



Bachelor Degree Project

BabylonJS and Three.js

*- Comparing performance when it comes to
rendering Voronoi height maps in 3D*



Author: Oscar Nordquist, Axel Karlsson
Supervisor: Johan Hagelbäck
Semester: VT/HT 2017
Subject: Computer Science

Abstract

WebGL is a technique that allows the browser to run 3D applications with the help of the GPU. Voronoi diagrams are a set of polygons that can be used to illustrate worlds of islands. In an web-application using Voronoi Polygons to create two dimensional worlds there is a future vision to enable three dimensional behavior. There are multiple frameworks and libraries that can be used to simplify the process of creating 3D applications in the browser. Due to the fact that 3D applications can be performance demanding, an experiment was conducted with BabylonJS and Three.js. In order to evaluate which one of the two performed better, RAM, GPU and CPU were evaluated when translating two dimensional Voronoi heightmaps into a 3D application. The results from this stress test prove that Three.js outperformed BabylonJS.

Contents

1 Introduction	4
1.1 Background	5
1.1.1 Voronoi Diagram	6
Figure 1.1: Visual Representation of a Voronoi Diagram	6
1.1.2 Heightmap	6
1.1.3 WebGL	7
1.1.4 Stress Test	7
1.1.5 Performance	8
1.2 Related work	8
1.3 Problem formulation	9
1.4 Motivation	10
1.5 Objectives	10
1.6 Scope	11
1.7 Target group	11
1.8 Outline	11
2 Method	13
2.1 Approach	13
2.2 Description	13
2.3 Machines and Devices	14
Device	14
2.4 Reliability and Validity	15
3 Implementation	16
3.1 Applications	16
3.1.1 Performance Recorder	16
Opn (version 5.2.0)	16
systeminformation (version 3.37.3)	16
tree-kill (version 1.2.0)	17
3.2 Libraries and Frameworks	17
3.2.1 BabylonJS	17
3.2.2 Three.js	17
3.3 Benchmarking	18
4 Results	20

4.1 Explanation of the Results	20
4.1.1 GPU	20
4.1.2 CPU	21
4.1.3 RAM	21
5 Analysis	35
5.1 GPU	35
5.2 CPU	37
5.3 RAM	40
Figure 5.9 - CPU Data is not approximately normally distributed (Babylon)	41
5.4 Summary	41
6 Discussion	43
7 Conclusion	44
7.1 Future work	45
References	46
Appendix 1	50
Appendix 2	52
Appendix 3	52

1 Introduction

The development of the web browser and the world wide web as a platform has taken tremendous steps in a relatively short period of time, since Tim Berners Lee initially invented the world wide web in 1989 at CERN [1]. When the development of the world wide web was started it was intended to mainly be a platform that should enable a simpler way to share information between, for example, researchers and universities[1]. The paradigm of the web browsers has changed and more and more advanced technology has become popular in the world of browsers.

3D graphics is not something that is new, games have been developed and played with three-dimensions for a relatively long period of time. There are a set of different methods for rendering 3D graphics in the browser. CSS3 enables one to give elements three dimensional behavior with a property called transform[2]. Another technique is a JavaScript API called WebGL which will be used with the help of the libraries BabylonJS and Three.js. The main focus of this thesis is to evaluate the CPU, RAM and FPS usage whilst rendering a heightmap based on a Voronoi diagram.

1.1 Background

Beanloop was in the development stage of a web application that was to be used to, for example, helping and simplifying the definition of company values, or visualizing one's personal life. In order to visualize the purpose of a company or one's life, different islands are presented and displayed as Voronoi (See explanation in section 1.2) polygons. In the future, the customer envisions “diving” into the islands; instead of the current 2D presentation of the islands. The customer wants to enable the user of the application to view the islands in 3D space, providing a more immersive perspective.

There are multiple ways of displaying 3D figures in a browser environment which can, for example, be done with HTML5 / CSS3 [1]. However, these technologies is one way of enabling visualization of 3D elements in the browser, another is WebGL. Merixstudio explains WebGL as

“ [...] an implementation of a low-level programming interface for 3D graphics based on the OpenGL ES 2.0. It uses DOM and HTML5 Canvas element for this purpose [2]. ”

In order to use WebGL, a large variety of libraries which simplify the development process exists. Our focus for this thesis is WebGL libraries and their performance. Since rendering 3D graphics in the browser can be a daunting task, as well as performance consuming, we want to compare the performance of a set of libraries to find the most suitable candidate for the customer's future vision.

1.1.1 Voronoi Diagram

A Voronoi Diagram is the result of partitioning a plane into a set of different points [6]. The inner representation of a Voronoi diagram can be explained with a node that represents specific point that occurred during partitioning, and this node has a point called a site. The site is placed in the center based on its relations to the neighbors. During the partitioning, edge-lines are drawn in the middle of two points. The edge-lines gets varied results based on p-values and weight [23]. Each node also has neighbors to the nodes (partitioned points) in the diagram. In **Figure 1.1** below displays an example of a Voronoi diagram with neighbors and sites. Voronoi Diagrams have been applied in areas such as Health in order to analyze muscle tissue[7]. In a study written by F. Aurenhammer [8], the usage appeal of Voronoi Diagrams can be found in, for example, Cluster Analysis and Collision Detection.

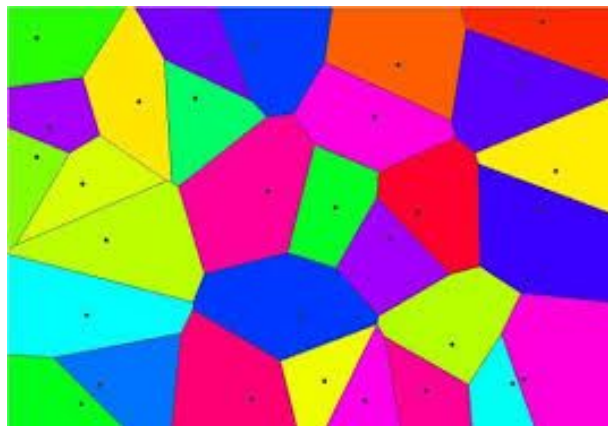


Figure 1.1: Visual Representation of a Voronoi Diagram

1.1.2 Heightmap

A heightmap is a picture of, for example, a landscape converted into a grayscale color scheme. The heightmap is useful in the sense that it

represents elevation data. The elevation data of the heightmap is represented by the colors and its shade.

The heightmap does not hold any values other than its' colors in order to represent the altitude. Below is an example of a heightmap that was used in the experiment of this report. The regions with a brighter shade are the regions with the highest altitude. The height-maps are to be used to transform the two-dimensional world into a three-dimensional world, to maintain its original form and to represent the elevation correctly.

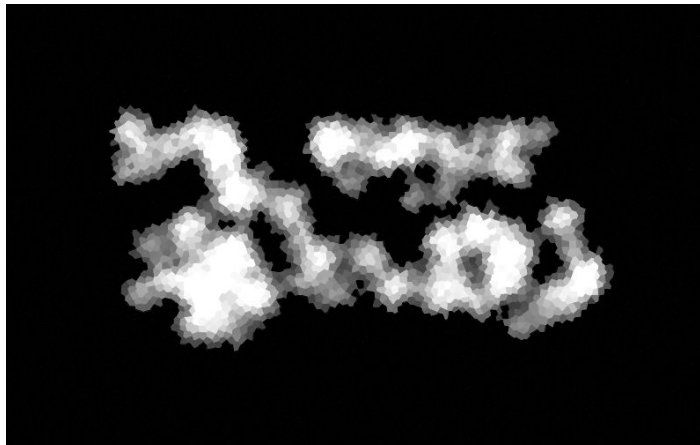


Figure 1.2: Example of a Heightmap

1.1.3 WebGL

The JavaScript API WebGL is used to render interactive 3D and 2D graphics. WebGL is available by default in all major browsers [5], but the performance is higher in Firefox and Chrome, since these are using a newer version called WebGL 2, which is a major update of the API.

When creating web content with massive structures of geometry in 3D, one should use WebGL since it makes use of the GPU rather than the CPU [3]. This is a good thing since GPUs have a much higher performance when it comes to rendering, and visualization [4].

1.1.4 Stress Test

A stress test is used to evaluate how a specific system performs under heavy load. Stress tests can be applied to many diverse tasks in computer science, for example, it can be applied when testing a website. This could be done by emulating a heavy load of traffic to the website in order to understand the flaws or to ensure that the website can handle heavy loads. It can also be used

when testing computer hardware to evaluate that the hardware components are stable under heavy load. Since stress tests are a key component for analyzing how a system performs under heavy load, this is a suiting test strategy since this thesis tries to evaluate how BabylonJS and Three.js handles heavy load.

1.1.5 Performance

This thesis will contain information regarding performance differences of WebGL libraries when used together with Voronoi polygons and heightmaps. The performance which is going to be measured is RAM-, CPU- and GPU load.

The computer processing unit, or CPU handles the logical control of software, which is important since the stress tests will consist of many actions which will be processed. These actions will affect the performance and will increase the processing power used. Therefore the main goal is to keep the processing power at a minimum since that would mean that the library has computational resources left for other tasks.

