Miguel Rocha
Instituto Superior Técnico
Lisbon, Portugal
miguel.antonio@tecnico.ulisboa.pt

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

1 INTRODUCTION

Big Data, a "large growing data sets that include heterogeneous formats: structured, unstructured and semi-structured data" [18], has become one of the most important and researched areas of computer science of today. Big data has very specific characteristics, mostly evidenced by the **5V**: variety, volume, velocity, value, and veracity [2]. The processing and analysis of big data require significant computational resources to guarantee flexibility, scalability, and consistent performance - all characteristics of a good visualization.

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.

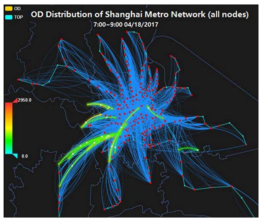
When one combines big data with streaming into a visualization, traditional vis. techniques are deemed unsuitable. Therefore, we must investigate the best way to map data in big data streaming visualizations while searching on how to obtain the best performance out of a visualization. Performance impact connects directly to the concept of performance visualization, a type of software visualization that includes aspects such as hardware performance [22].

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.

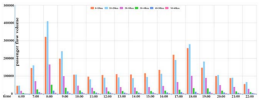
2 STATE OF THE ART

In the last couple of years, the amount of work developed in big data has increased. 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, usually with the aid of a visualization. Creating a comprehensive visualization has its challenge, mainly connected to the 5V and how each domain is different and inserted into its own specific context. In order to handle big data, some pre-processing is required to be applied to the data before it can be visualized by the end user. An example of this is the analysis of the passenger flow of the Shanghai Metro Network [25]. Two operations can be found on [25] - one for the same transfer station and the different station codes for different lines and another for the data of several trips, as its joined to find the volume of passenger flow that traveled a certain distance in a certain period of the day.

The work on [25] 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. A good example of the application of correlations in big data visualizations can be found in the work Rolling the Dice [11]. Rolling the Dice [11] is a visualization for exploration of multidimensional data through combination of correlation matrices and scatterplots. The combination of scatterplot and correlation matrices in Rolling the Dice [11] results in a scatterplot matrix, where each scatterplot (columns) corresponds to a dimension of the data set (lines).



(a) Dynamic migration map visualization



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

Figure 1: Visualization Analysis of the passenger flow of the Shanghai Metro Network [25].

Figure 2: Scatterplot matrix component used for over-view and interaction in Rolling the Dice [11].

Nowadays, most of the visualizations like the ones mentioned above - [12], [25], [11], [16] - 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.

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 [14], 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.

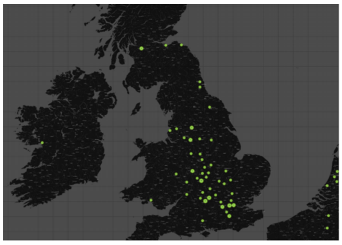


Figure 3: Visualization of results obtained with probe requests analysis on Unveil [14]. Each dot represents an aggregation of detected devices into a single location.

Creating a data stream visualization also presents some challenges. 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 [24], an interactive development environment. The third challenge presented above leads us to a situation where, for a data streaming visualization to work properly, the amount of data arriving cannot interfere (in a noticeable way) with performance. A possible solution to avoid performance hiccups is the use of graceful degradation, a concept vastly explored in VisMillion [20]. 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.

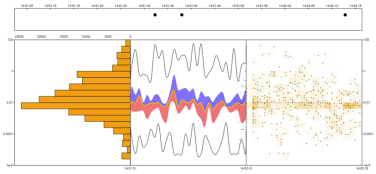


Figure 4: VisMillion interface [20].

In order to help the end user perceive that aggregation techniques like graceful degradation (and other types of techniques, like simplification, filtering and dimensionality reduction) are occurring in the visualization, transitions can be used to ease the perception of those visual changes. 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" [23].

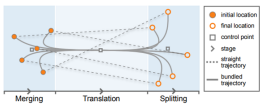
In order to achieve smooth transitions, staged animations can offer additional benefits to a visualization [13]. In spite of that, the application of animation techniques come with shortcomings, as shown in [4], [5], [8],[9]. These include how animated transitions attract, first, the attention of the 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.

