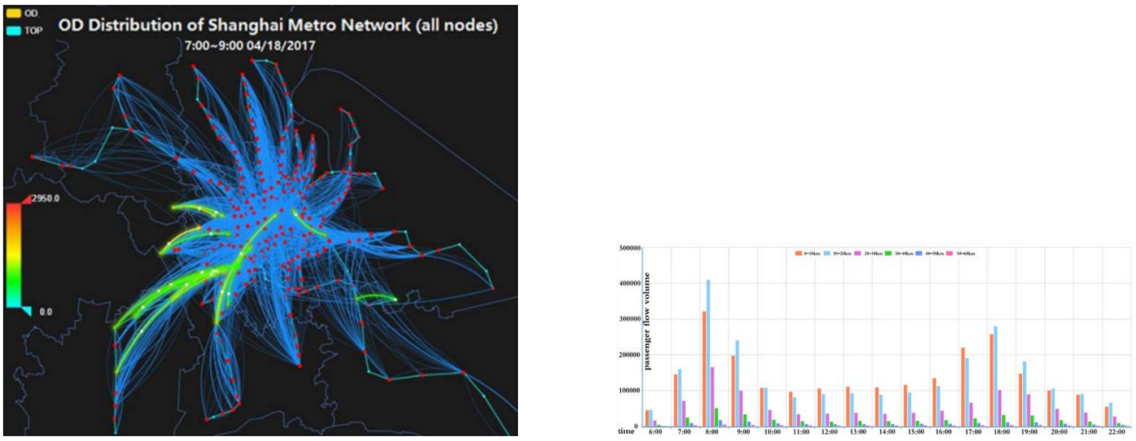
Figure 2.2: Markers, user drawn polygon and centric circles represent the data in this visualization [2].

The visualization displayed in fig. 2.2 uses a map as a basis. In the map, users can draw a polygon to reveal specific markers that represent the position of a real acquisition system - Figure 2.2(a). The action required to reveal those markers, however, does not provide enough information regarding the acidity value of the ocean water. To be able to showcase the acidity values, further user interaction was added. If the user clicks on a marker, a site-specific visualization - Figure 2.2(b) - is shown. The site-specific visualization consists of concentric circles, where each space between the circles (roughly equivalent to 100m in real life) is filled with a color part of a grey color scale of 256 values that represents the pH value for that section. This technique, employed for the representation of distances in visualization, allows for the simplification of comparing different values for different distances.

Another application of big data visualization is analysis of passenger flow of urban rail transit. An example of this is the analysis of the passenger flow of the Shanghai Metro Network [3]. The approach presented in fig. 2.3 is one that can be extended to other urban rail transit networks across the world, as they mostly share similar characteristics with each other. The work on [3] presents an analysis of four different data segmentation's, network passenger flow, line passenger flow, station passenger flow and section passenger flow, each one telling its own story to form a cohesive bigger picture.

For the passenger's origin and destination analysis, the respective data set is represented with a dynamic migration map, where we can get an overall picture of the flow between all the stations belonging to the Shanghai Metro Network - Figure 2.3(a). Date, period, line, or station are available filters to choose from. The visualization consists of a network that display the connections (links) between the different stations (nodes), colorized to represent the passenger flow volume and an arrow with the flow direction. An operation of aggregation was necessary during the pre-processing of the data, as the same transfer station has different station codes for different lines.

For the representation of the time-distance characteristics of the Shanghai Metro Network, a bar chart is used - Figure 2.3(b) - where the Y-axis represents the passenger flow volume and the X-axis the time of the day. Each bar represents how many passengers traveled a certain distance interval in a specific time of the day. The visualization of how many passengers traveled a certain distance interval in a specific time of the day is another example of an aggregation technique being used, as the data of several trips is joined to find the volume of passenger flow that traveled a certain distance in a certain period of the day.



(a) Dynamic migration map visualization

(b) Bar chart representing the time-distance characteristics.

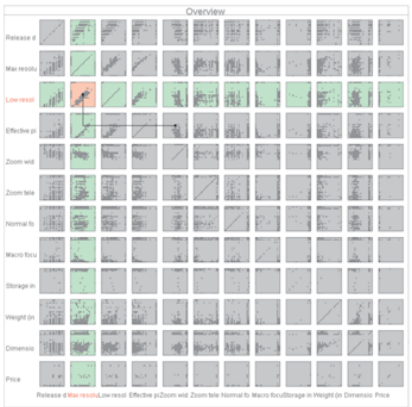
Figure 2.3: Visualization Analysis of the passenger flow of the Shanghai Metro Network [3].

The work on [3] is an example of how we can have multiple data types for analysis within limited dimensions. For this scenario, in order to obtain knowledge, correlations need to be performed. For example, in Figure 2.3(b), we observe a multi-bar chart for plotting different dimensions: each hour of the day between 6:00 and 22:00, the passenger flow volume - represented by the height of each bar - and, within each hour, a bar for how many kilometers of the Shanghai Metro Network passengers traveled. It is a lot of information to unpack, and while this multi-bar chart does a good job of displaying information in an easy to understand way for the end user, there are other techniques for simpler correlations.

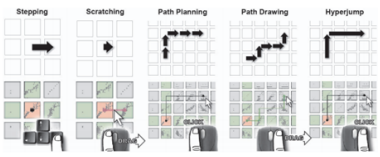
In big data visualizations, correlations can be used to represent multiple data dimensions within the same visualization. A good example of the application of correlations in big data visualizations can be found in the work Rolling the Dice [4]. Rolling the Dice [4] is a visualization for exploration of multidimensional data through combination of correlation matrices and scatterplots. While the second is one of the most used visual idioms to represent multidimensional data due to its simplicity, familiarity and high visual clarity [22], correlation matrices are a different breed. The use of correlation matrices allows for showcasing relationships between data and variables (through its columns and lines) in a quick and efficient way.

The combination of scatterplot and correlation matrices in Rolling the Dice [4] results in a scatterplot matrix - Figure 2.4(a) - where each scatterplot (columns) corresponds to a dimension of the data set (lines). This is a specialization of scatterplot visualizations, as these often give control of its mappings, from data dimensions to graphical properties, directly to the end user. Giving control to the end user over the visualization allows them to create their own combinations between different dimensions of the data set, contributing to further enhancing the exploration of the visualization and the data set for knowledge extraction. Still within the user interaction camp, *Rolling the Dice* [4] also employs some navigation operations.

These operations allow for smoother and easier navigation of the multiple scatterplots within the scatterplot matrix - Figure 2.4(b). A scatterplot matrix can be a great tool to employ in big data visualizations as it allows for exploration of multiple dimensions of the data set represented. The application of filtering techniques can also be of great use by the end user, allowing them to select the dimensions and data they want to visualize, allowing for a better exploration of the visualization and an overall better experience for the end user, as it can be customized to the context they are more interested in. Filtering also allows for a more efficient operation of knowledge extract on behalf of the end user.



(a) Scatterplot matrix component used for over-view and interaction.



(b) Overview of the navigation operations supported by the scatterplot matrix.

Figure 2.4: Main visualization and end user interaction properties in Rolling the Dice [4].

In visualizations like Rolling the Dice [4], where there is a lot of information shown to the end user, overplotting might become an issue. Overplotting is when data with similar values is aggregated and makes it almost impossible to analyze and visualize part of the data set in individual form. This is a common problem when we are using visual idioms with its basic visual representation in the form of points, as it is the case with scatterplots. A possible solution for overplotting is the plotting of quantitative data into categorical units, as shown in Colorized Mosaic Matrix [5]. Colorized Mosaic Matrix [5] is a visualization method for high-dimensional categorical data with color as a representation of its features - Section 2.1. Its plotting between the quantitative data into categorical units enables for a higher visualization of individual records, even if they have very similar values.

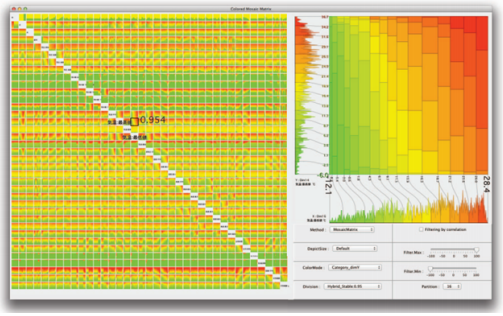


Figure 2.5: Proposed Interface for the Colorized Mosaic Matrix [5].

2.2 Data Streaming Visualizations

Nowadays, most of the applications are inserted in an ever-changing context. These are systems able to receive and process data in various shapes and forms (and from different sources) in real time. This is the concept behind **data streaming** and the dynamic data sets it creates. As mentioned, systems able to receive new data via streaming require constant updates. In other words, when we have a data streaming visualization, it must be able to display ever-arriving data to the end user in the most suitable way. During that process, it must make sure context is not lost across the visualization and that information loss is minimal.

A framework proposed on [23] allows for dynamic visualization of real-time streaming big data, in a way that is resilient to both its volume and rate of change. The proposed framework also investigates several challenges within (big) data streaming visualization.

The first challenge is connected to performing efficient processing and consumption of data streaming. In other words, how data can be processed and analyzed in an effective way. The second challenge is connected to automated detection of relevant changes in a data stream. In other words, anomalies need to be detected and highlighted for further analysis in behalf of the end user, meaning data representation can change over time to better suit the new requirements of the visualization.

The changes happening in the visualization are connected to the third challenge approached by the framework proposed on [23]. It involves choosing the best idiom for a certain context and data in a (semi) automatic way, through the aid of a recommendation engine. The representation of these changes must be done in a smooth way, so to guarantee no loss of context or information. This is one of the major challenges presented by the framework [23] as there is no fully automatic way to switch visualizations without losing context of what is being visualized. A solution to this problem is proposed in the framework [23], with the application of smooth and gradual transitions via intermediate visual idioms.

As established with big data visualizations, one of the most used techniques to simplify data is aggregation. In data streaming visualizations, aggregation continues to be quite commonly used and essential to the success of this type of visualizations. An example of this can be found on Unveil [6], an interactive and extendable platform with a real-time data set collected from passive and active attacks performed on smartphones. To display that data set in an efficient and comprehensive way, aggregation techniques are employed.

The collection of data on Unveil [6] is done through a collection of Raspberry Pi's, making up the first part of the system architecture. The Raspberry Pi's are managed by a back-end server responsible for analyzing and computing results from the collected data. The data set, after getting processed, is shown to the end user by a visualization platform, ran by a visualization server - Figure 2.6. The results displayed in Figure 2.6 are an example of a probe request analysis performed using Unveil [6]. Each dot represents the count of the number of devices detected in each location where the dot is placed. The bigger the dot, the bigger the count of detected devices for that location.

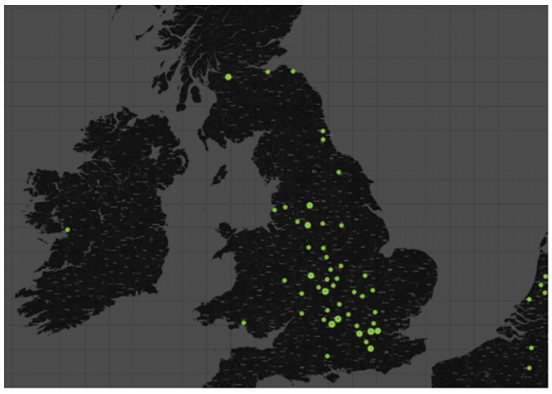


Figure 2.6: Visualization of results obtained with probe requests analysis on Unveil [6]. Each dot represents an aggregation of detected devices into a single location.

Another example of data streaming visualization is VAStream [7], a visualization that possesses capability for big data stream processing while providing user interaction at the same time. VAStream [7] is evaluated for two real-time streaming applications: real-time event detection using social media streams and real-time river sensors network stream used to detect water quality problems.

In VAStream [7], the raw data set that arrives is firstly filtered sp only data deemed important will be shown to the end user. The cleaned data set is then profiled and its meaningful features are extracted. Those features are then passed to an analytic module to extract the necessary knowledge from them. To obtain the map view presented in Figure 2.7, a simulation was performed with a simulated Twitter data stream of about 300 000 tweets per minute, where the streams are collected with a minute interval of each other.

Data streaming such the simulation performed to obtain the map presented in Figure 2.7 are high velocity streams of information and their processing needs to be done very quickly - almost in real-time. In order for this operation to be more feasible in terms of computational resources, micro batch-based processing can be applied to the data set. For every micro batch of data that arrives, a co-occurrence graph is built [24]. It is this graph that will allow for the deduction of an event and its location, such as the visualization map displayed in Figure 2.7.

Figure 2.7: VAStream map visualization. It showcases the location of each detected event [7].

Besides presenting its title visualization, VAStream [7] showcases some of the challenges encountered in creating data streaming visualizations. The first of those challenges is connected to reducing latency of the data influx while having a system still capable of achieving a high throughput for end-to-end processing from data consumption to visualization. The second challenge is connected to how the system adapts to changing workloads (or failures) and the third one is connected in how to provide flexibility in the infrastructure to adapt to the changing nature of the data and proper user demands.