When calculating how much data that the computer is currently using, random-access memory, or RAM, will be used. If the ram usage is low, the application would be classified as “light-weight”. That would mean the library is better at handling the application the lower the RAM usage is. Therefore it is one of the key components of performance statistics.

The graphics processing unit or GPU are used in order to accelerate the rendering of images in a frame buffer. In the section below, WebGL is introduced as a methodology for rendering 3D shapes. When one is talking about 3D rendering, there is a terminology called 3D pipeline. Included in the pipeline is a set of terminologies that have their own responsibility. These metrics are used in order to measure the GPUs’ performance during the experiments.

1.2 Related work

Since it has been proven that WebGL is an effective way to render polygons in 3D space, lots of research has been made on this particular subject. However, finding research about this thesis’ specific topic did turn out to be quite lacking and did not lead to any results. The related works that were

found are focused on specific sections of the thesis such as Voronoi or performance and not the thesis as a whole.

Atitayaporn Muennoi and Daranee Hormdee states in their thesis *3D Web-based HMI with WebGL Rendering Performance* that WebGL and its performance whilst rendering human-machine interfaces is increased when compared to 3D which is rendered without WebGL [13]. That topic is something that will not directly be discussed in this thesis, it does, however, contain information regarding WebGL and its performance which makes it relevant to this thesis. Some of the information in the article is quite valuable since it covers ways to calculate the performance of two different rendering methods, what data is to be collected from the two, what browsers are to be used, and the hardware used whilst doing their tests. The information in their thesis might not give this paper any technical information to be used, but since they were analyzing a similar problem, some parts can be applied to this paper.

Another paper with the name *Fully peer-to-peer virtual environments with 3D Voronoi diagrams* by Mahathir Almashor and Ibrahim Khalil covers Voronoi in 3D-space which gives a perspective on other useful areas for Voronoi diagrams [14]. Their thesis covers questions about the performance and usage of Voronoi diagrams, and their performance in a massively multiplayer online game, which is not something that will be discussed in this paper. However, their thesis is still relevant to this paper since it contains information regarding Voronoi and how they perform in 3D space. The paper also contains information regarding 2D to 3D conversion of Voronoi polygons which is quite interesting for this thesis. Even though the paper does not cover the same type of system environment as this thesis, it is rather interesting how they solved Voronoi performance in 3D in their environment. Some of the parts of the paper might apply to some areas of this thesis.

1.3 Problem formulation

The data which is to be presented is initially located in 2D space, as seen from above, and the client would like this presentation to be converted into 3D space. In order to do so, the data will have to be parsed into a presentational layer with an additional dimension. Information regarding the conversion of 2D space to 3D space whilst working with statistics and Voronoi polygons is quite limited. Therefore we investigate the impact two different libraries built upon WebGL have on performance. The components

that will be measured are CPU, RAM, and GPU, which are the main components regarding performance. When the said components have been measured, the customer will get a more educated opinion on which library should be used to retain as good performance as possible, making the 3D part of the system more accessible for more customers, since it will be able to run on slower machines under heavy load.

1.4 Motivation

The main reason for doing the said research with WebGL and the selected libraries performance on 2D to 3D conversion is to give the client a better perspective on what library should be selected when performance is one of the main concerns. The information regarding conversion of 2D to 3D space in WebGL whilst using Voronoi islands as a representation of statistics is non-existent. Therefore an experiment with Three.js and BabylonJS (See reason for libraries in section 2.1) will be performed in order to analyze if one of the libraries performs better than the other with the said task.

1.5 Objectives

O1	Generate a set of graphs with different sizes that are to act as test data.
O2	Convert the test data into the suitable format for the different libraries.
O3	Implement rendering functionality for the different libraries.
O4	Implement a stress test for the libraries.
O5	Implement automated testing environment that will also record relevant performance usage.
O6	Reflect and evaluate the results after the stress tests have been executed.

Table 1.1: Diagram explaining experiment execution flow

The reason for objective one is to make the testing reflect a real-world case since the different worlds will have controlled variations when running the tests. The purpose of the second objective is to make sure the data is in the correct format when the different libraries and frameworks are executed in the stress test. The goal for the third objective is to have two similar applications, one for BabylonJS and one for Three.js with the same

functionality. Objective four is tightly coupled with the third objective since the stress test is implemented inside the BabylonJS and Three.js applications, and its goal is to ensure that we have a testing environment. Objective five is to implement a third application which is responsible for running, measuring and to store the test data. The final objective is to analyze the data and to draw conclusions based on the data acquired from the different tests. The data captured by the third application is used and hopefully, the data is informative enough to educate the customer regarding the decision of a library or framework.

1.6 Scope

Since it is impossible to evaluate all existing WebGL libraries in this thesis, only two have been chosen. There is a large number of different libraries available, therefore we have compared a group of libraries by their popularity on GitHub and have decided with the current sets of possibilities, that ThreeJS and BabylonJS are two of the most popular WebGL libraries. Since the authors of this report do not possess any prior knowledge about WebGL tools or libraries/frameworks. The different popularity measures included was GitHub stars, contributors, and forks. More precise numbers about the libraries can be found in the Method section of this report.

1.7 Target group

The thesis target group is the community that exists within the domain of WebGL and statistics, how to combine the two and construct software which can handle 3D visualization when it comes to performance. Another group is JavaScript developers who would like to learn more about which libraries to use when it comes to WebGL.

1.8 Outline

The remaining sections of this report include further information about the approach, results, analysis, and discussion. The *Method* section gives the reader a thorough explanation about how the experiment was conducted, this includes the device used for running the experiment and an explanation about the different implementations that was used for the experiment. Next is the *Implementation* section, which we provide the reader with useful information about the actual implementations, the reason for this is to maximize the

information required for reproducing the experiment. The fourth section is the *Result*, where the results of the experiment is presented. After the result, an analysis was performed on the results to understand if the results were significant or not. At last, a small discussion is presented and future research presented.

2 Method

This section will introduce the approach and describe the method this report used in order to try to evaluate how two libraries and frameworks compare against each other.

2.1 Approach

For this experiment, two libraries and frameworks are required. In order to find and evaluate candidates for this, a search process was conducted. To limit and simplify the decision making regarding candidates, a set of criteria was specified that was expected to be fulfilled. When the candidates had been selected, a set of applications were implemented. This included applications in relation to the different libraries/frameworks and an additional application that was responsible for running, measuring and recording the benchmarks.

2.2 Description

As mentioned in the previous section, the first task was to find suitable libraries or frameworks that were to be used in the experiment. In order to find suitable candidates, the authors performed a minor search. Included in this search was a set of criteria, the criteria can be read below:

- *The libraries or frameworks implementation and source code must be available (Open Source) for anyone to view.*
- *The libraries/framework must be licensed in such way it is free and can be used for commercial purposes.*
- *The libraries/framework must have been updated in the last week when the candidate library/framework was found in order to ensure that it is actively maintained.*

The keyword(s) that was used in order to try to find suitable candidates was “*WebGL frameworks and libraries*”. During the search process, we found a comprehensive list [20] containing a number of different WebGL libraries and frameworks. From this list, there were multiple libraries and frameworks that met the criteria stated above. This resulted that additional popularity criteria had to be included in the selection process. The popularity criteria were based on the measurements that can found and connected the GitHub

repository. These are, stars, forks, and contributors.

After the selection process regarding libraries/frameworks was done, a set of applications was implemented. The applications implemented included applications related to the library and framework that had been selected. Additionally, a third application was implemented that was responsible for running, recording and executing the benchmarks and performance measurements for all libraries and frameworks that was included in the experiment. During the execution process, data were collected that included information regarding performance metrics. This data will then be presented in tables, graphs. The data will then be summarized and analyzed. To ensure the validity of the experiment, the benchmarking and performance measure was executed multiple times. This was done to evaluate if the different executions of the experiments were disturbed or affected by background processes.

2.3 Machines and Devices

In *section 3, Implementation*, it is described that the experiment is limited to an Intel-based GPU due to time constraints and also due to the lack of other devices. Therefore, the experiment will be conducted on one machine, and below is a more detailed description of the said machine.

Device

Maker/Model	<i>Asus UX305UA</i>
Operating System	<i>Ubuntu 16.04 (Xenail)</i>
Graphics	<i>Intel HD Graphics 520</i>
Processor	<i>Intel Core i7-6500U 2.5GHz x 4</i>
Memory	<i>8GB</i>

Figure 2.1: Diagram explaining experiment execution flow

2.4 Reliability and Validity

There are some artifacts to take into consideration when evaluating the final result. Due to limited access regarding hardware, the performance recorder has only been implemented to target Intel GPUs. Therefore this essay can only present information regarding how the two different tools perform on Intel GPUs. Another important perspective to take into consideration is software disturbance. The disturbance in reference is background processes on the machine running the test. In order to keep the disturbance at a minimum, only the most relevant applications will be allowed to run on the system at the time the stress test will be conducted. In order to remove possible disturbances on the system and try to verify the results, multiple performance tests will be run on the machine.

In order to maximize the reproducibility of the experiment, every relevant information will be included and available for external readers. This includes hardware specifications, and software versions.

3 Implementation

In this section the different implementations are presented, this includes, for example, third-party applications versions and how they are relevant for this experiment.

3.1 Applications

The following applications will be used when developing the autonomous experiment.