Intertwined in all these shortcomings is also the duration of the animation. Depending on how long the animation runs

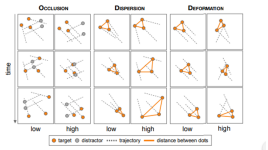
for, the impact of latency, depending on available computing resources, will vary. An animation lasting too little or too long will also impact the attention span of the user but also the consistency of the visualization. Animated transitions should be as fast as possible without making the end user overlook the actual transition [8]. 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 [3] and switching between different layout methods [6].

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, explored on [10].



(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 5: Trajectory bundling [10].

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. Therefore, it is important the performance of animated transitions is profiled for a a context where aggregation techniques are used, as highlighted in [15], showing how, in animated transitions, the animation chosen for a specific transition is very much connected to the context it is wrapped in.

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 [20], as it is prepared to serve big data and real time streaming data. In spite of that, VisMillion [20] 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 [19] is a visualization developed with the aim of fixing some of the gaps existing in VisMillion [20]. VisMillion and Change [19] explores horizontal transitions between the modules of VisMillion [20] to reduce loss of context across the visualization while mainting a consistent performance.

Our visualization, **TimeWarp**, aims to fix the missing gap of performance in VisMillion and Change [19] 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

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 [21] concept, followed by presenting the process of migrating VisMillion and Change [19] 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.

VisMillion, the concept

The concept of VisMillion [21] 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 and using a different visual idiom. The visualization uses the techniques of graceful degradation to present data.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.

The concept of VisMillion was enhanced firstly in [20] and then further enhanced in VisMillion and Change [19]. Our prototype is based on the VisMillion [21] concept and visualization [20] while carrying on the work done on VisMillion and Change [19] and FastViz [7].

Migration of VisMillion and Change

Migration is an operation that can be done at different levels. The process of a **migration** can be arbitrarily understood as the movement of code into a new platform and/or programming language [17].

In the case of our prototype, migration consists migrating VisMillion and Change [19], implemented in D3.js[1], to Three.js [2]. While D3.js is a JS library for manipulating documents based on data using HTML, , and 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 [1].

For VisMillion and Change [19], the migration 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. During the migration process of VisMillion and Change [19] 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.

TimeWarp Architecture

The architecture employed in TimeWarp follows the same structure as the architecture of VisMillion and Change [19]. Real time data streaming is simulated through data packages generated and sent by a data flow generator - Streamer, a Python[3] 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.

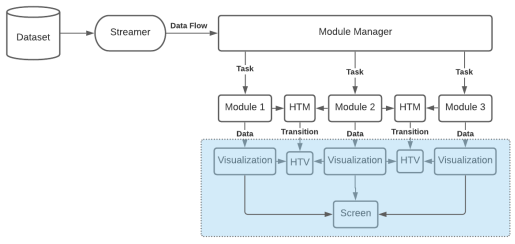


Figure 6: Architecture diagram of TimeWarp. Research focus is highlighted.

[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.

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

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 it.

The connection between the three modules is performed via another module: HTM (Horizontal Transition Module). The HTM is attached to a visualization - the HTV (Horizontal Transition Visualization). 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.

TimeWarp Interface

The TimeWarp interface follows the same basic principle of the VisMillion and Change [19] interface. The interface is simple, yet functional, 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. 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.

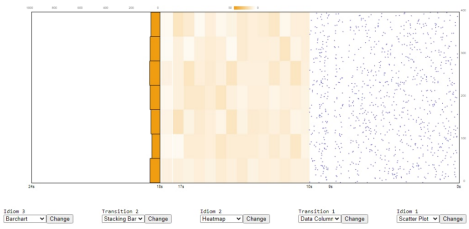


Figure 7: TimeWarp interface.

From the right to the left of our visualization, 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. This is the graceful degradation technique in TimeWarp. 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.

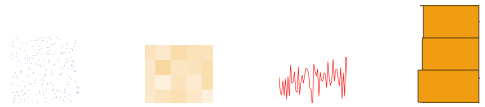
Idioms

All the visual idioms of VisMillion and Change [19] - scatterplot, heatmap, linechart and barchart - also exist in our

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

Table 1: Horizontal transitions implemented on TimeWarp, based on the work VisMillion and Change [19].