Regarding the first challenge, a possible solution is to only process and transfer currently depicted data points, as shown in I2 [25], an interactive development environment. The solution presented in I2 [25] has an advantage compared to previous research, as it allows for direct integration into data analysis applications. Previously built solutions require an intermediate layer between database and visualization [26]. The additional layer increases processing time before obtaining the final visualization. The third challenge presented above leads us to a situation where, for a data streaming visualization to work properly, the amount of data arriving - no matter how much it is - cannot interfere (in a noticeable way) with the visualization performance. A possible solution to avoid performance hiccups is the use of graceful degradation, a concept vastly explored in VisMillion [8]. Graceful degradation is a concept where, as data gets older, it progressively gets aggregated into a visualization that is not as detailed as the visualization for fresh data.

VisMillion [8] consists of three different visual idioms: scatterplot, streamgraph, and bar chart - Figure 2.8. Each visual idiom, placed horizontally side by side in the visualization, represents a different, continuous time span, where older data gets represented by the visual idiom most to the left (bar chart) and the newer data by the visual idiom most to the right of the visualization (scatterplot). The intermediate continuous time span is represented by a streamgraph. The structure of the visualization, where each visual idiom exists within in its module, guarantees it does not lose performance despite the arrival of new data. Not only that, but it also allows for the creation of data history that can be visualized by the user.

To complement the VisMillion [8] visualization, a space to identify outliers exists. Due to its three different idioms, different aggregation (based on the timestamp of arrival) and processing techniques were used. VisMillion [8] allows for a better control of how much memory is used - as it does not increase continuously - and of which information will be rendered. The combination of these two functionalities makes VisMillion [8] easily scalable and flexible..

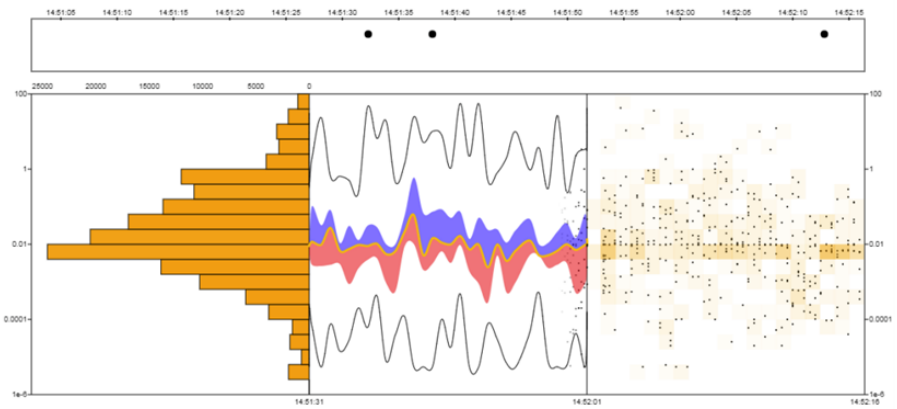


Figure 2.8: VisMillion interface [8].

2.3 Animated Transitions in Visualizations

Animations are used, in information visualization, for switching between different visual representations over time. They can be used within multiple contexts - each with their own complexity - but its main goal is to ease the perception of visual changes in the visualization for the end user. For that, a certain type of animation, called **transitions**, is used. A more concrete definition of transitions describes them as the "logical way to transform the input of a mapping into its output depending on their respective data type (e.g., text, color, shape)" [27]. Smooth transitions are important in a good visualization, as they can "shift a user's task from cognitive to perceptual activity, freeing cognitive processing capacity for application tasks" [28].

The concept behind animated transitions can be better explained by using an example like the User Interface (UI) rendering engine UsiView [27]. UsiView [27] ensures smooth transitions between different views within a UI. These views are the conceptual view - corresponds to the early stages of the designer's view for the UI - the internal view - consists of the code written by a developer to create the UI - and the external views - refers to the final UI visible and executable by the end user. These views are all connected by correspondences. These correspondences are named coding schemes. The coding schemes used in UsiView [27] are based on the work by [29] and are established and maintained with the goal of intertwining all the views together into a single package. This mapping is then used to determine transition types, based coding schemes, and later, correct animation technique to produce the final animated transition.

The idea behind the work on [27] is better explained in the research on animated transitions between some of the most basic data graphics out there, such as bar charts, pie charts and scatterplots [9], whose work focuses on two of the three levels of analysis described on [30]: syntax and semantics (the third level is called pragmatics). While syntax concerns the visual marks and their composition, which are used to determine coding schemes, semantics, on the other hand, focuses on the meaning of the graphic, what the data represents and its relationships. Semantics generates the context for the views displayed to the end user and it is important to map connections between different views. The analysis at semantic level requires the association of the syntactic properties of the graph to the data they represent, creating correspondences between context and view.

Syntax and semantics are used to create DynaVis [9], a visualization framework for animation and direct manipulation of data graphics. The research carried out on [9] concluded the use of staged animation provides additional benefits to a visualization and further discouragement of using complex multi-stage transitions. Figure 2.9 is an example of an animation where a minimal and gradual change of the visualization happens. The minimal and gradual change of the visualization allows for the backing data to remain constant but the visualized dimensions to change over time.

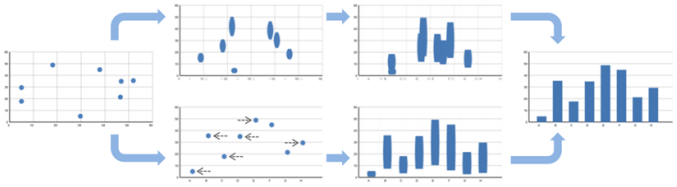


Figure 2.9: Evolution of an animation from a scatterplot to a bar chart using DynaVis [9].

The application of animation techniques come with shortcomings. [31], [32], [33] and [34] are works that enumerate shortcomings. These include how animated transitions attract, first, the attention of the end user, possibly leading to their distraction, how animated transitions require more cognitive workload than purely static visualizations, how their duration can induce lag into the visualization, and how their execution requires a larger pool of computational resources.

Furthermore, when employing animated transitions in a visualization, the number of animated objects present in the visualization should not exceed a certain threshold. Intertwined in all these shortcomings is also the duration of the animation. Depending on how long the animation runs for, the impact of latency, depending on available computing resources, will vary. An animation lasting too little or too long will also impact the attention span of the user but also the consistency of the visualization. Animated transitions should be as fast as possible without making the end user overlook the actual transition [33]. This is something very important in the development of animated transitions and it is usually represented using slow-in, slow-out timings.

The presentation of slow-in, slow-out timings lead us to investigate how, in an animated transition, not only the start and end states matter. The intermediate states are also important, as they allow for tracking the evolution of an animation. The tracking of the evolution of the animation is a technique commonly employed in situations where objects inside a visualization eventually switch location due to the underlying update of the data [35] and switching between different layout methods [36]. If a lot of changes are happening at the same time, represented by objects switching locations inside the visualization, other difficulties may arise, as the end user might get confused on what exactly they should be focusing on. A possible solution is the employing of bundled movement trajectories for a group of objects that have spatial proximity and share similar moving directions - Figure 2.10(a) - explored on [10].

The employing of bundled movement trajectories as a solution involves several principles applied in animated transitions, such as the proximity of objects in a visualization - leading them to be perceived as having similar trajectories [37] - and the perception of a group of objects moving together while being tracked by the end user. If visual cues of grouping are provided, such strategy can help improve people's tracking performance of objects in the visualization [38]. The work on [10] also looks at how distorted trajectories have minor impact on tracking single or multiple moving objects [39] , [40], concluding that bundled trajectories improve tracking precision as the number of objects inside the visualization

(a) Illustration of the movement of five different objects. Dashed and solid lines represent straight and bundled trajectories, respectively.

(b) Complexity metrics for the evaluation of the bundled trajectories.

Figure 2.10: Trajectory bundling [10].

increases. Tracking precision also increases if occlusion or deformation increase in the visualization and that bundled trajectories do not improve the performance of animated transitions when we are discussing simple tracking tasks or when data dispersion is quite high, as the end state has the tendency to leave objects quite spaced out [10]. Metrics used to conclude this are presented in Figure 2.10(b).

Picking up from this discussion on grouping, it has been mentioned how aggregation is a vastly used technique in big data and data streaming visualizations. A survey conducted in 2018 found that aggregation techniques were employed in 74% of the visualizations they profiled [41]. Therefore, it is important the performance of animated transitions is profiled for a a context where aggregation techniques are used. [11] investigates how animated transitions can be used to disambiguate different types of aggregation and communicate the meaning of those operations.

The work on [11] considers a set of eight common aggregation operations: count, sum, maximum (max), minimum (min), arithmetic mean (average), median, standard deviation (stdev), and interquartile range (iqr). The work on [11] is based around the design guidelines established on [9] to adhere to the principles of congruence and apprehension of [42].

The strategies applied to the operations in [11], based on target concept, staging, axis scales and staggering, result in two different designs: a simplified basic design with fewer animation stages, and an elaborate design intended to provide a complete overview of the operation. An example provided is how the count aggregate operation is animated - Figure 2.11. While in both the elaborate and basic design the animation starts by fading out the old axis and then fading in the new one, the intermediate step is different for both. In the basic animation, the intermediate step corresponds to a stacking up of the uniformly spaced out points and occurs in one single step. In the elaborate animation, the intermediate step corresponds to a stacking up executed in three sub-stages, in a way that groups with more points take longer to stack, reinforcing the larger count.

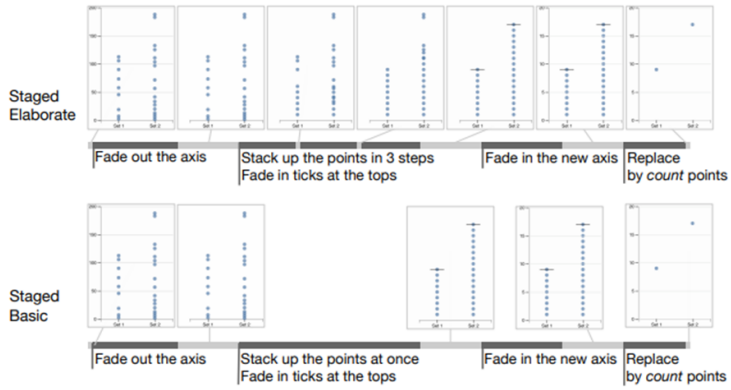


Figure 2.11: Basic and elaborate animated transitions for the count aggregate operation [11]

The study conducted by [11] concluded, for the count aggregate operation, the difference between the elaborate and basic design did not have much of an impact, but this was not the same for all the operations. The elaborated design works better for the average, median, stv and iqr operations while the max and min operations showed better results with the basic design. This showcases how, in animated transitions, the animation chosen for a specific transition is very much connected to the context it is wrapped in.

So far, we have discussed only 2D animated transitions, but 3D animated transitions also exist. In Rolling the Dice [4], a 3D rotation - Figure 2.12 - is used to provide some semantic meaning to the movement of the points inside the scatterplot matrix. The animation, following in the guidelines set on [9], is performed as a three-stage animation, which includes an extrusion into 3D, the actual rotation and, finally, the projection back into 2D.

While a 3D animated transition might take more time to render than a simple 2D one, there are times where it can be used to great effect. An example of this is to increase interaction for the end user in a visualization while also opening the door for the possibility of using Virtual Reality (VR) add-ons for the visualization.

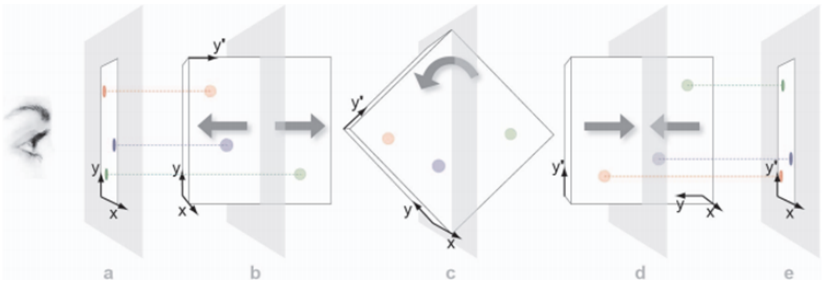


Figure 2.12: Stage-by-stage overview of the 3D rotation employed in Rolling the Dice [4].

2.4 Discussion

Over the last three sub-sections, multiple contributions to the art of visual analytics were presented, with a heavy focus on big data visualizations, data streaming visualizations and animated transitions in visualizations. We now present a summary and discussion of those themes, to serve as an inspiration for the development of our visualization and to find out what strategies can be employed to obtain better results. Table 2.1 summarizes the contributions presented for big data visualizations and data streaming visualizations, according to several criteria.

Most of the contributions that were presented in section 2.1, section 2.2 and section 2.3 share common characteristics. All the big data visualizations contributions are prepared to receive and handle big volumes of data. All the data streaming visualizations contributions are capable of processing real time data. In both big data visualizations and data streaming visualizations, the handling of the data is made easier by employing dimensionality reduction techniques. This is something we observed in some of the contributions we analyzed for big data visualizations and in all of data streaming visualizations contributions. Dimensionality reduction consists in the transforming of high dimensional data into a meaningful representation of reduced dimensionality [43]. In big data visualizations, the use of dimensionality reduction facilitates classification, visualization, and compression of high-dimensional data [43], while also improving the extendibility of the visualization.