3.1.1 Performance Recorder

The performance recorder measures and stores different metrics that were in the background and is written in JavaScript. Since this is a program that will not run in the browser, a runtime environment is relevant in order to execute the performance recorder program. For this specific task, Node.js¹ is used. The version used during the testing and development process of the application was the current LTS² version which at the time of writing was version 8.9.4.

In order to enable this application to record the underlying system usage, launch and kill relevant applications, a set of third-party applications was used. These applications are listed below.

Opn (version 5.2.0)

Opn[15] is a cross-platform software that allows programs to open applications inside node.js processes. The performance recorder uses this to start applications where the BabylonJS and Three.js applications are executed.

systeminformation (version 3.37.3)

systeminformation [16] is an open source software that allows programs to access system usage from a node.js process. The performance recorder utilizes this in order to access information about CPU usage, RAM usage, and GPU usage.

¹ Node.js - <https://nodejs.org/en/>

² LTS - Long Term Support - https://en.wikipedia.org/wiki/Long-term_support

tree-kill (version 1.2.0)

tree-kill [17] is a software that allows programs to kill processes in the process tree. The performance recorder utilizes this in order to kill a specific application when a new one is to be launched before starting a new test.

3.2 Libraries and Frameworks

This section contains information regarding the framework and library choices which were made.

3.2.1 BabylonJS

BabylonJS [18] is one of the frameworks and libraries that this project will use as a candidate for the performance evaluation. The authors have selected to include this in the experiment since this framework passed all the stated selection criteria (*see criteria specification in section 2.2*). It was also one of the frameworks that scored the highest based on the popularity criteria specified (*see criteria specification in section 2.2*). On GitHub³, BabylonJS had 6307 stars, 1403 forks and 178 contributors at the time of writing. In the experiment version, 3.2.0-alpha6 was used.

3.2.2 Three.js

Three.js[19] is the other candidate that we are to use for the experiment. This library has been chosen on the same terms that were presented for the previous candidate (*see 3.2.1*). When this section was written, Three.js possessed 40,112 stars, 14,931 forks and had 924 contributors on GitHub³. At the time when the experiment was conducted and during the development process of the experiment artifact version 0.89.0 was used.

³ GitHub - <https://github.com/>

3.3 Benchmarking

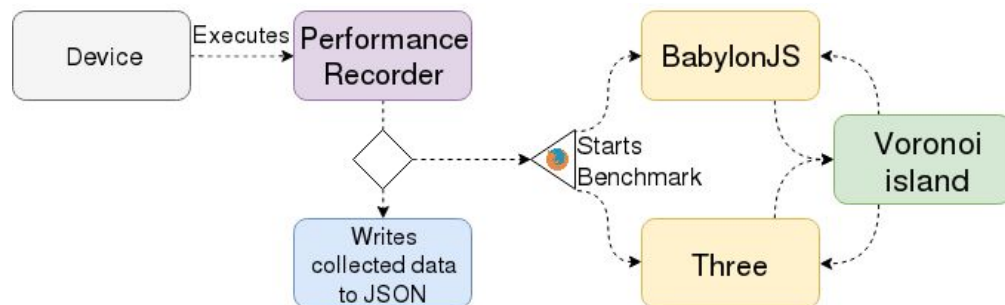


Figure 3.1: Diagram explaining experiment execution flow

As mentioned earlier in this report, a stress test was conducted in order to evaluate how two libraries and frameworks perform compared to each other. An example flow of an execution goes as follows:

- The performance recorder script is started with a set of commands which either runs the tests on the BabylonJS or on the Three.js implementation.
- Based on these instructions, it opens a browser selected by the script on a specific page where the selected libraries and frameworks implementation are running.
- When the application is started, the performance recorder script also provides the implementation with the appropriate heightmap. The heightmaps are shared on both platforms since the testing should be conducted on the same data and under the same circumstances.
- When the browser has opened the testing environment where either the BabylonJS or Three.js application is running and the stress test starts. When the performance recorder has started the browser, the scripts starts to measure performance load.
- The data collected from the performance measurements are written to a set of files. The stress test runs for a total of 60 seconds for each height map and the performance recorder reads benchmark-metrics every 3 seconds.
- During the stress test, a 3D camera follows a predefined path at a rapid pace in order to put as much pressure as possible on the system.
- When the stress test is done, the browser is shut down, and the process is restarted with a new heightmap as the target.

- This process is repeated until all heightmaps have been tested for the specific platform.

4 Results

The following data which is presented is the result of executing the tests described above. Each table consists of three graphs, whereas one represents information regarding GPU, one describes the RAM performance, and the third visualizes the libraries impact on the CPU. These graphs display the summary of the sets of data which was gathered.

4.1 Explanation of the Results

This section will explain what the tables below include and how they are relevant for the purpose of this project.

Ten different tables are presented below, and each table includes the result collected from the executed stress tests. Each collection of cells represents test data for a single map. A map in this context is a collection of Voronoi polygons that represents one or more islands. All maps used in the experiment can be found under the section *appendices* as Appendix 1. Additionally, all data that was used for summarizing the tests and results can be found in the *appendices* section as Appendix 2.

4.1.1 GPU

In the tables GPU cell, the different metrics in relation to the GPU are presented. The bar-chart inside the GPU cell includes information regarding both Babylon and Three displayed next to each other in order to see the performance difference between the two.

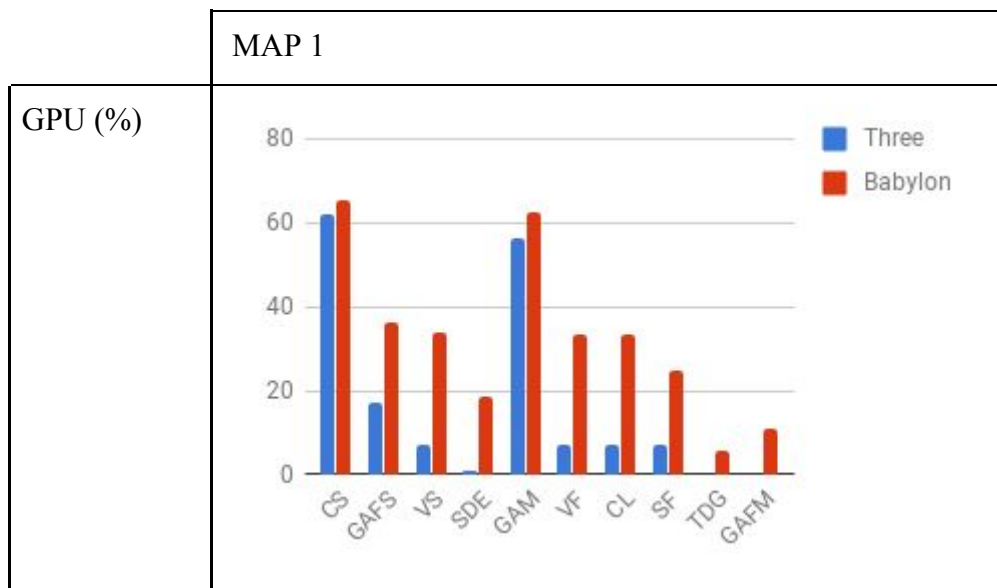
The data that represents the GPU is visualized as bar-charts and describes the amount of used memory at an average for each map. The data that is presented represents the average GPU metric usage for ten runs with eight different measure points. For further clarification, each map was run ten times and data were collected during these runs at eight different times. The table cells below presenting the GPU data includes a set of acronyms. These acronyms represent a metric regarding the GPU load. Since this report is not about the architecture of a graphics card, only what the acronym stands for will be presented and not an explanation about the acronym. The acronyms can be found in Appendix 3.

4.1.2 CPU

Inside the CPU cell, the metrics are presented regarding the CPU usage during the experiment. Similar to the GPU cell, this includes both Babylon and Three data. The data is presented in the same graph displaying the difference between the two. The data that is visualized inside the graph represents the average CPU-usage percentage for all runs during the specific map.

4.1.3 RAM

Inside this cell, the RAM metrics are presented that was gathered during testing. This data also includes data relevant for both platforms. The numbers in the bar-chart regarding the RAM metrics should be interpreted with GB (gigabyte) metric in mind. The different bars inside the bar chart represents the average usage och all specified metrics for ten tests per image and at eight different times of measurements.