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 [19] in order to compare performances between the two visualizations. Horizontal transitions occur between these visual idioms when they are assigned to a specific module. T



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

Figure 8: Visual idioms of TimeWarp.

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.

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 [19], 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 horizontal transitions implemented in our prototype - specified in table 1 were the transitions with the best results for user testing in the user evaluation performed in [19]. All the implemented transitions obey to the principles determined on [13], 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.

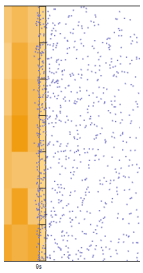


Figure 9: Scatterplot to heatmap horizontal transition.

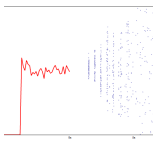


Figure 10: Scatterplot to linechart horizontal transition.

.

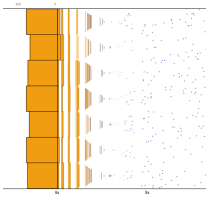


Figure 11: Scatterplot to barchart horizontal transition.

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[4], a special version of Mesh[5] with instanced rendering support. The usage of of Instanced Mesh also helps to reduce the number of draw calls for the pipeline.

The use of Instanced Mesh is applied to two visual idioms in *TimeWarp*: scatterplot and heatmap. while Instanced Mesh can possibly offer performance gains, it also brings some drawbacks, mostly related to not allowing individual manipulation of elements of a visualization and their properties. The existence of these drawbacks is why heatmap and

[4]https://threejs.org/docs/api/en/objects/InstancedMesh
[5]https://threejs.org/docs/api/en/objects/Mesh

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

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 **??**. Conclusions on the performance of our prototype will be based on the analysis of results obtained during the performance testing.

There are three set of tests to be done with our prototype. Firstly, the performance considerations specified in **??** 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). 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 [19] - 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).

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.

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 FPS (Frames Per Second) of the visualization during each test and data flow value being sent to our prototype. 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.

Performance Tests Methodology

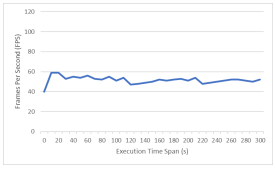
For the three set of tests performed with our prototype, the number of FPS were recorded in intervals of ten seconds. The number of FPS were calculated with the inverse of the difference between the current time and time of the last computed. Every ten seconds, when FPS were calculated, the value was 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. 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 [19] performed in a worst case scenario. All the tests were performed across a time interval of 5 minutes (300 seconds).

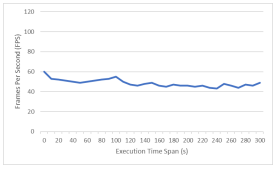
Visual Idioms

Dots vs Instanced Mesh

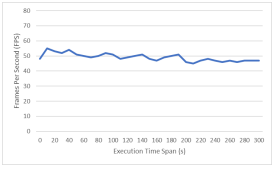
In order to test how the performance considerations mentioned in **??** impact the performance of TimeWarp, heatmap and scatterplot were tested with (Instanced Mesh) and without (Dots) the performance considerations implemented. For each test, they were tested for four different data flow values and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



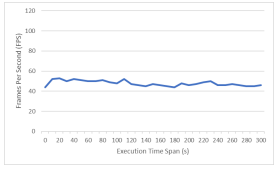
(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.



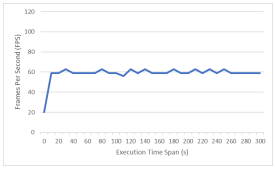
(c) Data flow of 1000 points per second.



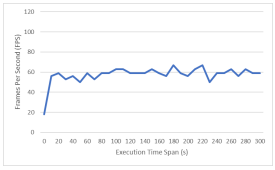
(d) Data flow of 10000 pints per second.

Figure 12: FPS of TimeWarp with heatmap (Dots) as its only visual idiom.