A good example of the application of dimensionality reduction techniques is in MAP-Vis [1], as it transforms aggregated values into discrete key values for effective storage and retrieval. An even more noticeable example of the application of dimensionality reduction techniques comes up in Colorized Mosaic Matrix [5], as it can be easily adapted to display various types of data when necessary, and in Rolling the Dice [4], whose scatterplot matrix can be easily extendable to support and display multiples types of data (high-dimensional or not).

In streaming data visualizations, similarly to big data visualizations, dimensionality reduction techniques are used to facilitate data handling. They aid in processing continuous amounts of data that arrive to the visualization by selecting and extracting desired features from the data [44]. An example of this can be found in Unveil [6], in the signal extraction. The extracted locations are then aggregated with other nearby locations to create a dot that represents them. VAStream [7] employs a similar technique in its analysis of social media streams, leading us to conclude feature extraction is more desirable for contributions that handle data streaming flows.

	Contribs.	Data Type	Big Volume	Big Velocity	Dim. Reduction	Real Time	Animation
Big Data Visualizations	[21]	Determined by user	X	-	-	-	-
	[1]	Multi-set high dimensional	X	-	X	-	-
	[2]	Categorical, Numerical	X	-	-	-	-
	[3]	Categorical, Numerical, Time Series	X	-	X	-	-
	[4]	Time Series	X	-	X	-	X
	[5]	Categorical	X	-	X	-	-
Data Streaming Visualizations	[23]	Framework					
	[6]	Flow Nature	-	-	X	X	-
	[7]	Flow Nature	-	-	X	X	-
	[25]	Time Series	-	-	X	X	-
	[8]	Time Series	X	-	X	X	-

Table 2.1: Contributions presented for big data visualizations and data streaming visualizations.

Despite the benefits of applying dimensionality reduction, it has some drawbacks. Firstly, when using dimensionality reduction techniques in a visualization, it is necessary to guarantee context and information loss is minimal throughout the visualization. This is where techniques like aggregation and simplification come in. The first one is particularly useful, being used predominantly in the contributions we presented - particularly on the contribution that looked to analyze the Shanghai Metro Network [3], where the data requires heavy pre-processing to find the necessary connections between the different data types available in the data set. The use of aggregation and simplification techniques also help with maintaining the context of information, but they do not fix entirely the problem of loss of information.

Another drawback of employing dimensionality reduction is its processes are not cheap – they require a fair bit of computational resources. Because of its requirement for a fair bit of computational power, the employment of dimensionality reduction techniques can lead to an increase of latency within the visualization. A possible solution to this problem is provided on I2 [25]. It connects distributed data analysis programs with the visualization of results in a way that only currently depicted data points are processed and transferred. It is possible to analyze, in Table 2.1, that all contributions struggled to display the data in an efficient way, revealing very little input in terms of velocity.

When we discuss creating a big data streaming visualization, the concepts mentioned above – avoiding loss of context, minimal information loss and efficient display of data - gain additional importance. They become near requirements to enhance the quality of the visualization but their deployment can end up becoming a hurdle for the performance of the visualization. It is then necessary to create an efficient visualization regarding its performance, but a visualization also capable of adapting itself to the

Contribs.	Object Tracking	Staged Animation	Transitions
[6]	-	X	Horizontal
[9]	X	-	Vertical
[10]	X	-	Vertical
[11]	-	-	Vertical
[4]	-	-	Vertical

Table 2.2: Contributions presented for animated transitions in visualizations.

ever-changing data arriving to it. In other words, a good visualization not only looks into the concepts of big data and streaming, but also to the concept of performance - both hardware and software. A visualization that holds performance as one of its main keys for success can be dubbed a performance visualization [18]. A visualization that has flexibility implemented and can adapt itself to the changes triggered by arriving data needs to be a performance visualization by default, in order to have minimal performance loss during the representation of changes happening within the data set in the visualization to the end user. A way to achieve minimal loss of performance in a visualization is through the application of smooth and gradual transitions via intermediate visual idioms [23]. Smooth and gradual transitions can exist in the form of animated transitions.

Animated transitions are used to improve the overall experience of a visualization and to represent changes that occur within the visualization. These changes can occur in terms of data, of visual idiom or in terms of level of detail being displayed in the visualization. Animated transitions can be used to avoid loss of context – as they are used to represent changes in the visualization – and to aid with possible space limitations in a visualization. An example of this is found in Rolling the Dice [4], where a 3D staged animation provides additional semantic meaning to the movement of points inside the scatterplot matrix, helping maintaining the context between the two states of the visualization. Table 2.2 summarizes the content of each contribution on the topic of animated transitions in visualizations, according to several criteria.

Most of the contributions presented for animated transitions employ object tracking and/or staged animations. Object tracking is an algorithm for tracking the displacement of one or several objects inside a certain scene (in this case, the scene in question is a visualization). By animating trajectories, changes occurring within the visualization become more dynamic and easier to identify and understand. A good example of this is the work on [10], which concluded bundled trajectories improve tracking precision as the number of objects inside the visualization increase. Bundled trajectories are a good strategy to use for complex tracking strategies. Complex tracking strategies are usually connected to clusters of data, a technique commonly employed in the analysis of big data.

Staged animation, on the other hand, corresponds to one of the twelve basic principles of animation and its goal is to direct the attention of the end user to what is important within the visualization. It is important to employ staged animation in dense networks and in visualizations where changes occur frequently, as it produces fewer mistakes when compared with other techniques. Staged animation can

also be an efficient way to display aggregation operations [11], and therefore, can be of good use in big data streaming visualizations, as we have established aggregation techniques and operations are commonly used in those type of visualizations.

In Table 2.2, it can also be observed most contributions presented for animated transitions in visualizations use vertical transitions. A vertical transition is when transitions happen inside the same module of the visualization when there is need to visualize or analyze data of a module from another perspective but still within the same time span. The staged animation employed in Rolling the Dice [4] is an example of this. On the other hand, if a visualization is made up of multiple modules - where each module represents information belonging to a different time span - horizontal transitions are used from one visual idiom of a module to the visual idiom of the next module. An example of horizontal transitions can be found in [6], ensuring smooth transitions between different views (modules) within a UI. Both vertical and horizontal transitions aid with maintaining context of the visualization and reducing loss of information.

Taking into consideration the advantages and disadvantages of each contribution, we can conclude that none of them is prepared to meet our defined objective - create a big data streaming visualization able to display data across several visual idioms with minimal loss of context and with minimal loss of information across different time spans while maintaining a consistent and linear performance for different data flow rates.

The closest contribution we found to our defined objective is VisMillion [8], as it is prepared to serve big data and real time streaming data. In spite of that, VisMillion [8] is lacking in mechanisms to avoid loss of context and information within the visualization - such as animated transitions - and in performance mechanisms that allow for efficiency in displaying the data in a comprehensive way to the end user. VisMillion and Change [13] is a visualization developed with the aim of fixing some of the gaps existing in VisMillion [8]. VisMillion and Change [13] explores horizontal transitions between the modules of VisMillion [8] to reduce loss of context across the visualization. While this helps in avoiding loss of context between the different modules of the visualization, it does not take into consideration possible performance loss for the visualization across time.

Our visualization, **TimeWarp**, aims to fix the missing gap of performance in VisMillion and Change [13] by developing efficient ways of displaying data to the end user, while making sure context is not lost across the visualization and information loss is still minimal.

3

TimeWarp: The Prototype

Contents

In this section, the concept behind our prototype - TimeWarp - will be explored. The exploration of our prototype will be split into two different steps. Firstly, it will be presented a brief history of the VisMillion [45] concept, followed by presenting the process of migrating VisMillion and Change [13] to Three.js to improve overall performance of our prototype. Secondly, the concepts and elements resulting from the migration will be presented, with a focus on horizontal transitions and visualization of quantitative data on our prototype.

3.1 VisMillion, the concept

The concept of VisMillion [45] has the objective of allowing visualization of large quantities of data in real time represented across different modules able to complement each other for creating a cohesive and consistent visualization. Each module represents data across a different time span. In each time span, the time stamp of the data matters. Depending if data is newer or older, the representation for that particular piece of data can change. If the data is newer, a representation with a greater level of detail is used. If the data is older, a representation with less level of detail is used. This is the concept of graceful degradation. These visual representations of data happen in the form of different visual idioms that exist in the visualization.

The concept of VisMillion was enhanced firstly in [8] and then further enhanced in VisMillion and Change [13], where horizontal transitions were explored with the goal of reducing loss of context between different modules in the visualization, and in FastViz [46], where vertical transitions were designed to be implemented into the concept in order to avoid loss of context and loss of information inside each module. Our prototype is based on the VisMillion [45] concept and visualization [8] while carrying on the work done on VisMillion and Change [13] and FastViz [46].

3.2 Migration of VisMillion and Change

Migration is an operation that can be done at different levels. At its lowest levels, migration takes the form of transforming a specific piece of code - it can be a full program or not - from one language into another [47] (it can be a newer version of the language or a totally different one). In higher levels, migration can include changes to the architecture of a system [47] if the new system requirements deem it necessary. In all, the process of a **migration** can be arbitrarily understood as the movement of code into a new platform and/or programming language [47].

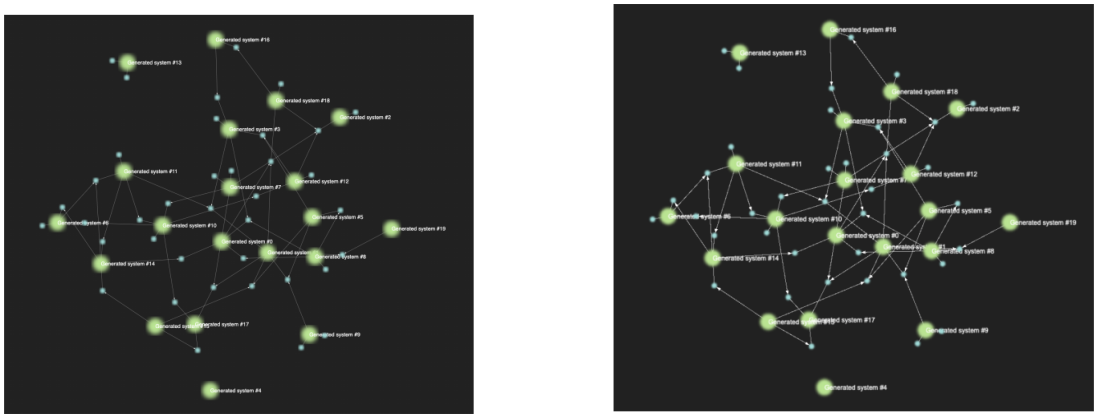
In the case of our prototype, the type of migration to be employed is a relatively low level one and it consists of migrating VisMillion and Change [13], implemented in D3.js[1], to Three.js [2]. While D3.js

[1] https://d3js.org/. D3.js is a JS library for producing dynamic, interactive data visualisations in web browsers.

[2] https://threejs.org/. Three.js is a cross-browser JS library and application programming interface used to create and display animated 3D computer graphics in a web browser using WebGL.

is a JS library for manipulating documents based on data using Hypertext Markup Language (HTML), Scalable Vector Graphics (SVG), and Cascading Style Sheets (CSS), Three.js is a JS 3D library that renders with WebGL, allowing it to make use of a computer's GPU while hiding its details of rendering and modelling [48].

The use of WebGL to render the Three.js code allows for a theoretical increase in performance when compared to a visualization implemented with D3.js, as Three.js uses WebGL's simplified pipeline model (a basic OpenGL[3] pipeline). When the use of WebGL's simplified pipeline model is combined with WebGL's ability to make use of a computer's GPU, the end result is a more efficient rendering of a visualization, as WebGL allows for the rendering of complex 2D scenes with the use of less resources [49] compared to a visualization implemented with D3.js, as it is the case of VisMillion and Change [13].



(a) A snapshot of the visualization module developed in D3.js using SVG

(b) Snapshot of the visualization module developed in PixiJS.

Figure 3.1: Visualizations developed to evaluate performances of different JS rendering frameworks [12].

Further proof of the advantages of using a WebGL based JS library is displayed on [12], where various JS rendering frameworks were tested for performance evaluation. In its comparison between a visualization that was created based on D3.js - Figure 3.1(a) - and another created using PixiJS[4] - Figure 3.1(b), the visualization implemented with D3.js performed on pair with the one implemented with PixiJS when the number of elements in the visualization was still relatively low. When the number of elements increases, "the performance suffers drastically" [12]. Is is also worth noting the observations done on [12] comes while not fully extracting the potential of a WebGL based JS library, as the performance tests were run on a laptop without a dedicated graphics card.

[3]https://www.opengl.org/. OpenGL is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics.

[4]https://pixijs.com/. PixiJS is a rendering library that will allow you to create rich, interactive graphics, cross platform applications, and games without having to dive into the WebGL API or deal with browser and device compatibility.