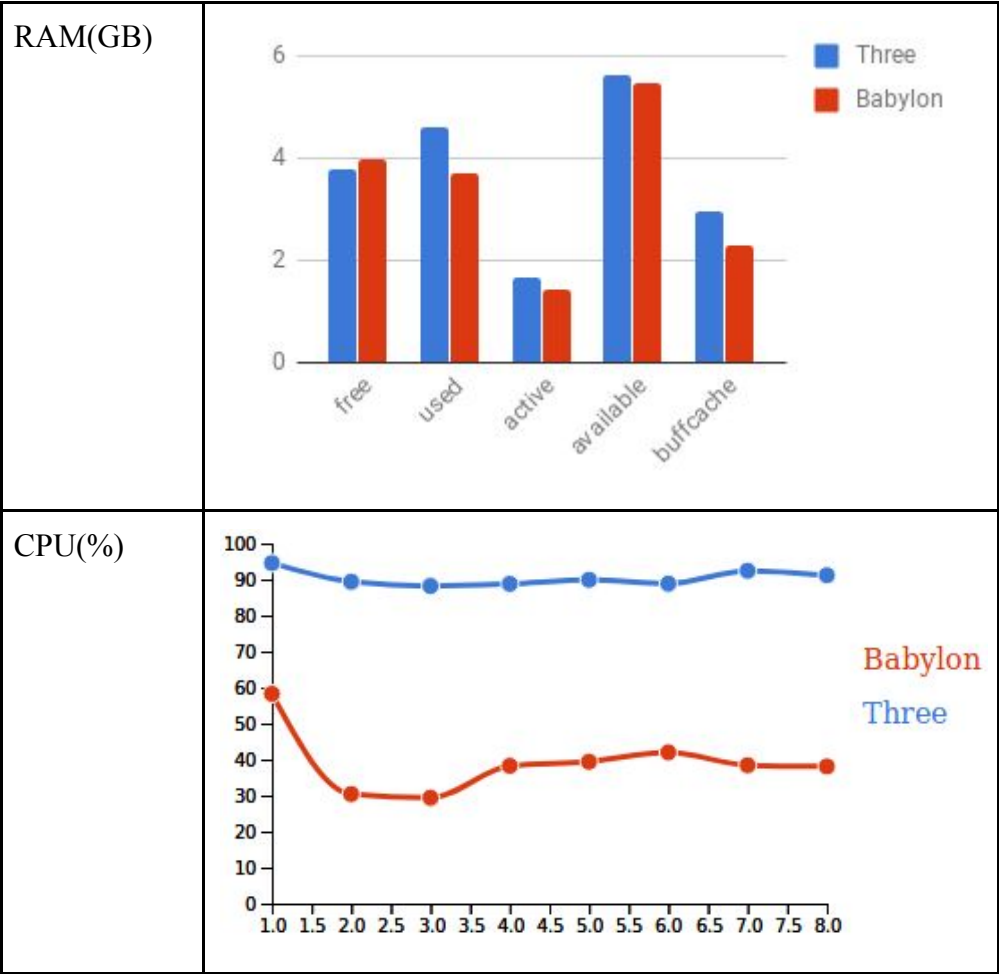


Table 4.1: Data representation of Map 1.

MAP 2

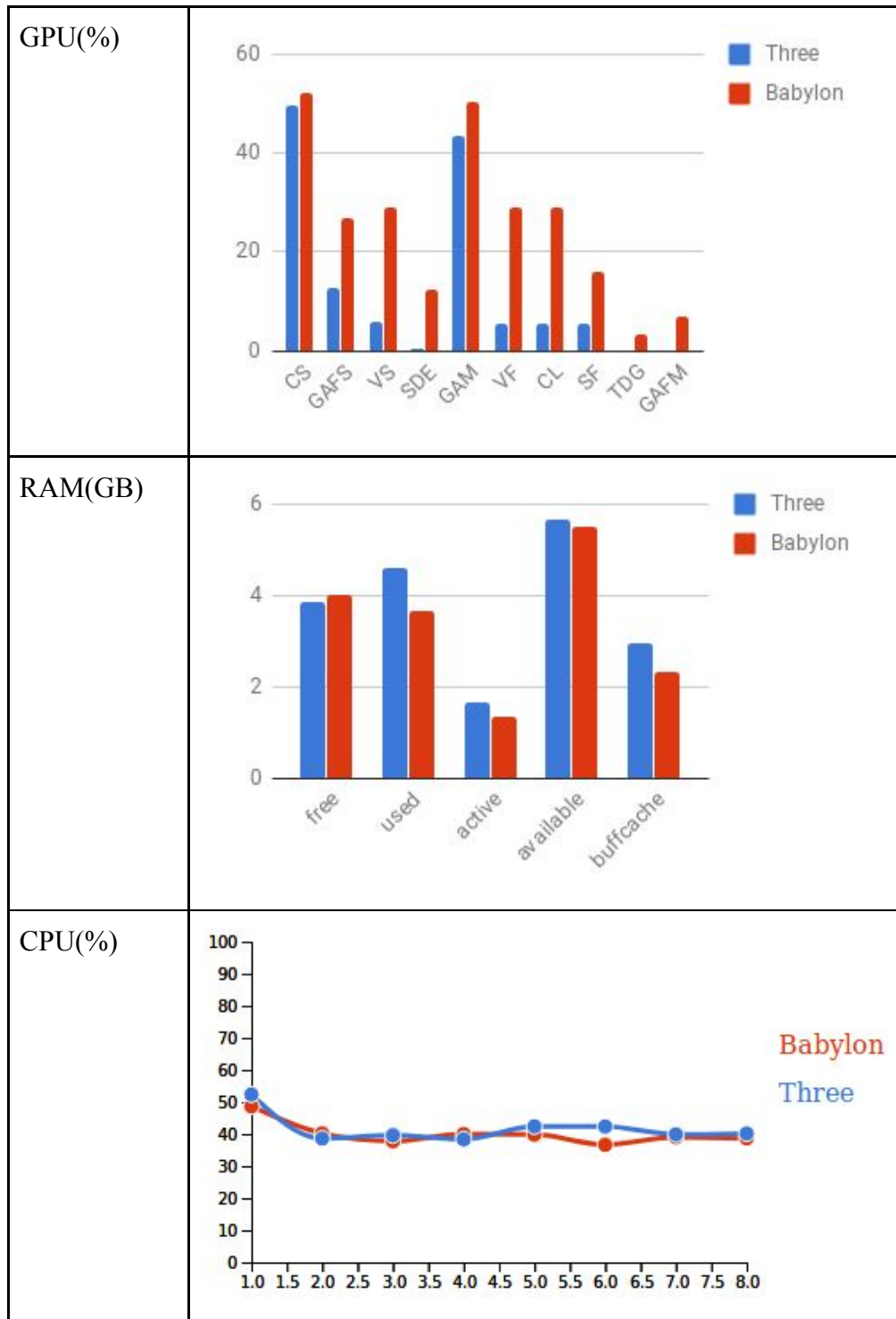


Table 4.2: Data representation of Map 2.

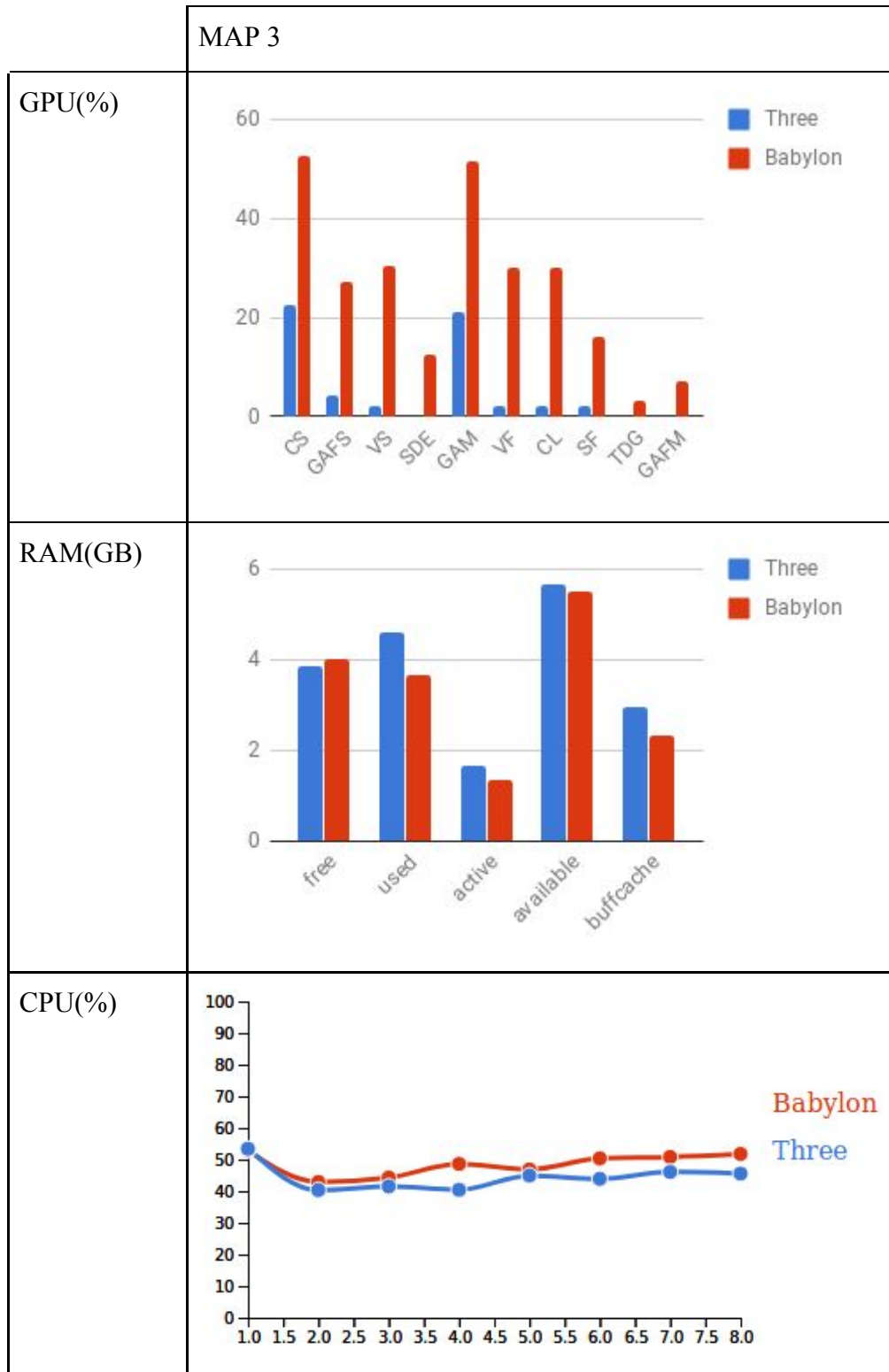
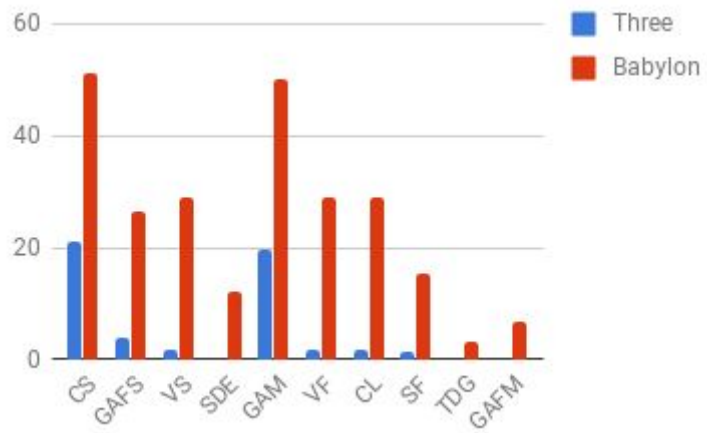