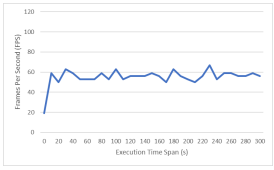
Independently of the data flow, the test comparison between heatmap with - Figure 13 - and without - Figure 12 - Instanced Mesh implemented achieved relatively similar results in terms of consistent FPS values. Despite similarities, the FPS values of heatmap without Instanced Mesh are



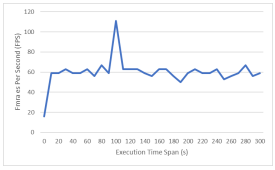
(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.



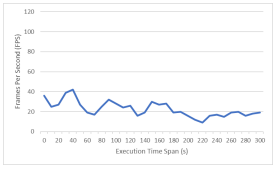
(c) Data flow of 1000 points per second.



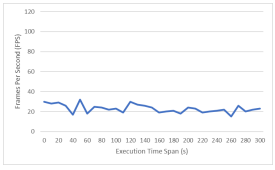
(d) Data flow of 10000 pints per second.

Figure 13: FPS of TimeWarp with heatmap (Instanced Mesh) as its only visual idiom.

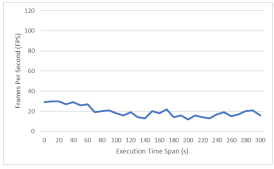
slightly lower compared to heatmap with Instanced Mesh implemented.



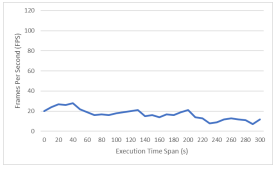
(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.



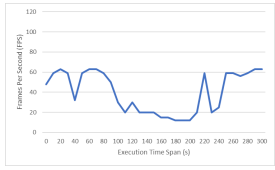
(c) Data flow of 1000 points per second.



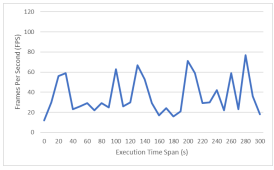
(d) Data flow of 10000 pints per second.

Figure 14: FPS of TimeWarp with scatterplot (Dots) as its only visual idiom.

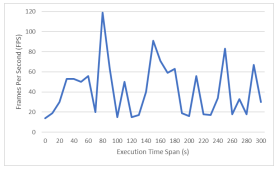
Contrary to the minimal difference found between the two versions of heatmap, the differences in performance between scatterplot implemented with performance considerations - fig. 15 - and scatterplot implemented without performance considerations - fig. 14 - is more noticeable. The peak FPS and the average FPS for each test with Instanced Mesh implemented is higher than the corresponding test with a simple Dots implementation. For both situations, however, as the data flow value increases, the FPS drop becomes more noticeable as the execution approaches the five minutes mark,



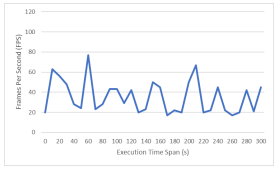
(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.



(c) Data flow of 1000 points per second.



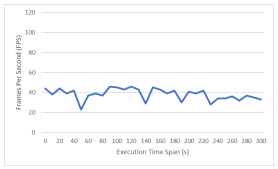
(d) Data flow of 10000 pints per second.

Figure 15: FPS of TimeWarp with scatterplot (Instanced Mesh) as its only visual idiom.

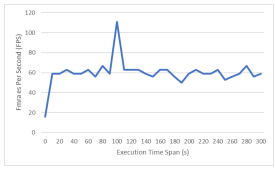
although that drop is more pronounced in the tests with a simple Dots implementation.

D3.js vs Three.js

In order to verify the success of migrating the visual idioms of VisMillion and Change [19] 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.



(a) VisMillion and Change

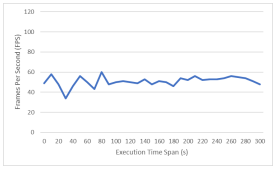


(b) *TimeWarp*

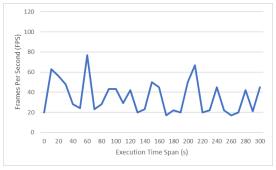
Figure 16: FPS of VisMillion and Change and TimeWarp for heatmap with data flow of 10000 points per second.

When comparing the performance of heatmap in VisMillion and Change [19] 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 [19].