For VisMillion and Change [13], the migration process is part of finding out if the same conclusions on performance between D3.js and WebGL found on [12] can be applied to our prototype. In other words, the migration of VisMillion and Change [13] from D3.js to Three.js is happening to find out if Three.js offers any significant consistent performance improvements when met with a continuous stream of big data while keeping a minimal loss of context and information across different time spans. To better understand the differences between VisMillion and Change [13] and TimeWarp, the concepts and elements that constitute each visualization need to be analyzed. During the migration process of VisMillion and Change [13] to Three.js, its architecture, concepts and elements were kept as close as possible to its original logic. To better understand the constitution of our prototype, its concepts and elements will be analyzed.

3.3 TimeWarp Architecture

The architecture employed in TimeWarp - fig. 3.2 follows the same structure as the architecture of VisMillion and Change [13]. Real time data streaming is simulated through data packages generated and sent by a data flow generator - Streamer, a Python[5] script. Streamer sends packets of data to be processed by our visualization. The arrival of new packages might cause changes to the visualization. Task calls and generated data flow are received by the modules that make up our visualization.

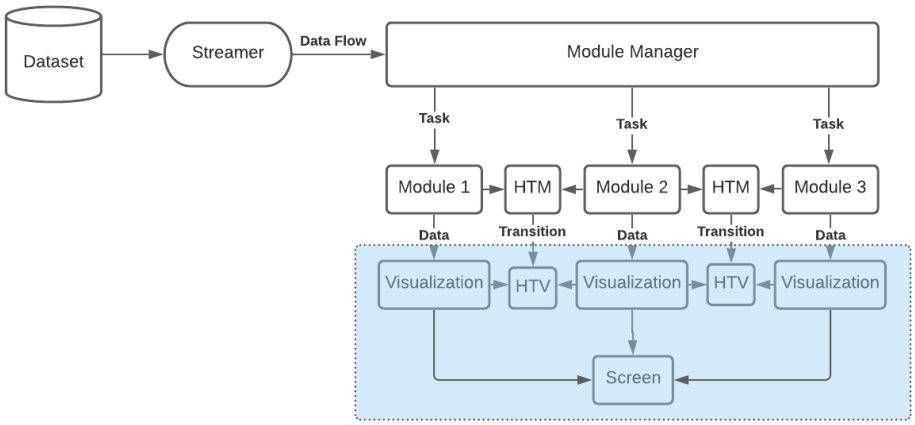


Figure 3.2: Architecture diagram of TimeWarp. Research focus is highlighted.

Each module consists of several methods that, when called upon, transfer information to their respective visual idiom. Visual idioms receive the necessary instructions to produce the visualization. This separation allows modules to work independently. The concept of graceful degradation allows them to be linked across the time span of our visualization, contributing to the cohesiveness and consistency of

[5]Python is an interpreted high-level general-purpose programming language.

it.

The connection between the three modules is performed via another module: Horizontal Transitions Module (HTM). The HTM is attached to a visualization - the Horizontal Transition Visualization (HTV). Since a horizontal transition happens between two different modules, the HTM is always connected to two modules and contains the horizontal transition techniques implemented in our visualization. HTM also guarantees operations such as data aggregation, dimensionality reduction and statistical measures are performed during the actual horizontal transition.

The focus of our research is highlighted with light blue in fig. 3.2.

3.4 TimeWarp Interface

The TimeWarp interface - Figure 3.3 - follows the same basic principle of the VisMillion and Change [13] interface - Figure 3.4 - simple, yet functional visualization, composed by several modules. In our prototype, a module can holster either a visual idiom or an animated transition - a horizontal transition, in this case - depending on its position in the time span. Figure 3.3 displays three modules, each displaying a different visual idiom. This is the same concept as on VisMillion and Change [13], where each module also has a visual idiom assigned to it during the course of its run time.

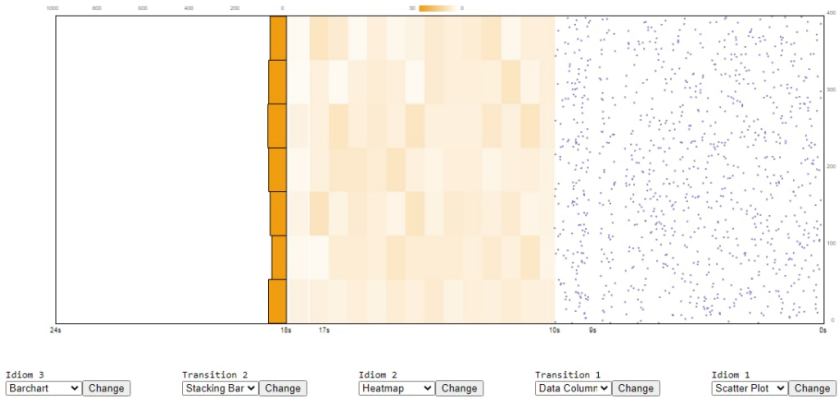


Figure 3.3: TimeWarp interface.

Similarly to VisMillion and Change [13], our prototype follows the same structure for displaying modules. Modules are displayed in a horizontal fashion, with the start of the visualization on the right side of the interface and the end of the visualization on the left side of the interface.

By taking into consideration the time span of our prototype time span starts on the right, it can also be compared how our prototype uses the concept of graceful degradation in the same way as VisMillion and Change [13]. From the right to the left, in both Figure 3.3 and Figure 3.4, it can be observed how the level of detail in each module progressively decreases as we move further from the start of the visualization, as each visual idiom shows more and more aggregated data. Operations of aggregation and filtering are used to apply the graceful degradation concept in our prototype, aiding also in creating a consistent and easy to read visualization across different time spans. The operations of aggregation and filtering applied depend on the visual idiom represented within each module.

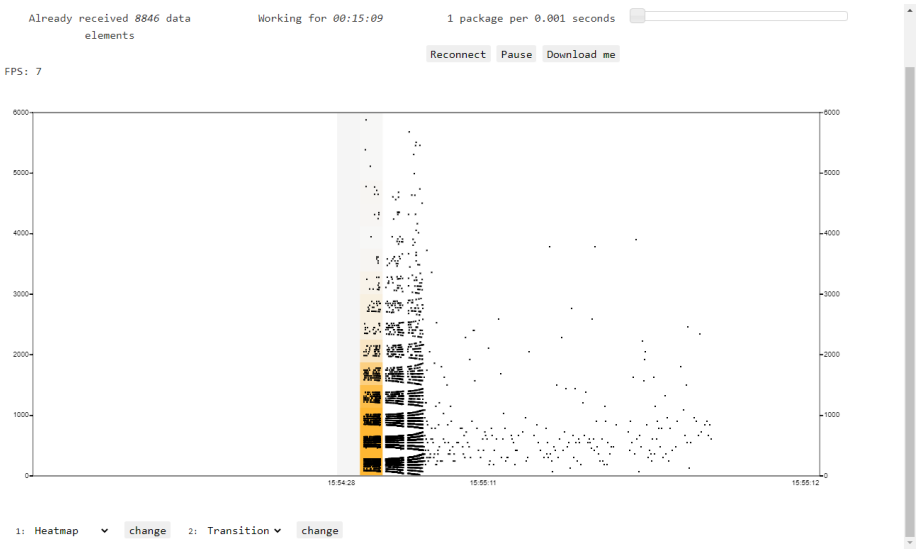
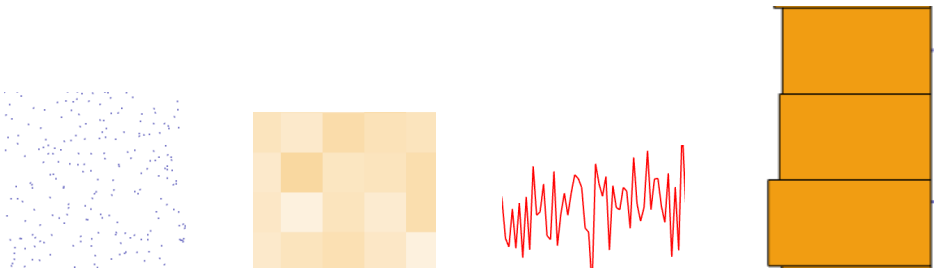


Figure 3.4: VisMillion and Change [13] interface.

3.5 Visual Idioms

In order to have a visualization able to display data across different time spans while employing the graceful degradation concept, our prototype has various visual idioms that can be displayed across its modules. Each visual idiom allows for the display of data with a different level of aggregation and detail.

All the visual idioms of VisMillion and Change [13] also exist in our prototype, having been migrated from D3.js to Three.js. The logic of implementation of all visual idioms on TimeWarp was kept as similar as possible to the visual idioms of VisMillion and Change [13] in order to compare performances between the two visualizations. Horizontal transitions occur between these visual idioms when they are assigned to a specific module. Figure 3.5 showcases the visual idioms of our prototype. These visual idioms - scatterplot, heatmap, linechart and barchart - are to be explored with further detail in the sections below.



(a) Scatterplot (b) Heatmap (c) Linechart (d) Barchart

Figure 3.5: Visual idioms of TimeWarp.

3.5.1 Scatterplot

In a general fashion, a scatterplot - Figure 3.5(a) - allows for correlating two variables within a data set with the aid of dots positioned between the horizontal and vertical axis. Each dot is positioned according to its value, obtained through the correlation operation between the two variables represented in the scatterplot.

In a similar fashion to the scatterplot (named scatterchart) in VisMillion and Change [13], the scatterplot in TimeWarp is able to display data in its original form - in real time, without any type of aggregation or simplification techniques applied to the data. Because of this, its natural to find the scatterplot as the visual idiom being displayed in the first module of our prototype, helping to visualize the freshest data that has arrived to our prototype, according to the concept of graceful degradation.

3.5.2 Heatmap

The heatmap - Figure 3.5(b) - represents a colorized matrix which represents the relationship between two different variables. In a similar way to the scatterplot, a heatmap is able to display correlations between two different variables of a data set. How it does it, however, is different. While a scatterplot does a simple correlation, the correlation performed by the heatmap involves aggregating data. The result of the aggregation operation will be converted into a hue value to be represented within each entry of the colorized matrix. A higher hue - resulting in a more intense color - is a consequence of more values getting aggregated.

The heatmap in TimeWarp, as it requires aggregation of values, can be visualized in later modules, as it shows data with less detail, obeying to the concept of graceful degradation. Besides allowing for the visualization of correlations, a heatmap visual idiom allows for the visualization of flow and/or volume of a data set, the change of value ranges within a data set and the existence of patterns within a data set.

3.5.3 Linechart

The linechart visual idiom - Figure 3.5(c) - represents a series of connected dots to form a line, providing the motion of continuity to the data flow. The linechart visual idiom can be used to represent upward and downward trends in data sets and tracking of possible existing patterns in data sets.

Similarly to VisMillion and Change [13] , the linechart in TimeWarp is used to represent aggregated data split in time intervals in a way that the the average of each aggregation is visualized in the visual idiom.

3.5.4 Barchart

A barchart visual idiom - Figure 3.5(d) - represents aggregated data across several categories or time intervals. Data is represented with the use of rectangular bars that vary its height or width (in accordance to how the barchart is positioned) in proportion to the aggregated values it represents. The barchart visual idiom can be used to compare intervals and to check frequency distribution across the data set.

Similar to the implementation of VisMillion and Change [13], the barchart in TimeWarp is displayed horizontally, in accordance with the rest of the visualization. The height of the bars grows to the left, in accordance with the data flow of the visualization. As the barchart represents an aggregation of values, it can only be visualized in the last module of the visualization in order to verify the concept of graceful degradation.

3.6 Horizontal Transitions

In order for the concept of graceful degradation to work properly in TimeWarp, visual elements need to be displayed to the user as visual clues to the interactions happening in our prototype between modules. In TimeWarp, the visual elements performing this job are horizontal transitions. Horizontal transitions occur between two different visual idioms, each one existing within its own module. Horizontal transitions not only allow for the application of the graceful degradation concept in our prototype but also allow for minimal loss of context and information across different time spans - two of the objectives attached to the creation of TimeWarp.

Regarding how horizontal transitions can reduce loss of context across different time spans, the answer comes from how the concept of graceful degradation is applied in our prototype. Similarly to VisMillion and Change [13], as the visualization progresses and data gets aggregated, modules should have a larger time span to represent data in order to minimize loss of information in the visualization. This is how horizontal transitions can aid with minimizing loss of information.

	Heatmap	Linechart	Barchart
Scatterplot	Fade In-Fade Out	Data Column	Plot Lines

Table 3.1: Horizontal transitions implemented on TimeWarp, based on the work VisMillion and Change [13].

The implementation of horizontal transitions in *TimeWarp* is part of the migration process from D3.js to Three.js. Therefore, they follow the same structure as the one presented on VisMillion and Change [13], with the scatterplot as the first visual idiom of the visualization - positioned on the first module - and the remainining visual idioms - heatmap, linechart and barchart - are considered the second visual idiom, as they display data with lower detail due to the aggregation technique of graceful degradation applied in our visualization. The animated transitions, as shown in Table 3.1, occur from the scatterplot to the second visual idiom.