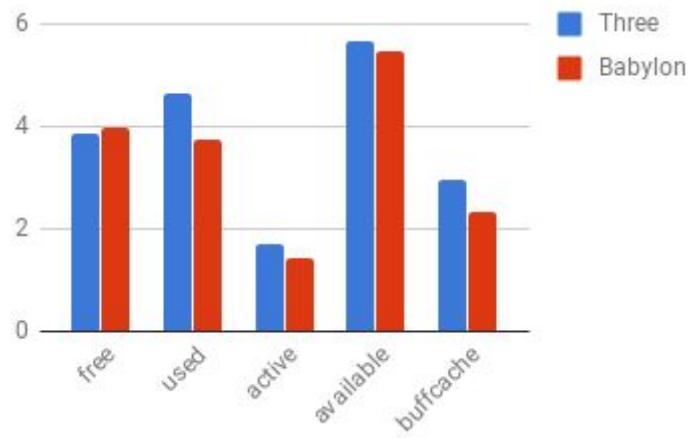
Table 4.3: Data representation of Map 3.

MAP 4

GPU(%)



RAM(GB)



CPU(%)

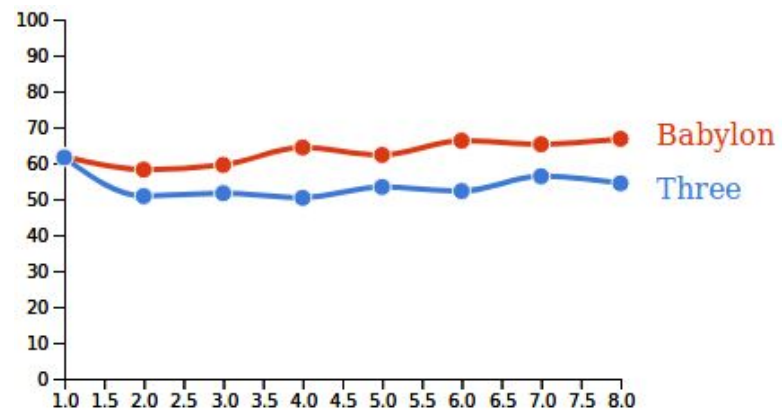


Table 4.4: Data representation of Map 4.

	MAP 5																																	
GPU(%)	<p>A bar chart comparing GPU usage (%) for 'Three' (blue) and 'Babylon' (red) across ten categories: CS, GAFS, VS, SDE, GAM, VF, CL, SF, TDG, and GAFM. The y-axis ranges from 0 to 50. Babylon generally shows higher GPU usage than Three, with peaks in CS and GAM.</p> <table><tr><th>Category</th><th>Three (%)</th><th>Babylon (%)</th></tr><tr><td>CS</td><td>18</td><td>49</td></tr><tr><td>GAFS</td><td>2</td><td>25</td></tr><tr><td>VS</td><td>1</td><td>27</td></tr><tr><td>SDE</td><td>0</td><td>11</td></tr><tr><td>GAM</td><td>17</td><td>48</td></tr><tr><td>VF</td><td>1</td><td>27</td></tr><tr><td>CL</td><td>1</td><td>27</td></tr><tr><td>SF</td><td>1</td><td>14</td></tr><tr><td>TDG</td><td>0</td><td>3</td></tr><tr><td>GAFM</td><td>0</td><td>6</td></tr></table>	Category	Three (%)	Babylon (%)	CS	18	49	GAFS	2	25	VS	1	27	SDE	0	11	GAM	17	48	VF	1	27	CL	1	27	SF	1	14	TDG	0	3	GAFM	0	6
Category	Three (%)	Babylon (%)																																
CS	18	49																																
GAFS	2	25																																
VS	1	27																																
SDE	0	11																																
GAM	17	48																																
VF	1	27																																
CL	1	27																																
SF	1	14																																
TDG	0	3																																
GAFM	0	6																																
RAM(GB)	<p>A bar chart comparing RAM usage (GB) for 'Three' (blue) and 'Babylon' (red) across five categories: free, used, active, available, and buffcache. The y-axis ranges from 0 to 6. 'Three' generally has higher RAM usage than 'Babylon' in the 'free' and 'used' categories, while 'Babylon' is higher in 'available'.</p> <table><tr><th>Category</th><th>Three (GB)</th><th>Babylon (GB)</th></tr><tr><td>free</td><td>3.8</td><td>4.0</td></tr><tr><td>used</td><td>4.6</td><td>3.7</td></tr><tr><td>active</td><td>1.7</td><td>1.4</td></tr><tr><td>available</td><td>5.6</td><td>5.4</td></tr><tr><td>buffcache</td><td>3.0</td><td>2.3</td></tr></table>	Category	Three (GB)	Babylon (GB)	free	3.8	4.0	used	4.6	3.7	active	1.7	1.4	available	5.6	5.4	buffcache	3.0	2.3															
Category	Three (GB)	Babylon (GB)																																
free	3.8	4.0																																
used	4.6	3.7																																
active	1.7	1.4																																
available	5.6	5.4																																
buffcache	3.0	2.3																																

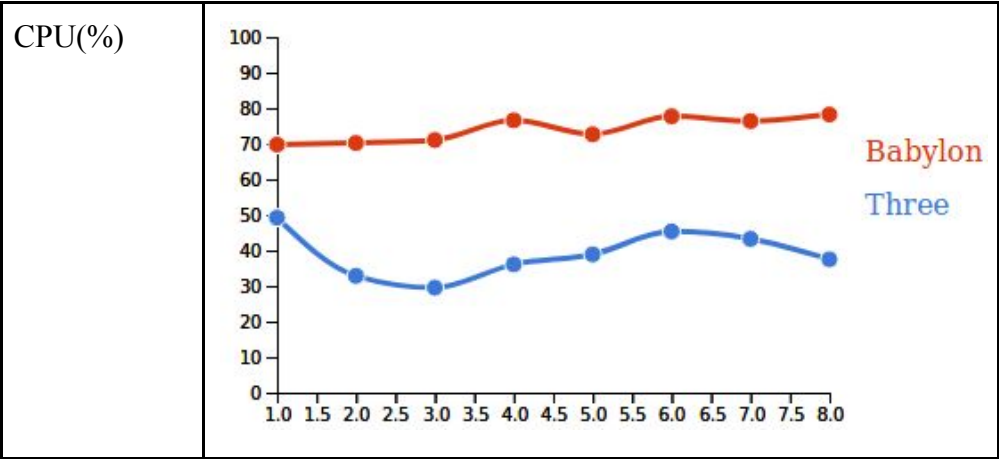
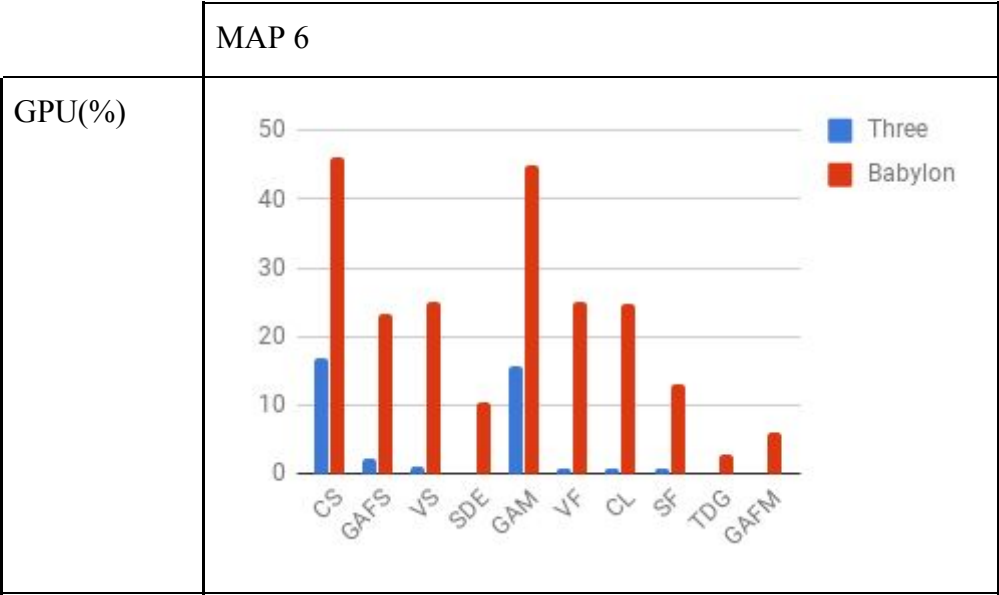


Table 4.5: Data representation of Map 5.