The performance comparison between the tests with scatterplot in VisMillion and Change [19] and scatterplot in TimeWarp, implemented with Instanced Mesh, is less favorable



(a) VisMillion and Change



(b) TimeWarp

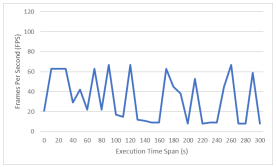
Figure 17: FPS of *VisMillion and Change* [Pereira (2019)] and *TimeWarp* for scatterplot with data flow of 10000 points per second.

when compared to the results of heatmap. The performance of scatterplot in VisMillion and Change [19] has a higher average FPS across execution of the tests, as shown in **??**. The FPS values also start to decrease in TimeWarp as the execution evolves, while in VisMillion and Change they stay relatively consistent.

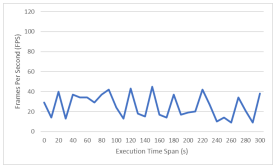
Horizontal Transitions

Horizontal Transitions in TimeWarp

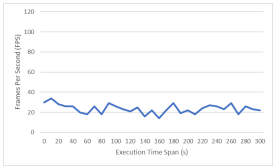
In order to test the performance of horizontal transitions implemented in TimeWarp, the number of FPS was measured during an execution of five minutes for each of the four data flow value: 10, 100, 1000 and 10000 points per second. During that five minute time span, the respective horizontal transition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



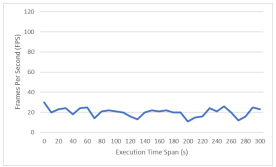
(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.



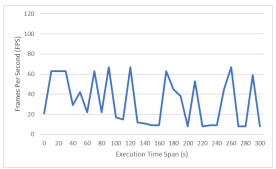
(c) Data flow of 1000 points per second.



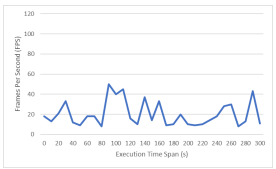
(d) Data flow of 10000 pints per second.

Figure 18: FPS of TimeWarp with horizontal transition between scatterplot and barchart

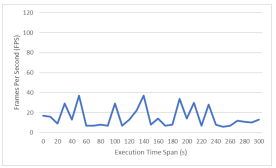
For the horizontal transition between scatterplot and barchart, it is observed the average number of FPS during the five minute execution Of the test decreases. In spite of the decrease in the average FPS, the variance level decreasing for bigger data flow values shows there is a consistency to the performance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



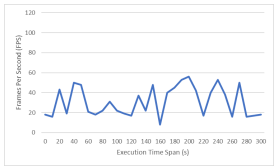
(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.



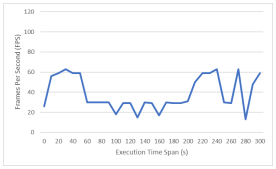
(c) Data flow of 1000 points per second.



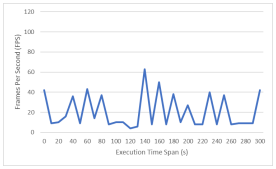
(d) Data flow of 10000 pints per second.

Figure 19: FPS of TimeWarp with horizontal transition between scatterplot and heatmap.

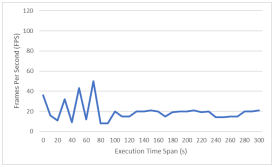
For the horizontal transition between scatterplot and heatmap, the average number of FPS for all four tests is very similar, with the data flow of ten points per second being slightly above the other tests. This shows when the horizontal transition is between scatterplot and heatmap, there is a consistency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



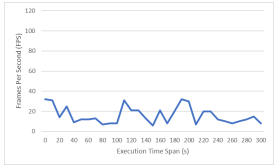
(a) Data flow of 10 points per second.



(b) Data flow of 100 points per second.



(c) Data flow of 1000 points per second.



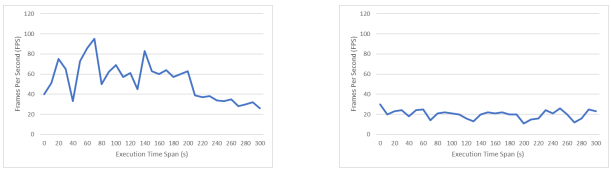
(d) Data flow of 10000 pints per second.

Figure 20: FPS of TimeWarp with horizontal transition between scatterplot and linechart.