The horizontal transitions implemented in our prototype - specified in table 3.1 were the transitions with the best results for user testing in the user evaluation performed in [13]. The horizontal transitions who achieved the best results in [13] were Fade-In and Fade-Out for scatterplot to heatmap, Data Column for scatterplot to linechart and Plot Lines for scatterplot to barchart.

All the implemented transitions obey to the principles determined on [9], as they are simple, essentially done in one single stage in order for them to be easy to perceive and understand on behalf of the end user. The implemented horizontal transitions in our prototype are to be explored with further detail in the sections below.

3.6.1 Scatterplot to Heatmap

In the Data Column horizontal transition - fig. 3.6 - the dots of the scatterplot move into a rectangle corresponding to the time interval of the transition. When inside the rectangle, the value of each dot gets moved into an accumulator. The value of each accumulator is then transformed into a hexadecimal color value to fill out the rectangle. The calculated value for the colour inside the HTM will be the same as the colour value calculated within the module that holds the heatmap, allowing for a smoothing out of the transition.

It is worth nothing the horizontal in our visualization slightly differs from the approach of [13]. In VisMillion and Change [13], the horizontal transition between scatterplot and heatmap has the dots of the scatterplot moving across the visualization to form vertical columns. The vertical columns, as more dots get accumulated, will increase or decrease the hue of the rectangle according to the accumulation - fig. 3.7. After the accumulation process is done, the dots will fade out of view to the end user and the rectangle moves to the next module.

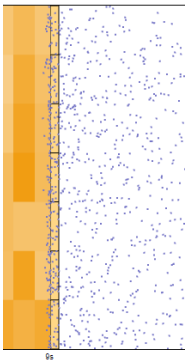


Figure 3.6: Scatterplot to heatmap horizontal transition.

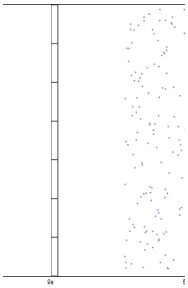


Figure 3.7: Rectangles for aggregation of dots in TimeWarp

.

3.6.2 Scatterplot to Linechart

In VisMillion and Change [13], a similar performance was presented between Fade-In and Fade-Out and Funnil horizontal transitions. For TimeWarp, we chose to implement the Funnil approach - fig. 3.8 - due to it being simpler to implement with Three.js.

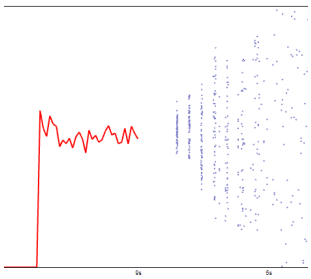


Figure 3.8: Scatterplot to linechart horizontal transition.

.

In the transition between scatterplot and linechart, the points of the scatterplot converge to the initial point of the linechart. For each group of dots, they converge with small time gaps between each other into the beginning of the linechart, allowing for a smooth transition between modules.

3.6.3 Scatterplot to Barchart

Each bar of the barchart is incremented with the transition of the data points of the scatterplot for each time interval. As the barchart serves the purpose of an accumulator in the last module of the visualization - as determined by the application of graceful degradation in our visualization - the more data it gets accumulated in a bar, the wider the bar will be.

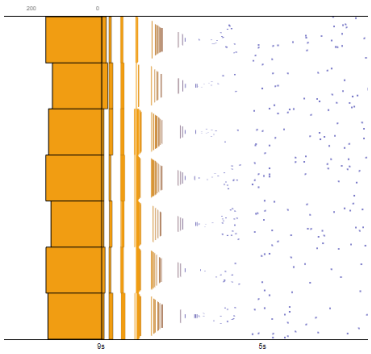


Figure 3.9: Scatterplot to barchart horizontal transition.

The transition between scatterplot and barchart - fig. 3.9 - begins with the points of the scatterplot converging into an area where lines will slowly expand into rectangles that match the height of each bar - fig. 3.10. Those rectangles will be used to increment the width of the bars of the barchart.

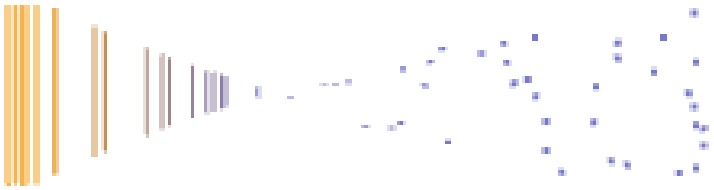


Figure 3.10: Expansion of points to rectangles in the transition between scatterplot and barchart.

3.7 Performance considerations for TimeWarp

In order to try to improve further the performance of our prototype, some performance considerations were implented for additional testing during the migration of the visualization from D3.js to Three.js. The performance consideration implemented in TimeWarp a Three.js object called Instanced Mesh[6], a special version of Mesh[7] with instanced rendering support. The usage of of Instanced Mesh also helps to reduce the number of draw calls for the pipeline.

[6] https://threejs.org/docs/api/en/objects/InstancedMesh
[7] https://threejs.org/docs/api/en/objects/Mesh

The use of Instanced Mesh is applied to two visual idioms in *TimeWarp*: scatterplot and heatmap. The reason to only implement Instanced Mesh for these two visual idioms is these two can possibly gain considerable performance improvements from using Instanced Mesh, as both idioms require rendering a large number of objects with the same geometry and material.

Is is worth nothing that, while Instanced Mesh can possibly offer performance gains, it also brings some drawbacks. It removes some of the possibilities and operations that can be performed during execution with Three.js, as it groups several objects into one. Therefore, it does not allow for individual manipulation of objects (or group of objects) and their properties, like their colour, opacity or size during execution, making it harder to create animated transitions between visual idioms as many animated transitions use those manipulations to showcase to the end user a certain meaning behind the transition.

The existence of these drawbacks is why heatmap and scatterplot were implemented with and without Instanced Mesh, in order to test if the performance gains of Instanced Mesh were enough to justify its drawbacks.

4

Prototype Evaluation

Contents

In this chapter, we approach, in detail, the methodology behind the performance tests performed for the evaluation of TimeWarp. The evaluation of our prototype is compromised of several performance tests meant out to measure the performance of the prototype regarding its stability, scalability, the improvements offered by the migration of D3.js to Three.js and the improvements offered by the performance considerations specified in section 3.7. Conclusions on the performance of our prototype will be based on the analysis of results obtained during the performance testing.

The performance testing of TimeWarp was conducted using the Google Chrome browser (version 94.0.4606.81 64 bits) installed in laptop with Windows 10 Pro as its operative system, a Intel(R) Core(TM) i5-4210U CPU @ (1.70 GHz 2.40 GHz) CPU, 6GB of RAM memory and a NVIDIA GeForce 820M with 2GB of dedicated memory in a screen with resolution of 1366x768.

4.1 Performance Tests

There are three set of tests to be done with our prototype. Firstly, the performance considerations specified in section 3.7 will be tested, with a comparison between the performance of the implementations of heatmap and scatterplot with and without the use of Instanced Mesh to verify (Dots vs Instanced Mesh). This set of tests is done to test if the performance gains of Instanced Mesh were enough to justify its drawbacks. Secondly, a second set of tests will be done to check performance of our prototype with the horizontal transitions between visual idioms occurring. Thirdly, the final set of tests have the goal of comparing performance of VisMillion and Change [13] - written in D3.js - with the performance of our prototype - written in Three.js - to verify the performance gains obtained with the use of WebGL's simplified pipeline model and its ability to make use of a computer's GPU to render out a visualization (D3.js vs Three.js).

4.2 Performance Tests Metrics

For each performance test, several metrics were recorded in order to evaluate our prototype. The main metrics recorded were the number of Frames per Second (FPS) of the visualization during each test and data flow value being sent to our prototype. The value of FPS means how many still images are played in each second to the user. The higher and the more consistent the FPS values across a time span, the better, as it means our brain will perceive things as a smooth motion. The data flow value indicates how much data is being sent, per second, to be processed by our visualization.

From the number of FPS, some other metrics can be calculated, to be used in the evaluation of our prototype. These metrics include the average number of FPS during execution, the minimum and maximum value of FPS achieved during the test and the variance value of FPS for each test.

4.3 Performance Tests Methodology

For the three set of tests performed with our prototype, the number of FPS were recorded in intervals of ten seconds. The number of FPS were calculated with the inverse of the difference between the current time and time of the last computed. Every ten seconds, when FPS were calculated, the value was saved to a .csv file, which was then processed to calculate the average, minimum, maximum and variance values for FPS.

In all three set of tests, data was sent to TimeWarp through data packets generated by a Python script. The data was then processed into data bins according to its time stamp of arrival to the visualization before being sent to the first module to be displayed with the correct visual idiom. For the first two set of tests, four different data flow values were tested: 10, 100, 1000 and 10000 points per second. The variation of data flow value was performed directly on the Python script. For the final set of tests, the single data flow tested was of 10000 points per second in order to see how our prototype and VisMillion and Change [13] performed in a worst case scenario.

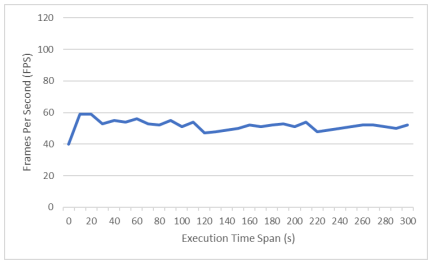
All the tests were performed across a time interval of 5 minutes (300 seconds). In the end, graphics were created to correlate the evolution of FPS across the five minutes of execution for each test, while tables were created to present data flow values with each corresponding average number of FPS and its minimum, maximum and variance values of FPS for each test. The analysis of each performance test combines the analysis of the graphics and tables presented.

4.4 Visual Idioms

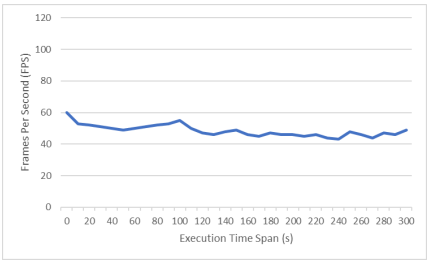
4.4.1 Dots vs Instanced Mesh

In order to test how the performance considerations mentioned in section 3.7 impact the performance of TimeWarp, heatmap and scatterplot were tested with (Instanced Mesh) and without (Dots) the performance considerations implemented. For each test, they were tested for four different data flow values and the FPS were registered during the execution of each test, with then metrics being calculated from the registered values.

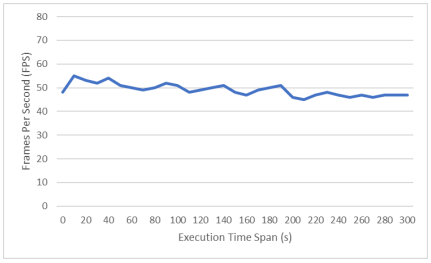
Independently of the data flow, the test comparison between heatmap with - Figure 4.2 - and without - Figure 4.1 - Instanced Mesh implemented achieved relatively similar results in terms of consistent FPS values. Despite similarities, the FPS values of heatmap without Instanced Mesh - table 4.1 - are slightly lower compared to heatmap with Instanced Mesh implemented - table 4.2.



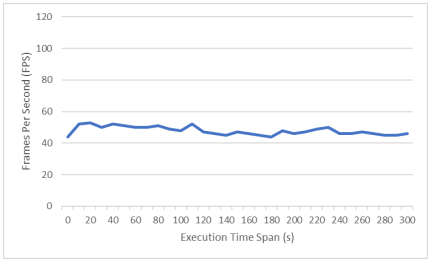
(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.



(c) Data flow of 1000 points per second.



(d) Data flow of 10000 pints per second.

Figure 4.1: FPS of TimeWarp with heatmap (Dots) as its only visual idiom.

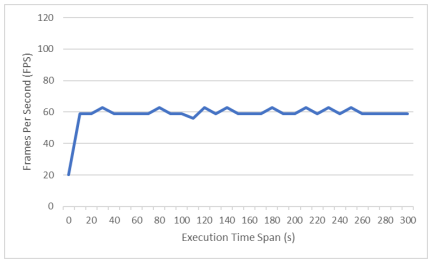
Data Flow/Metrics	Avg. FPS	Min. FPS	Max. FPS	FPS Variance
10 points per second	51,710	40	50	12,271
100 points per second	49,065	45	55	6,125
1000 points per second	49,065	45	55	6,125
10000 points per second	47,839	44	53	6,716

Table 4.1: Performance metrics of TimeWarp with heatmap (Dots) as its only visual idiom.

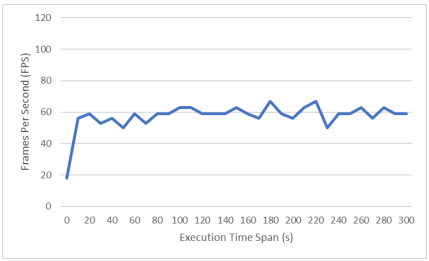
Data Flow/Metrics	Avg. FPS	Min. FPS	Max. FPS	FPS Variance
10 points per second	58,677	20	63	53,380
100 points per second	57,516	18	67	68,250
1000 points per second	55,258	19	67	59,804
10000 points per second	60	16	111	159,871

Table 4.2: Performance metrics of TimeWarp with heatmap (Instanced Mesh) as its only visual idiom.