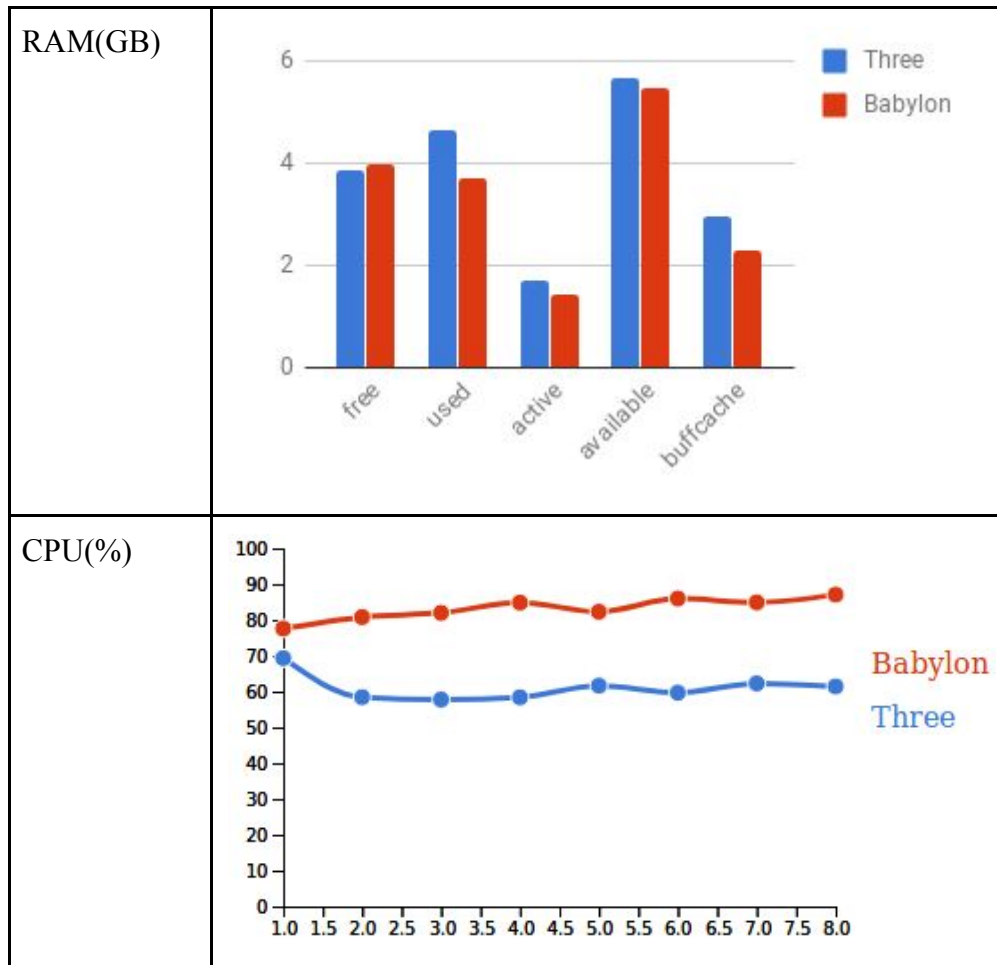
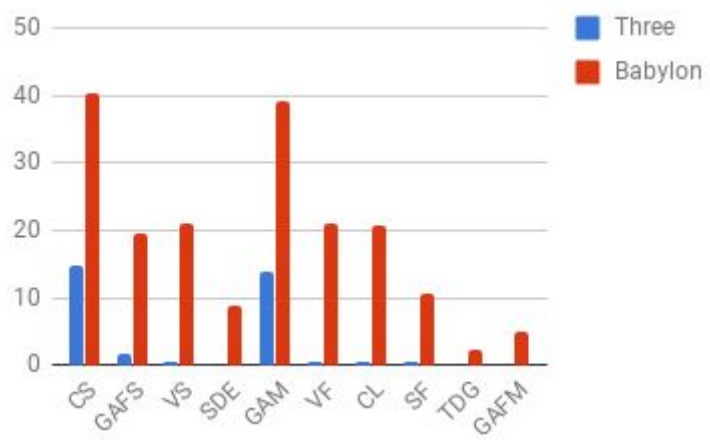


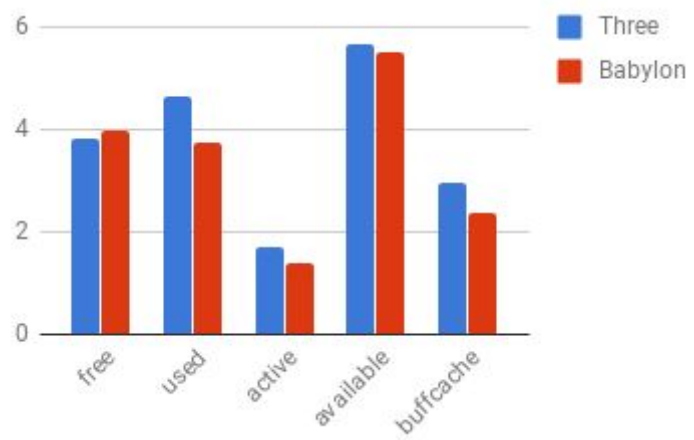
Table 4.6: Data representation of Map 6.

MAP 7

GPU(%)



RAM(GB)



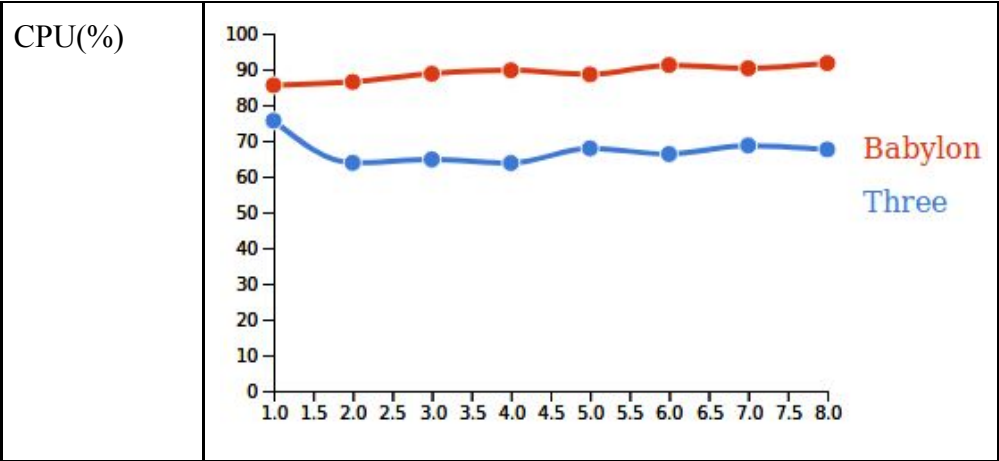
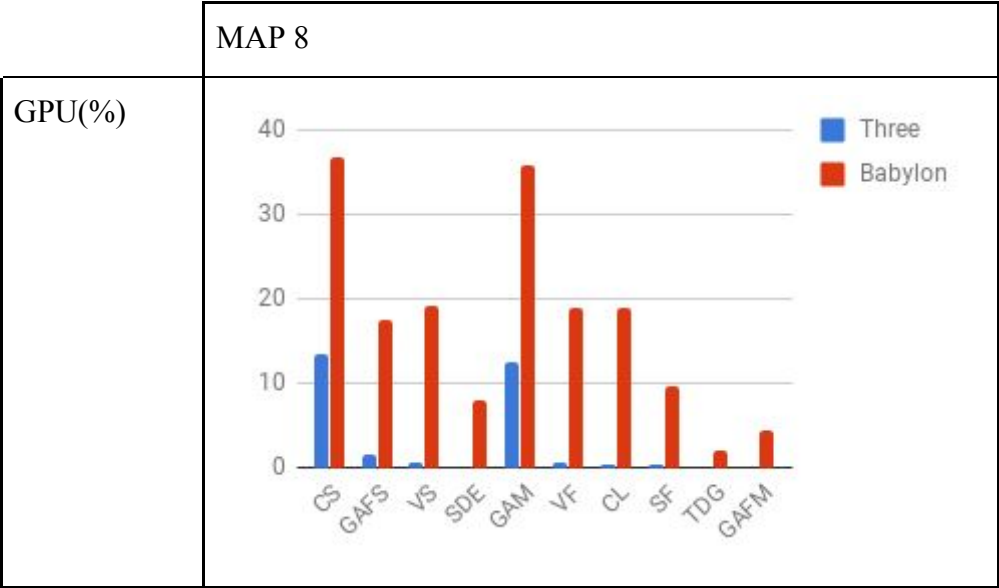


Table 4.7: Data representation of Map 7.