For the horizontal transition between scatterplot and linechart -, as the data flow value increases, the average number of FPS decreases.

D3.js vs Three.js

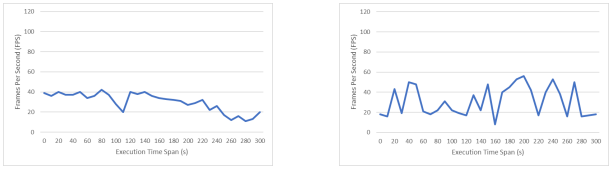
n order to verify the success of migrating the visual idioms of VisMillion and Change [19] from D3.js to Three.js, a comparison of the performances of VisMillion and Change and our prototype occurred. To compare performances, tests were run for a data flow of 10000 points per second during an execution 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 (b) TimeWarp

Figure 21: FPS of VisMillion and Change 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 [19] our prototype for the performance of the horizontal transition between scatterplot and barchart reveals that, even if VisMillion and Change [19] starts with a higher value of FPS when compared to our prototype, it soon starts to decrease as the execution evolves in time.

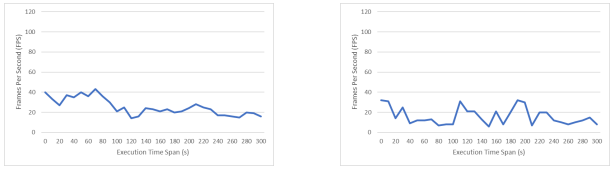


(a) VisMillion and Change (b) TimeWarp

Figure 22: FPS of VisMillion and Change 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 [19] 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.

The performance of horizontal transition between scatterplot and linechart in VisMillion and Change [19] and our



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

Figure 23: FPS of VisMillion and Change and TimeWarp for horizontal transition between scatterplot and linechart with data flow of 10000 points per second.

prototype is relatively similar shows how VisMillion and Change [19] looses performance during execution, while TimeWarp is capable of peaks of performance far above what the tests with VisMillion and Change [19] reveal.

5 CONCLUSION

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. This comes as a conclusion when discussing the results of the performance tests performed with our prototype.

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. 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,

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.

6 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 [7] in Three.js will occur. Vertical transitions will occur between two different visual idioms but within the same module. 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.

REFERENCES

[1] Ed Angel and Eric Haines. 2017. An interactive introduction to WEBGL and three.JS. *ACM SIGGRAPH 2017 Courses*. https://doi.org/10.1145/3084873.3084875

[2] Yojna Arora and Dinesh Goyal. 2016. Big data: A review of analytics methods amp; techniques. *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*. https://doi.org/10.1109/IC3I.2016.7917965

[3] Benjamin Bach, Emmanuel Pietriga, and Jean-Daniel Fekete. 2014. GraphDiaries: Animated Transitions andTemporal Navigation for Dynamic Networks. *IEEE Transactions on Visualization and Computer Graphics* 20 (5 2014). Issue 5. https://doi.org/10.1109/TVCG.2013.254

[4] Patrick Baudisch, Desney Tan, Maxime Collomb, Dan Robbins, Ken Hinckley, Maneesh Agrawala, Shengdong Zhao, and Gonzalo Ramos. 2006. Phosphor. *Proceedings of the 19th annual ACM symposium on User interface software and technology - UIST '06*. https://doi.org/10.1145/1166253.1166280

[5] B.B. Bederson and A. Boltman. [n.d.]. Does animation help users build mental maps of spatial information? *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis'99)*. https://doi.org/10.1109/INFVIS.1999.801854

[6] Nan Cao, D. Gotz, J. Sun, and Huamin Qu. 2011. DICON: Interactive Visual Analysis of Multidimensional Clusters. *IEEE Transactions on Visualization and Computer Graphics* 17 (12 2011). Issue 12. https://doi.org/10.1109/TVCG.2011.188

[7] Filipa Margarida Barros Castanheira. 2021. FastViz - Visualizing Dynamically Evolving Big Data.