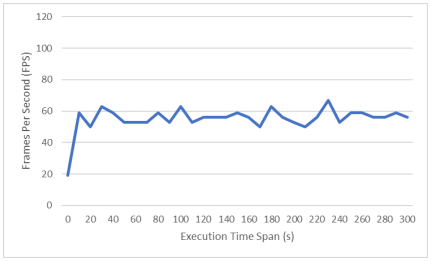
Not only that, but as the value of the data flow increases across tests, the number of average FPS across each test execution tends to decrease in a bigger fashion in the implementation with no Instanced Mesh, as evidenced on tables table 4.1 and table 4.2. The increase in the average value of FPS in the test with data flow of 10000 points per second (fig. 4.2(d)), contrary to the tendency, can be justified by the anomalous value of 111 FPS reached during the execution of the test. The anomalous value is also the cause of the bigger variance value of that test when compared to the other three.



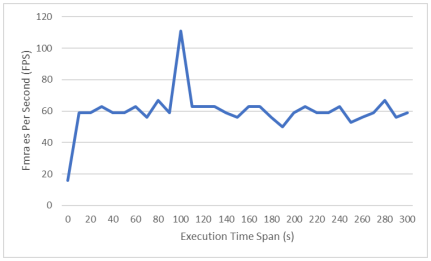
(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.



(c) Data flow of 1000 points per second.

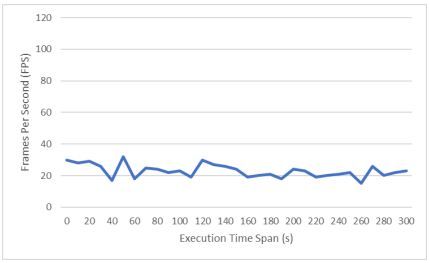


(d) Data flow of 10000 pints per second.

Figure 4.2: FPS of TimeWarp with heatmap (Instanced Mesh) as its only visual idiom.

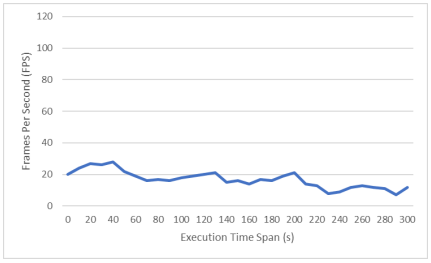
Contrary to the minimal difference found between the two versions of heatmap, the differences in performance between scatterplot implemented with performance considerations - fig. 4.4 - and scatterplot implemented without performance considerations - fig. 4.3 - is more noticeable.

(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.

(c) Data flow of 1000 points per second.



(d) Data flow of 10000 pints per second.

Figure 4.3: FPS of TimeWarp with scatterplot (Dots) as its only visual idiom.

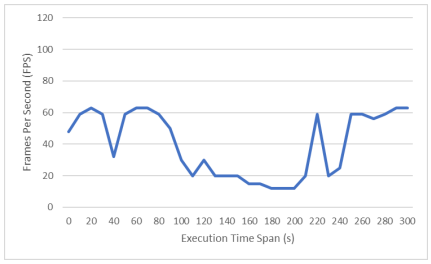
Data Flow/Metrics	Avg. FPS	Min. FPS	Max. FPS	FPS Variance
10 points per second	22,677	9	42	58,089
100 points per second	23	15	32	16,977
1000 points per second	19,613	12	30	28,556
10000 points per second	16,839	7	28	27,490

Table 4.3: Performance metrics of TimeWarp with scatterplot (Dots) as its only visual idiom.

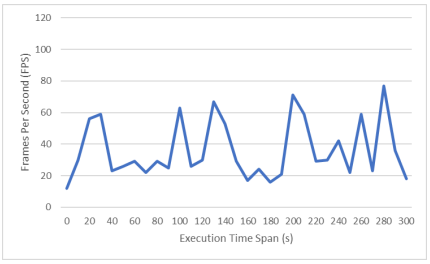
Data Flow/Metrics	Avg. FPS	Min. FPS	Max. FPS	FPS Variance
10 points per second	40,129	12	63	401,403
100 points per second	36,226	12	77	331,981
1000 points per second	42,354	14	125	717,455
10000 points per second	35,226	17	77	263,723

Table 4.4: Performance metrics of TimeWarp with scatterplot (Instanced Mesh) as its only visual idiom.

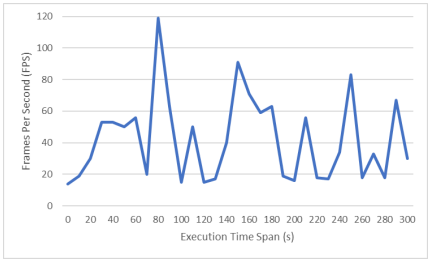
The peak FPS and the average FPS for each test with Instanced Mesh implemented is higher than the corresponding test with a simple Dots implementation, as observed in tables table 4.3 and table 4.4. For both situations, however, as the data flow value increases, the FPS drop becomes more noticeable as the execution approaches the five minutes mark, although that drop is more pronounced in the tests with a simple Dots implementation.



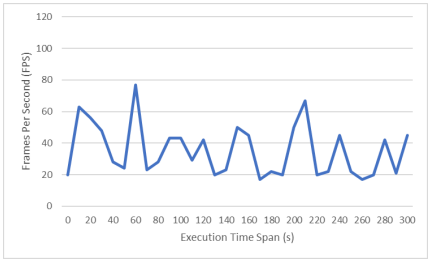
(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.



(c) Data flow of 1000 points per second.



(d) Data flow of 10000 pints per second.

Figure 4.4: FPS of TimeWarp with scatterplot (Instanced Mesh) as its only visual idiom.

4.4.2 D3.js vs Three.js

In order to verify the success of migrating the visual idioms of VisMillion and Change [13] from D3.js to Three.js, a comparison of the performances of VisMillion and Change and our prototype occurred. To compare performances, tests were run for a data flow of 10000 points per second during an execution

Visualization/Metrics	Avg. FPS	Min. FPS	Max. FPS	FPS Variance
VisMillion and Change	38,226	23	46	32,626
TimeWarp	60	16	111	159,871

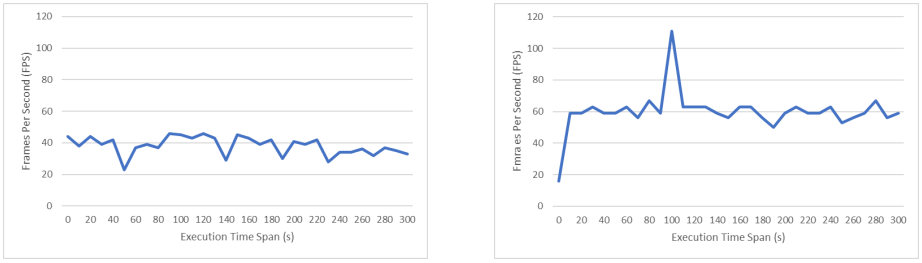
Table 4.5: Performance metrics of VisMillion and Change [13] TimeWarp for heatmap with data flow of 10000 points per second.

Visualization/Metrics	Avg. FPS	Min. FPS	Max. FPS	FPS Variance
VisMillion and Change	50,903	34	60	22,991
TimeWarp	35,226	17	77	263,723

Table 4.6: Performance metrics of *VisMillion and Change* **[Pereira (2019)]** and *TimeWarp* for scatterplot with data flow of 10000 points per second.

time span of five minutes, where FPS values were registered and metrics were calculated for each test. The two visual idioms compared were the two visual idioms - heatmap and scatterplot - tested in the section above with performance considerations implemented (Instanced Mesh).

When comparing the performance of heatmap in VisMillion and Change [13] with the performance of heatmap in our prototype, with Instanced Mesh implemented, it is noticeable how the FPS of TimeWarp stay relatively consistent across the execution length, while also achieving a higher value of average FPS during execution of the test compared to VisMillion and Change [13], as shown in table 4.5.

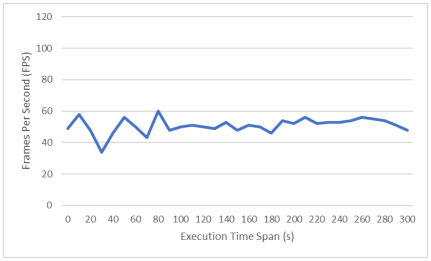


(a) VisMillion and Change **[Pereira (2019)]**

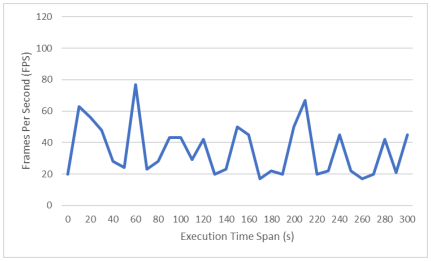
(b) TimeWarp

Figure 4.5: FPS of VisMillion and Change [13] and TimeWarp for heatmap with data flow of 10000 points per second.

The performance comparison between the tests with scatterplot in VisMillion and Change [13] and scatterplot in TimeWarp, implemented with Instanced Mesh, is less favorable when compared to the results of heatmap. The performance of scatterplot in VisMillion and Change [13] has a higher average FPS across execution of the tests, as shown in table 4.6. The FPS values also start to decrease in TimeWarp as the execution evolves, while in VisMillion and Change they stay relatively consistent.



(a) VisMillion and Change **[Pereira (2019)]**



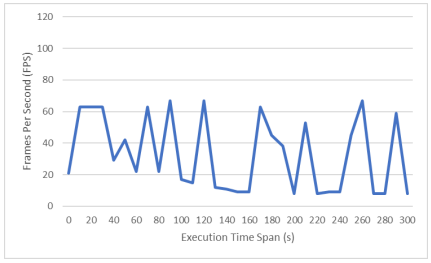
(b) TimeWarp

Figure 4.6: FPS of *VisMillion and Change* **[Pereira (2019)]** and *TimeWarp* for scatterplot with data flow of 10000 points per second.

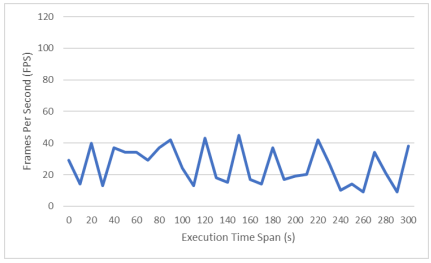
4.5 Horizontal Transitions

4.5.1 Horizontal Transitions in TimeWarp

In order to test the performance of horizontal transitions implemented in TimeWarp, the number of FPS was measured during an execution of five minutes for each of the four data flow value: 10, 100, 1000 and 10000 points per second. During that five minute time span, the respective horizontal transition between two visual idioms occurred in continuous fashion.

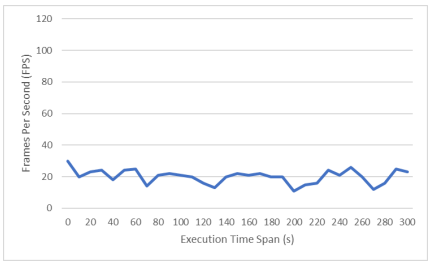


(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.

(c) Data flow of 1000 points per second.



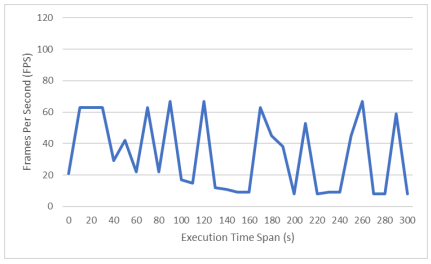
(d) Data flow of 10000 pints per second.

Figure 4.7: FPS of TimeWarp with horizontal transition between scatterplot and barchart

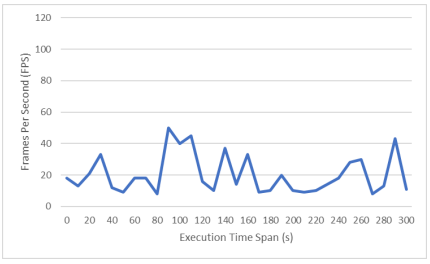
Data Flow/Metrics	Avg. FPS	Min. FPS	Max. FPS	FPS Variance
10 points per second	33	8	67	514,097
100 points per second	25,645	9	45	133,455
1000 points per second	23,548	14	34	20,377
10000 points per second	20,161	11	30	18,522

Table 4.7: Performance metrics of TimeWarp with horizontal transition between scatterplot and barchart.