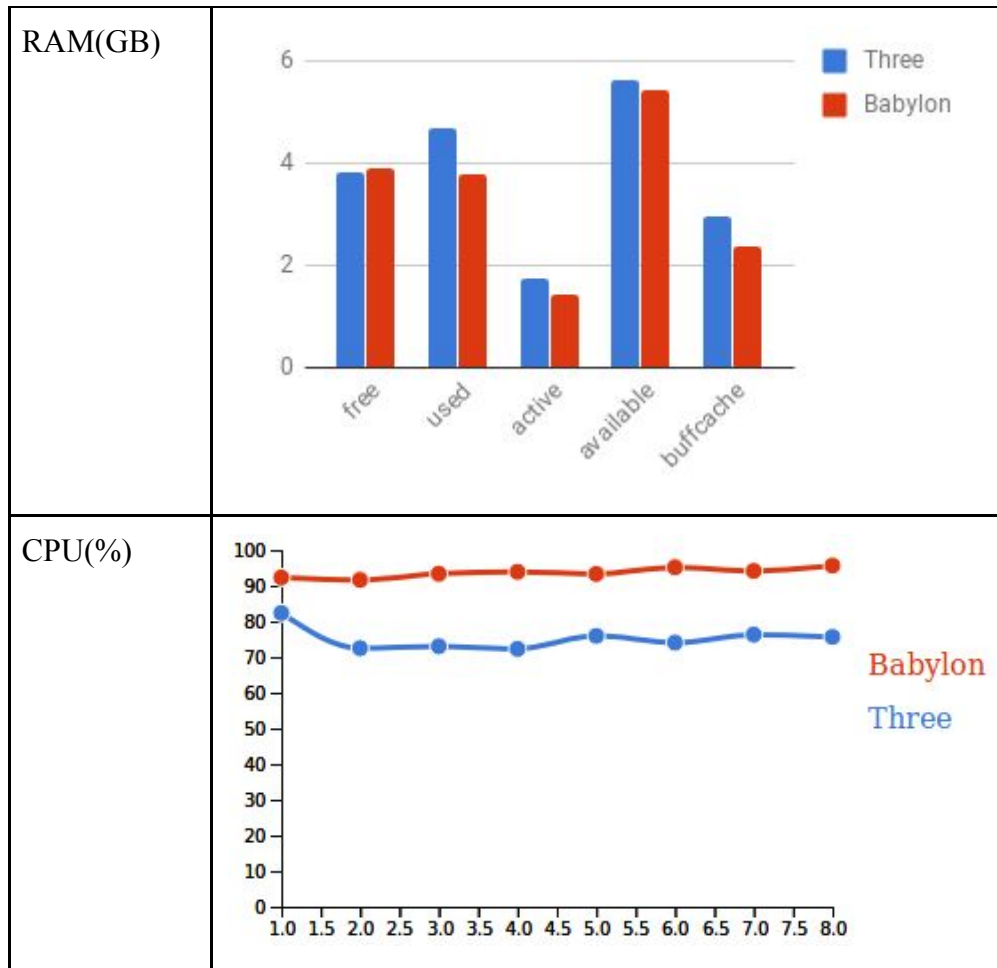
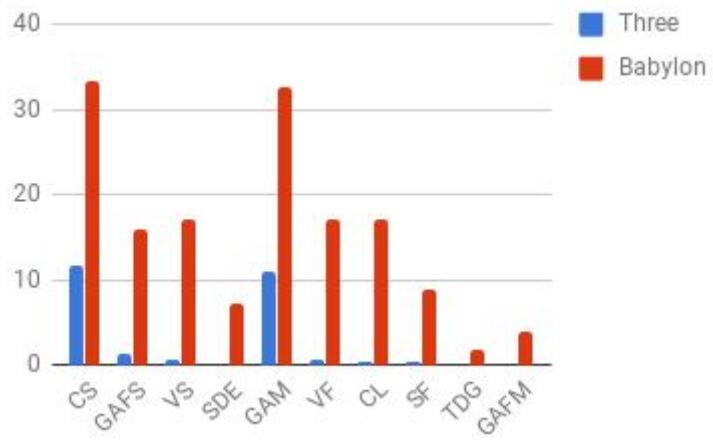


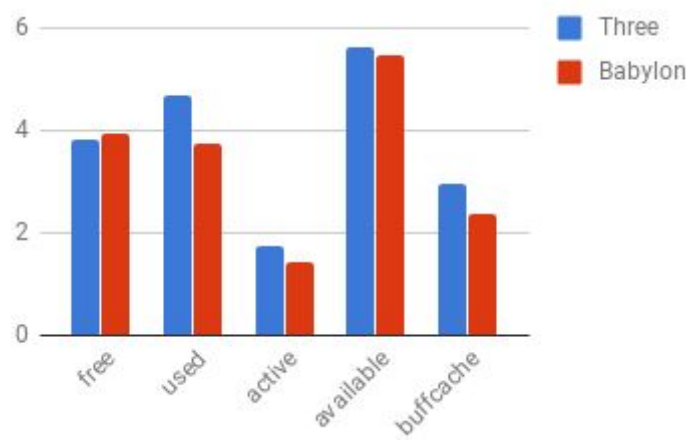
Table 4.8: Data representation of Map 8.

MAP 9

GPU(%)



RAM(GB)



CPU(%)

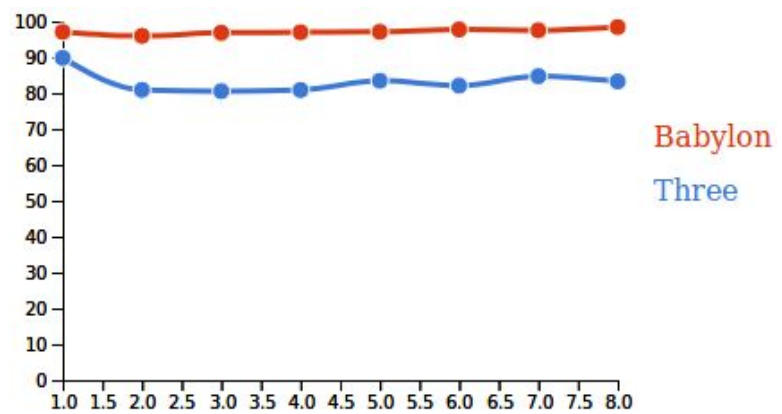
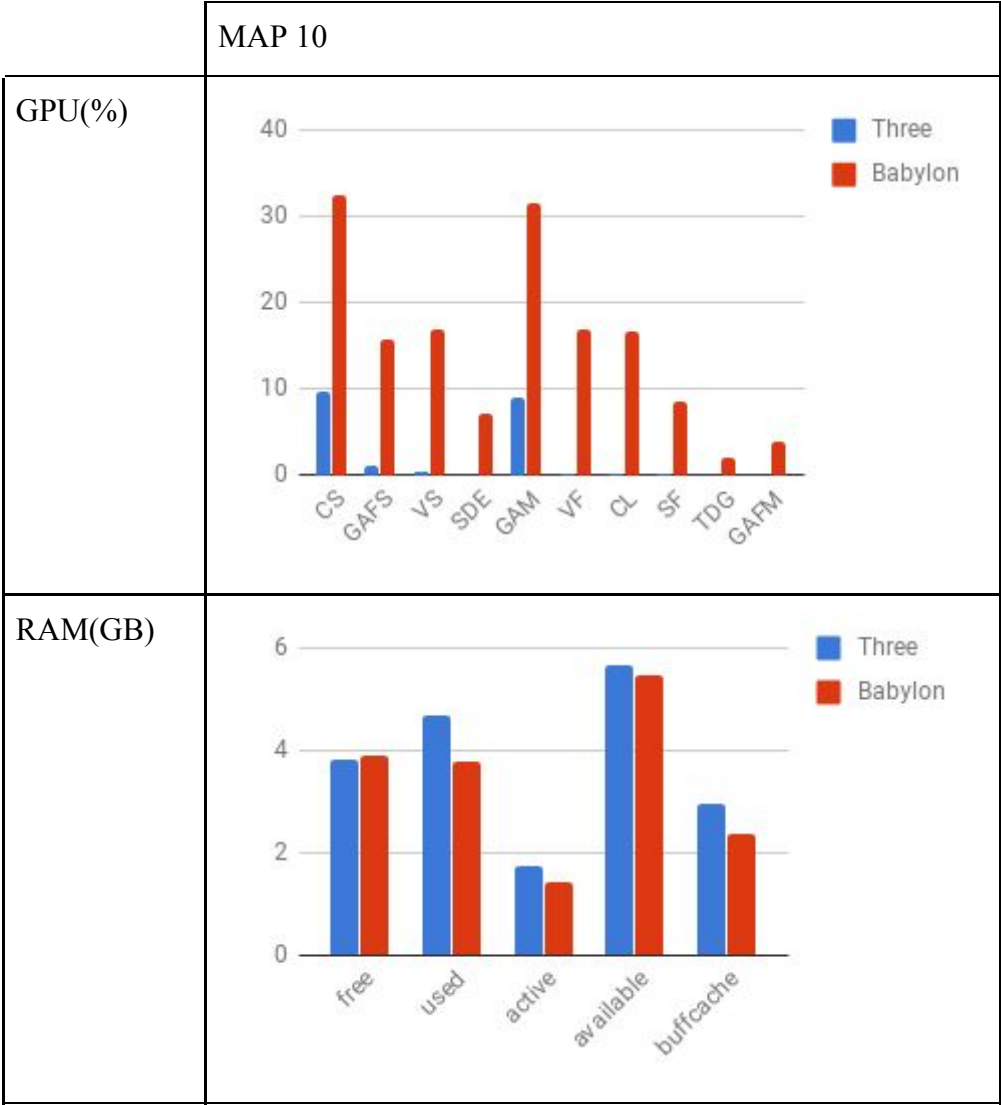


Table 4.9: Data representation of Map 9.