[8] Fanny Chevalier, Pierre Dragicevic, Anastasia Bezerianos, and Jean-Daniel Fekete. 2010. Using text animated transitions to support navigation in document histories. *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. https://doi.org/10.1145/1753326.1753427

[9] Pierre Dragicevic, Stéphane Huot, and Fanny Chevalier. 2011. Gliimpse: Animating from Markup Code to Rendered Documents and Vice-Versa Gliimpse: Animating from Markup Code to Rendered Documents and Vice Versa. https://hal.inria.fr/inria-00626259

[10] Fan Du, Nan Cao, Jian Zhao, and Yu-Ru Lin. 2015. Trajectory Bundling for Animated Transitions. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/2702123.2702476

[11] N. Elmqvist, P. Dragicevic, and J.-D. Fekete. 2008. Rolling the Dice: Multidimensional Visual Exploration using Scatterplot Matrix Navigation. *IEEE Transactions on Visualization and Computer Graphics* 14 (11 2008). Issue 6. https://doi.org/10.1109/TVCG.2008.153

[12] Antonino Galletta, Salma Allam, Lorenzo Carnevale, Moulay Ali Bekri, Rachid El Ouahbi, and Massimo Villari. 2018. An innovative methodology for big data visualization in oceanographic domain. *Proceedings of the International Conference on Geoinformatics and Data Analysis*. https://doi.org/10.1145/3220228.3220238

[13] Jeffrey Heer and George Robertson. 2007. Animated Transitions in Statistical Data Graphics. *IEEE Transactions on Visualization and Computer Graphics* 13 (11 2007). Issue 6. https://doi.org/10.1109/TVCG.2007.70539

[14] Shubham Jain, Eden Bensaid, and Yves-Alexandre de Montjoye. 2019. UNVEIL: Capture and Visualise WiFi Data Leakages. *The World Wide Web Conference*. https://doi.org/10.1145/3308558.3314143

[15] Younghoon Kim, Michael Correll, and Jeffrey Heer. 2019. Designing Animated Transitions to Convey Aggregate Operations. *Computer Graphics Forum* 38 (6 2019). Issue 3. https://doi.org/10.1111/cgf.13709

[16] Hiroaki Kobayashi, Kazuo Misue, and Jiro Tanaka. 2013. Colored mosaic matrix: Visualization technique for high-dimensional data. *Proceedings of the International Conference on Information Visualisation*, 378–383. https://doi.org/10.1109/IV.2013.50

[17] K. Kontogiannis, J. Martin, K. Wong, R. Gregory, H. Müller, and J. Mylopoulos. 2010. Code migration through transformations. *CASCON First Decade High Impact Papers on - CASCON '10*. https://doi.org/10.1145/1925805.1925817

[18] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih. 2018. Big Data technologies: A survey. *Journal of King Saud University - Computer and Information Sciences* 30 (10 2018). Issue 4. https://doi.org/10.1016/j.jksuci.2017.06.001

[19] Tiago Miguel Borralho Pereira. 2019. VisMillion and Change.

[20] Goncalo Pires, Daniel Mendes, and Daniel Goncalves. 2019. VisMillion: A novel interactive visualization technique for real-time big data. *2019 International Conference on Graphics and Interaction (ICGI)*. https://doi.org/10.1109/ICGI47575.2019.8955070

[21] Gonçalo Fialho Pires, Daniel Jorge Viegas Gonçalves Júri Presidente, Miguel Nuno Dias Alves Pupo Correia Orientador, Daniel Jorge Viegas Gonçalves Vogal, and Sandra Pereira Gama. 2018. VisBig Visualizar BigData em tempo real.

[22] Reusability T Diane Rover. [n.d.]. Performance Visualization. www.egr.msu.edu/~rover

[23] John T Stasko. 1993. Animation in user interfaces: principles and techniques.

[24] Jonas Traub, Nikolaas Steenbergen, Philipp M. Grulich, Tilmann Rabl, and Volker Markl. 2017. I2: Interactive real-Time visualization for streaming data. *Advances in Database Technology - EDBT* 2017-March, 526–529. https://doi.org/10.5441/002/edbt.2017.61

[25] Huang Zhiyuan, Zhang Liang, Xu Ruihua, and Zhou Feng. 2017. Application of big data visualization in passenger flow analysis of Shanghai Metro network. *2017 2nd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*. https://doi.org/10.1109/ICITE.2017.8056905