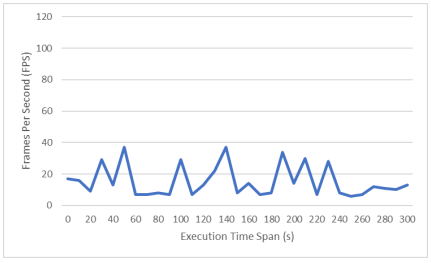
For the horizontal transition between scatterplot and barchart - fig. 4.7 -, it is observed the average number of FPS during the five minute execution Of the test decreases, as shown in table 4.7. In spite of the decrease in the average FPS, the variance level decreasing for bigger data flow values shows there is a consistency to the performance of the transition. In the final two tests - fig. 4.7(c) and fig. 4.7(d) -, it can also be observed that, as the execution time span evolves, the number of FPS starts to decrease.



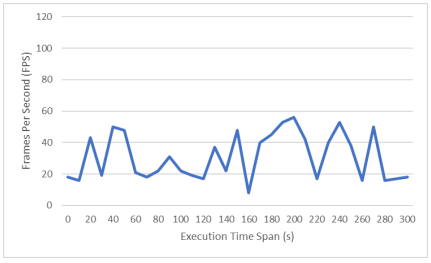
(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.



(c) Data flow of 1000 points per second.



(d) Data flow of 10000 pints per second.

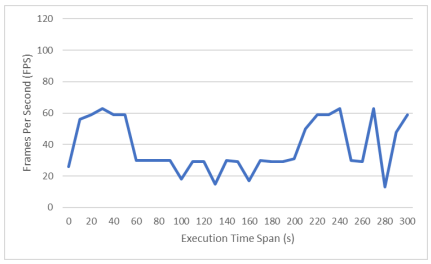
Figure 4.8: FPS of TimeWarp with horizontal transition between scatterplot and heatmap.

For the horizontal transition between scatterplot and heatmap - fig. 4.8 -, the average number of FPS for all four tests is very similar, with the data flow of ten points per second - fig. 4.8(a) - being slightly above the other tests. This shows when the horizontal transition is between scatterplot and heatmap, there is a consistency in its performance even as the data flow value increases, as shown in table 4.8.

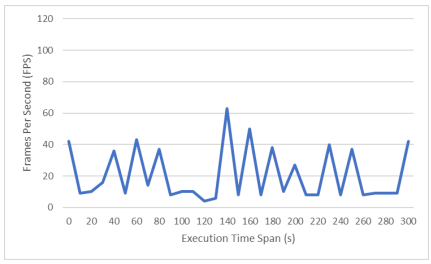
For the horizontal transition between scatterplot and linechart - fig. 4.9 -, as the data flow value

Data Flow/Metrics	Avg. FPS	Min. FPS	Max. FPS	FPS Variance
10 points per second	33	8	67	541,097
100 points per second	20,258	8	50	149,740
1000 points per second	15,323	6	37	96,477
10000 points per second	30,967	8	56	210,676

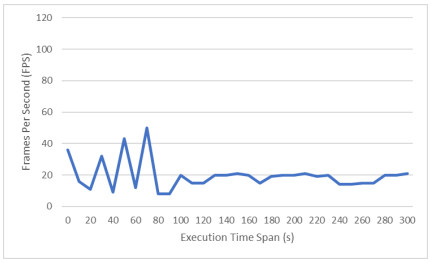
Table 4.8: Performance metrics of TimeWarp with horizontal transition between scatterplot and heatmap.



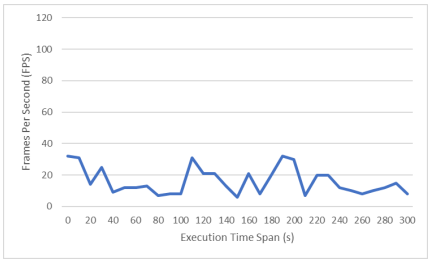
(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.



(c) Data flow of 1000 points per second.



(d) Data flow of 10000 pints per second.

Figure 4.9: FPS of TimeWarp with horizontal transition between scatterplot and linechart.

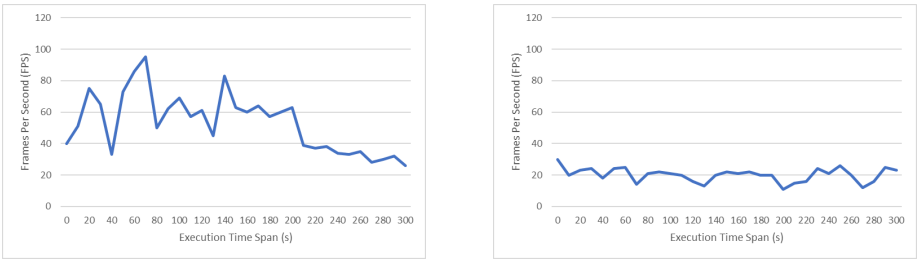
Data Flow/Metrics	Avg. FPS	Min. FPS	Max. FPS	FPS Variance
10 points per second	38,742	13	63	264,32
100 points per second	20,516	4	63	269,733
1000 points per second	19,645	8	50	83,777
10000 points per second	16	6	32	69,420

Table 4.9: Performance metrics of TimeWarp with horizontal transition between scatterplot and linechart.

increases, the average number of FPS decreases, as shown in table 4.9. This decrease means there is a decrease in performance as data flow increases.

4.5.2 D3.js vs Three.js

In order to verify the success of migrating the visual idioms of VisMillion and Change [13] from D3.js to Three.js, a comparison of the performances of VisMillion and Change and our prototype occurred. To compare performances, tests were run for a data flow of 10000 points per second during an execution time span of five minutes, where FPS values were registered and metrics were calculated for each test. This test was run for all the horizontal transitions tested in VisMillion and Change and implemented in TimeWarp.



(a) VisMillion and Change **[Pereira (2019)]**

(b) TimeWarp

Figure 4.10: FPS of VisMillion and Change [13] and TimeWarp for horizontal transition between scatterplot and barchart with data flow of 10000 points per second.

The comparison between the performances of VisMillion and Change [13] our prototype for the performance of the horizontal transition between scatterplot and barchart reveals that, even if VisMillion and Change [13] starts with a higher value of FPS when compared to our prototype, it soon starts to decrease as the execution evolves in time.

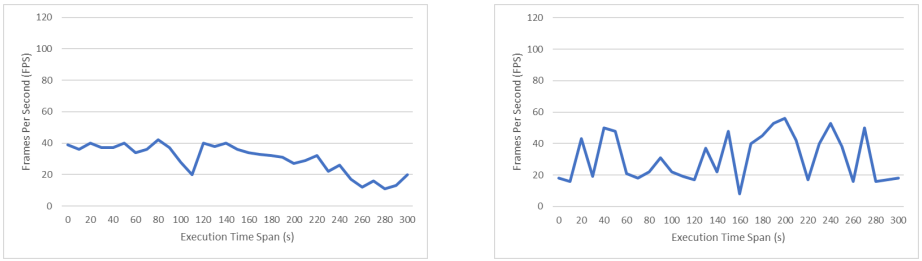
Visualization/Metrics	Avg. FPS	Min. FPS	Max. FPS	FPS Variance
VisMillion and Change	53,032	26	95	334,354
TimeWarp	20,161	11	30	18,522

Table 4.10: Performance metrics of VisMillion and Change [13] and TimeWarp for horizontal transition between scatterplot and barchart with data flow of 10000 points per second.

Visualization/Metrics	Avg. FPS	Min. FPS	Max. FPS	FPS Variance
VisMillion and Change	30,161	11	42	84,716
TimeWarp	30,967	8	56	210,676

Table 4.11: Performance metrics of VisMillion and Change [13] TimeWarp for horizontal transition between scatterplot and heatmap with data flow of 10000 points per second.

In TimeWarp, despite the inconsistency in the values of FPS (due to hardware limitation), the performance of the transition between scatterplot and barchart stays relatively consistent in its peaks. This reveals, for the transition between scatterplot and barchart, a stable performance across a long period of execution, despite having a much lower average of frames per second when compared to VisMillion and Change [13], as observed in table 4.10.



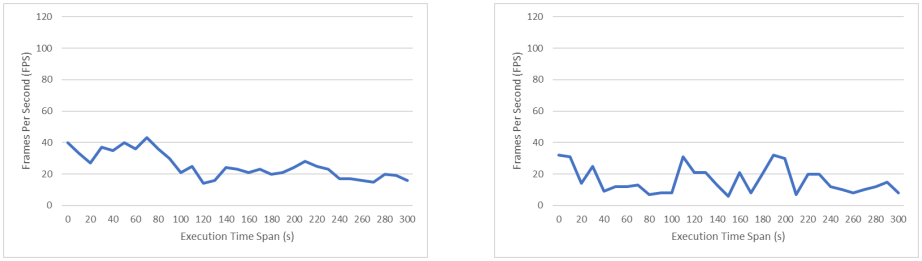
(a) VisMillion and Change **[Pereira (2019)]**

(b) TimeWarp

Figure 4.11: FPS of VisMillion and Change [13] and TimeWarp for horizontal transition between scatterplot and heatmap with data flow of 10000 points per second.

The comparison between performance of VisMillion and Change [13] and TimeWarp with horizontal transition between scatterplot and heatmap reveals performance gains in our prototype. The peak value of FPS in our prototype stays relatively consistent during the execution time span, contrary to VisMillion and Change [13] where the number of FPS gradually drops as the execution of the test evolves. The performance gains in our prototype are noticeable in table 4.11, and despite some inconsistent performance with low-end hardware, it matched VisMillion and Change [13] on average FPS during execution.

The performance of horizontal transition between scatterplot and linechart in VisMillion and Change [13] and our prototype is relatively similar in terms of average FPS, as shown in table 4.12. Once we analyze fig. 4.12, it can be profiled how VisMillion and Change [13] looses performance during execution, while TimeWarp is capable of peaks of performance far above what the tests with VisMillion and Change [13] reveal.



(a) VisMillion and Change **[Pereira (2019)]**

(b) TimeWarp

Figure 4.12: FPS of VisMillion and Change [13] and TimeWarp for horizontal transition between scatterplot and linechart with data flow of 10000 points per second.

Visualization/Metrics	Avg. FPS	Min. FPS	Max. FPS	FPS Variance
VisMillion and Change	25,323	14	43	67,702
TimeWarp	16	6	32	69,420

Table 4.12: Performance metrics of VisMillion and Change [13] and TimeWarp for horizontal transition between scatterplot and linechart with data flow of 10000 points per second.

4.6 Discussion

Through the results obtained in the various performance tests performed, we can observe our current prototype provides an inconsistent performance in a low-end system. This becomes particularly notorious when a horizontal transition between two visual idioms is happening, as observed all throughout the tests performed with horizontal transitions. Without transitions happening, the performance of TimeWarp is relatively consistent in a low-end system.

The performance of scatterplot is a point of interest in our prototype. With Instaced Mesh implemented, our prototype does gain a big performance boost compared to the simple Dots implementation. Comparing the gains of scatterplot with the gains of heatmap - who also gets a performance boost from the implementation with Instanced Mesh , the gains in perforance with scatterplot are much more notorious. While heatmap had an increase of 8,443 frames per second with Instanced Mesh implemented, scatterplot had an average gain of 17,954 frames per second across all tests. When comparing the performance of the scatterplot and heatmap of VisMillion and Change [13] with the performance of these two visual idioms in our prototype, the gains in performance in our prototype are well noticed in heatmap. On the other hand, the gains in performance of scatterplot are masqueraded by its inconsistent performance in low-end systems, as TimeWarp can indeed achieve a better peak performance in terms of FPS when compared to VisMillion and Change [13].

When analyzing the performance of horizontal transitions between scatterplot and the remaining visual idioms, the inconsistencies born out of the scatterplot performance become more apparent, especially when compared to the performance of horizontal transitions in VisMillion and Change [13]. While

the tests with VisMillion and Change [13] loose performance over time, the tests with our prototype do show - except with the horizontal transition between scatterplot and linechart - a consistency in the peak FPS values hit during execution.

Since all performance tests were carried out with low-end hardware, its hard to say if more consistent results could be achieved, for the same test execution with better hardware. In spite of that, as the peak performance of our prototype was consistent during execution, it is possible to hit a consistent performance in those peaks with better hardware.

There is also a big difference in results with some of the tests performed. In particular, the horizontal transition between scatterplot and barchart, who despite losing performance during execution in VisMillion and Change [13], still had a higher FPS value at the end of the test when compared to the highest value achieved in the same test with our prototype. The other noteworthy case of performance difference is the performance loss in the horizontal transition between scatterplot and linechart, who for both VisMillion and Change [13] and TimeWarp, started to lose performance (lower FPS) with the evolution of execution.

5

Conclusions

Contents

This dissertation presented a study on the performance of a big data streaming visualization able to display data across several visual idioms with minimal loss of context and with minimal loss of information across different time spans via the use of animated, horizontal transitions and simplification and aggregation, like graceful degradation, across several modules. The main focus of this work was to analyze how the performance of a big data streaming visualization could be improved from D3.js to Three.js in visual idioms and the horizontal transitions happening between them.

Through analyzing the state of the art of visual analytics, it was observed most big data streaming visualizations do not focus on combining performance with a capability to display big data across several visual idioms with minimal loss of context and information across different time spans. Most of those systems find challenges in dealing with big data and in finding ways to represent modifications data suffers in real-time to the end user without context or information loss. To try to reduce context and information loss, different techniques to filter, aggregate and simplify data arriving to the system can be applied while minimizing the impact on the performance of the system.

To create a visualization able to hit the object of our study - a big data streaming visualization, called TimeWarp, able to display data across several visual idioms with minimal loss of context and information across different time spans while maintaining a consistent and linear performance for different data flow rates - several steps were taken. Firstly, to create our prototype, a code migration was performed from D3.js (VisMillion and Change [13]) to Three.js, in order to improve the performance of the visualization through rendering the visualization using WebGL. The migration from D3.js to Three.js contemplates the horizontal transitions studied in VisMillion and Change [13]. Secondly, after the migration was done, some performance considerations were implemented on some visual idioms of our prototype in order to test how much they could improve performance. Those performance considerations involve the use of a Three.js object called Instanced Mesh, a special version of Mesh with instanced rendering support.

The resulting prototype follows a server-client architecture, where the server is responsible for sending big data packets to the client, responsible for the processing of the data into the visualization. Our prototype is organized in different modules, managed by an entity responsible for associating a visual idiom to a specific module and a horizontal transition between modules. The horizontal transitions between modules are a way to help with reducing context loss in our prototype, conveying the idea of operations of filtering, aggregation and simplification occurring between each module.

In order to test our prototype, three sets of performance tests were performed. The first set of tests tested the impact of Instanced Mesh in the performance of the visualization, comparing it to the simple Dots implementation. The second set of tests tested the performance of the horizontal transitions implemented in our prototype, while the third set of tests compared the performance of VisMillion and Change [13] with the performance of our prototype. For the first two tests, several different levels of data flow were generated by the server and sent to the client - 10, 100, 1000 and 10000 points per second

- while the third test only compared the later limit case of 10000 points per second. For each test, the FPS of the visualization were registered in intervals of ten seconds. From the FPS values calculated, the average number of FPS, the maximum and minimum value and the variance value were then calculated for each test.

Regarding the migration from D3.js to Three.js, Three.js offers a boost in performance as long as the visualization does not require for many object movements to be performed at the same time. In particular, when horizontal transitions are happening, our prototype struggles to hit a consistent performance with low-end hardware, struggling to match the FPS values of VisMillion and Change. This comes as a consequence of Three.js using WebGL to render the visualization, which takes advantage of the GPU for rendering purposes. In a computer with a low-end GPU, this causes performance hiccups, as the GPU can not handle the requirements of big data and streaming. In particular, the horizontal transition between scatterplot and linechart implemented in our prototype showed worse results than expected across its execution time span, as the FPS of the visualization started to decrease during execution of the test Another horizontal transition that did not perform as expected was the transition between scatterplot and barchart, displaying a worse performance when compared to its VisMillion and Change [13] counterpart.

The fact our tests were run with low-end hardware was not the only cause to the lack of performance of our prototype. While that was a big cause to the problems encountered during evaluation of the prototype, another cause for lack of performance came from the use of Instanced Mesh. While Instanced Mesh does provide a performance boost for the visualization, as we were able to concluded with our tests with heatmap and scatterplot in our prototype, when there is a horizontal transition occurring, Instanced Mesh increases the complexity of the transition, due to it not allowing tasks like individual manipulation of objects (or group of objects) and their properties, like colour, opacity or size, which usually require little computational resource to reproduce. Because of not allowing those operations to perform, the complexity of the horizontal transitions had to be increased to replicate those operations and that causes an increase in the complexity of the transitions, leading to a decrease in performance for our prototype.

Finally, considering everything said above, it is clear our goal of creating a big data streaming visualization able to display data across several visual idioms with minimal loss of context and information across different time spans while maintaining a consistent and linear performance for different data flow rates was not fully met. While our prototype was able to act as a big data streaming visualization and display data across several visual idioms with minimal loss of context and performance, we can not fully conclude about hitting our goal of consistent and linear performance for different data flow rates due to latency problems when run with low-end hardware and an increased complexity due to the use of Instanced Mesh in the implementation of the visualization.

5.1 Future Work

The development of our prototype will be carried on into the future in order to be improved and perfected. This will involve a reassessment of some parts of the visualization to understand how to extract better performance for systems with low-end hardware and how to improve consistency in the performance of the visualization across long execution time spans.

As added future work, the implementation of the vertical transitions investigated in [46] in Three.js will occur. Vertical transitions will occur between two different visual idioms but within the same module. Operations required to perform vertical transitions will be contained in a module called Vertical Transition Visualization (VTV), allowing for different representations of information in the visualization module, contributing for creating a visualization able to adapt itself to data changes by showing the most suitable visual idiom for a certain context. The VTV will be managed by Vertical Transition Module (VTM), responsible for the handling of the operations for the occurrence of vertical transitions.

To be also implemented in the future is a module capable of receiving and analyzing data packages containing metadata which, once analyzed, will provide information to the visualization on the current context of the visualization. The context obtained from the metadata will be used to determine the most suitable visual idiom to display in each module and trigger the specified animated transitions - both vertical and horizontal - to occur in the visualization.

Bibliography

[1] X. Guan, C. Xie, L. Han, Y. Zeng, D. Shen, and W. Xing, "Map-vis: A distributed spatio-temporal big data visualization framework based on a multi-dimensional aggregation pyramid model," *Applied Sciences*, vol. 10, 1 2020.

[2] A. Galletta, S. Allam, L. Carnevale, M. A. Bekri, R. E. Ouahbi, and M. Villari, "An innovative methodology for big data visualization in oceanographic domain." ACM, 4 2018.

[3] H. Zhiyuan, Z. Liang, X. Ruihua, and Z. Feng, "Application of big data visualization in passenger flow analysis of shanghai metro network." IEEE, 9 2017.

[4] N. Elmqvist, P. Dragicevic, and J.-D. Fekete, "Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, 11 2008.

[5] H. Kobayashi, K. Misue, and J. Tanaka, "Colored mosaic matrix: Visualization technique for high-dimensional data," 2013, pp. 378–383.

[6] S. Jain, E. Bensaid, and Y.-A. de Montjoye, "Unveil: Capture and visualise wifi data leakages." ACM, 5 2019.

[7] S. Katragadda, R. Gottumukkala, S. Venna, N. Lipari, S. Gaikwad, M. Pusala, J. Chen, C. W. Borst, V. Raghavan, and M. Bayoumi, "Vastream." ACM, 7 2019.

[8] G. Pires, D. Mendes, and D. Goncalves, "Vismillion: A novel interactive visualization technique for real-time big data." IEEE, 11 2019.

[9] J. Heer and G. Robertson, "Animated transitions in statistical data graphics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, 11 2007.

[10] F. Du, N. Cao, J. Zhao, and Y.-R. Lin, "Trajectory bundling for animated transitions." ACM, 4 2015.

[11] Y. Kim, M. Correll, and J. Heer, "Designing animated transitions to convey aggregate operations," *Computer Graphics Forum*, vol. 38, 6 2019.

[12] A. Lindberg, “Performance evaluation of javascript rendering frameworks,” p. 30, 2020.

[13] T. M. B. Pereira, “Vismillion and change,” 12 2019.

[14] A. Oussous, F.-Z. Benjelloun, A. A. Lahcen, and S. Belfkih, “Big data technologies: A survey,” *Journal of King Saud University - Computer and Information Sciences*, vol. 30, 10 2018.

[15] Y. Arora and D. Goyal, “Big data: A review of analytics methods amp; techniques.” IEEE, 12 2016.

[16] L. Yang, Z. Ma, L. Zhu, and L. Liu, “Research on the visualization of spatio-temporal data,” *IOP Conference Series: Earth and Environmental Science*, vol. 234, 3 2019.

[17] E.-C. Jung and K. Sato, “A framework of context-sensitive visualization for user-centered interactive systems,” 2005.

[18] R. T. D. Rover, “Performance visualization.” [Online]. Available: www.egr.msu.edu/~rover

[19] D. Reinsel, J. Gantz, and J. Rydning, “Data age 2025: The evolution of data to life-critical don’t focus on big data; focus on the data that’s big sponsored by seagate the evolution of data to life-critical don’t focus on big data; focus on the data that’s big,” 2017. [Online]. Available: www.idc.com

[20] M. Mani and S. Fei, “Effective big data visualization.” ACM Press, 2017.

[21] M. Golfarelli and S. Rizzi, “A model-driven approach to automate data visualization in big data analytics,” *Information Visualization*, vol. 19, 1 2020.

[22] E. R. Tufte, “The visual display of quantitative information.”

[23] A. M. Khan, D. Gonçalves, and D. C. Leão, “Towards an adaptive framework for real-time visualization of streaming big data.” [Online]. Available: https://www.

[24] S. Katragadda, S. Virani, R. Benton, and V. Raghavan, “Detection of event onset using twitter.” IEEE, 7 2016.

[25] J. Traub, N. Steenbergen, P. M. Grulich, T. Rabl, and V. Markl, “I2: Interactive real-time visualization for streaming data,” vol. 2017-March. OpenProceedings.org, 2017, pp. 526–529.

[26] L. Battle, M. Stonebraker, and R. Chang, “Dynamic reduction of query result sets for interactive visualizaton.” IEEE, 10 2013.

[27] C.-E. Dessart, V. G. Motti, and J. Vanderdonckt, “Animated transitions between user interface views.” ACM Press, 2012.

[28] J. T. Stasko, “Animation in user interfaces: principles and techniques,” 1993.

[29] J. Mackinlay, “Automating the design of graphical presentations of relational information,” *ACM Transactions on Graphics*, vol. 5, 4 1986.

[30] S. M. Kosslyn, “Understanding charts and graphs,” *Applied Cognitive Psychology*, vol. 3, 7 1989.

[31] P. Baudisch, D. Tan, M. Collomb, D. Robbins, K. Hinckley, M. Agrawala, S. Zhao, and G. Ramos, “Phosphor.” ACM Press, 2006.

[32] B. Bederson and A. Boltman, “Does animation help users build mental maps of spatial information?” IEEE Comput. Soc.

[33] F. Chevalier, P. Dragicevic, A. Bezerianos, and J.-D. Fekete, “Using text animated transitions to support navigation in document histories.” ACM Press, 2010.

[34] P. Dragicevic, S. Huot, and F. Chevalier, “Gliimpse: Animating from markup code to rendered documents and vice-versa gliimpse: Animating from markup code to rendered documents and vice versa,” 2011. [Online]. Available: https://hal.inria.fr/inria-00626259

[35] B. Bach, E. Pietriga, and J.-D. Fekete, “Graphdiaries: Animated transitions andtemporal navigation for dynamic networks,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, 5 2014.

[36] N. Cao, D. Gotz, J. Sun, and H. Qu, “Dicon: Interactive visual analysis of multidimensional clusters,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, 12 2011.

[37] M. Suganuma and K. Yokosawa, “Grouping and trajectory storage in multiple object tracking: Impairments due to common item motions,” *Perception*, vol. 35, 4 2006.

[38] S. Yantis, “Multielement visual tracking: Attention and perceptual organization,” *Cognitive Psychology*, vol. 24, 7 1992.

[39] S. L. Franconeri, Z. W. Pylyshyn, and B. J. Scholl, “A simple proximity heuristic allows tracking of multiple objects through occlusion,” *Attention, Perception, Psychophysics*, vol. 74, 5 2012.

[40] P. Jolicoeur, S. Ullman, and M. Mackay, “Curve tracing: A possible basic operation in the perception of spatial relations,” *Memory Cognition*, vol. 14, 3 1986.

[41] A. Sarikaya, M. Gleicher, and D. A. Szafir, “Design factors for summary visualization in visual analytics,” *Computer Graphics Forum*, vol. 37, 6 2018.

[42] B. TVERSKY, J. B. MORRISON, and M. BETRANCOURT, “Animation: can it facilitate?” *International Journal of Human-Computer Studies*, vol. 57, 10 2002.

[43] P. O. Box, L. V. D. Maaten, E. Postma, and J. V. D. Herik, "Tilburg centre for creative computing dimensionality reduction: A comparative review dimensionality reduction: A comparative review," 2009. [Online]. Available: http://www.uvt.nl/ticc

[44] J. Yan, B. Zhang, N. Liu, S. Yan, Q. Cheng, W. Fan, Q. Yang, W. Xi, and Z. Chen, "Effective and efficient dimensionality reduction for large-scale and streaming data preprocessing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, 3 2006.

[45] G. F. Pires, D. J. V. G. J. Presidente, M. N. D. A. P. C. Orientador, D. J. V. G. Vogal, and S. P. Gama, "Visbig visualizar bigdata em tempo real," 2018.

[46] F. M. B. Castanheira, "Fastviz - visualizing dynamically evolving big data," 1 2021.

[47] K. Kontogiannis, J. Martin, K. Wong, R. Gregory, H. Müller, and J. Mylopoulos, "Code migration through transformations." ACM Press, 2010.

[48] E. Angel and E. Haines, "An interactive introduction to webgl and three.js." ACM, 7 2017.

[49] T. Parisi, *WebGL: Up and Running*, 1st ed. O'Reilly Media, Inc., 2012.