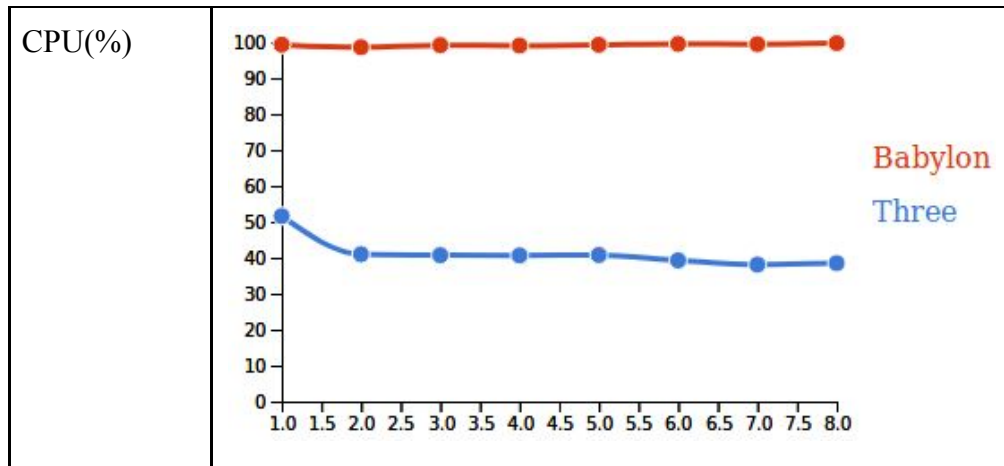


Table 4.10: Data representation of Map 10.

5 Analysis

This section of the report presents an in-depth analysis of the results presented in the previous section.

5.1 GPU

The GPU data is very interesting for this type of testing since WebGL is mainly run by the GPU. It can be seen in the results that there is a difference, between the BabylonJS and Three.js results. To make sure that there is a relevant difference that is worth taking into consideration, statistical testing needs to be done. The first approach was to perform a T-test, which requires the dataset to be normally distributed. The histogram graphs are shown in Figure 5.2 and 5.3 below, indicates that the dataset is not normally distributed and therefore another test more suitable for non-normally distributed data was used, namely Wilcoxon Rank Sum Test [21]. To simplify this process, an application called JStat [22] was used. This application simplifies the process of conducting a Wilcoxon Rank-Sum test, which will determine whether there is a statistical difference between the two datasets. The result from the said test is presented in Figure 5.1. To enable replication of the test, the dataset used in the Wilcoxon test can be found in *Appendix 2*. The result from the Wilcoxon test, shown in Figure 5.1, shows that there is a statistically significant difference between the two and that the Tree platform performed better in better in terms of GPU performance. This proves that the difference presented in the previous section is significant and can be taken into account if GPU usage is an important factor when deciding which framework to use.

Wilcoxon Rank-Sum test

Dataset	N	Mean	Stdev
Three	100	5.93	11.417
Babylon	100	22.24	15.136

Note: Approximation using Normal Distribution

Significance level (P-value):	0.000 ($\alpha = 0.05$)
Result:	There is a significant difference between the datasets

Figure 5.1 - Wilcoxon Rank Sum test show that there is a significant difference

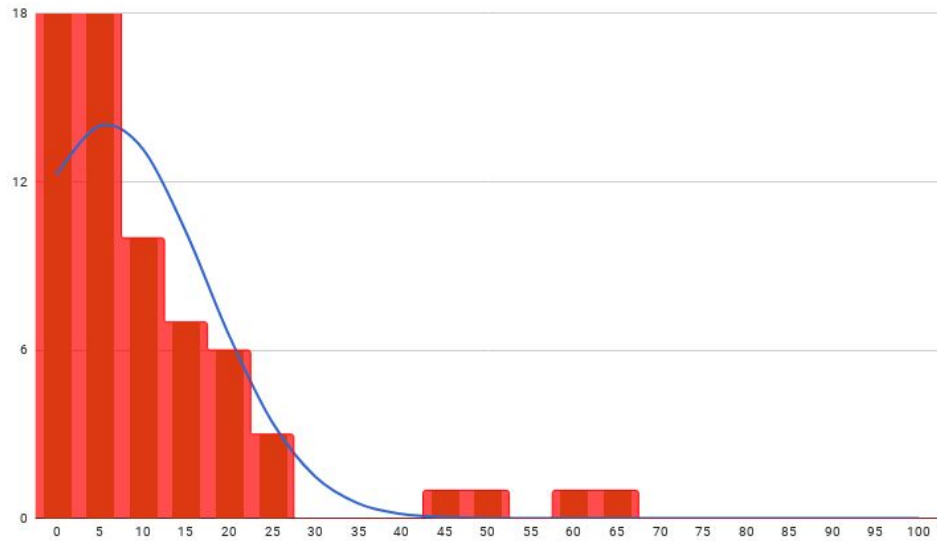


Figure 5.2 - CPU Data is not approximately normally distributed (Three)

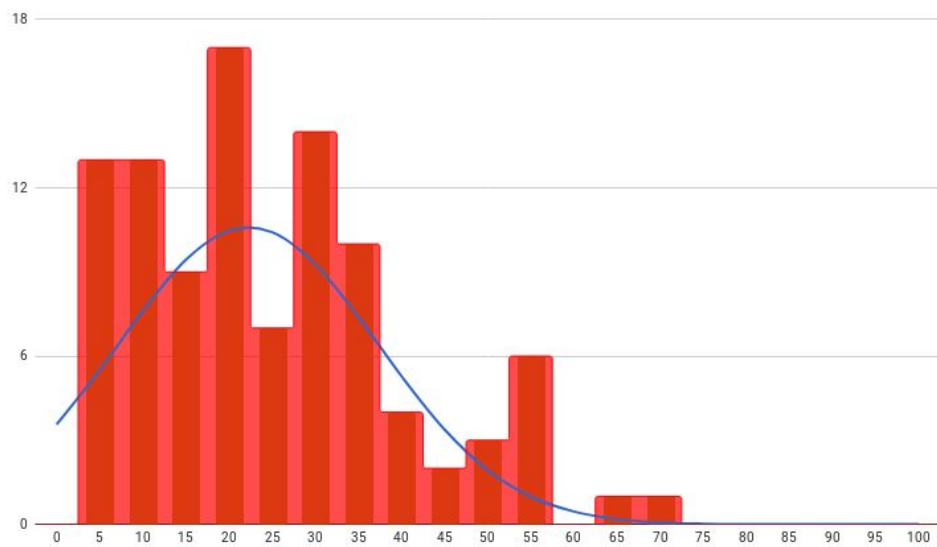


Figure 5.3 - CPU Data is not approximately normally distributed (Babylon)

5.2 CPU

The CPU data, in Figures 4.1 to 4.10 shows that in general there is a lower CPU load when running the Three application. The graphs above are interesting since they show that there is a visual difference. Even more interesting is to evaluate whether the difference is significant enough that it should be taken into consideration when choosing between Babylon and Three.

Similar to the GPU result, the P-value was calculated in order to try to evaluate whether there is a significant statistical difference between Three and Babylons impact on the CPU performance. The first approach to find the P-value was to evaluate whether the data were normally distributed and then perform a T-test in order to get the P-value. However, the CPU data for Babylon was not normally distributed, whereas the CPU data for Three was. Since normally distributed data is a required, and Figures 5.5 and 5.6 shows that the data is not normally distributed, a Wilcoxon Rank Sum test [21] was used. The P-value must be below 0.05 in order to prove that there is a significant statistical difference. More specifically, the P-value presented was 0.000 rounded to three decimal points, therefore CPU can be taken into account when choosing between Babylon and Three. Figure 5.4 is a screenshot from the JStat [22] application and more specifically the results of the Wilcoxon Rank Sum test. The figure also shows that there is a significant statistical difference in favor of Three. The dataset that was used when running this test can also be found in the *Appendix section* as Appendix 2.

Wilcoxon Rank-Sum test

Dataset	N	Mean	Stdev
Three	80	59.77	18.265
Babylon	80	72.73	22.739

Note: Approximation using Normal Distribution

Significance level (P-value):	0.000 ($\alpha = 0.05$)
Result:	There is a significant difference between the datasets

Figure 5.4 - Wilcoxon Rank Sum test show that there is a significant difference

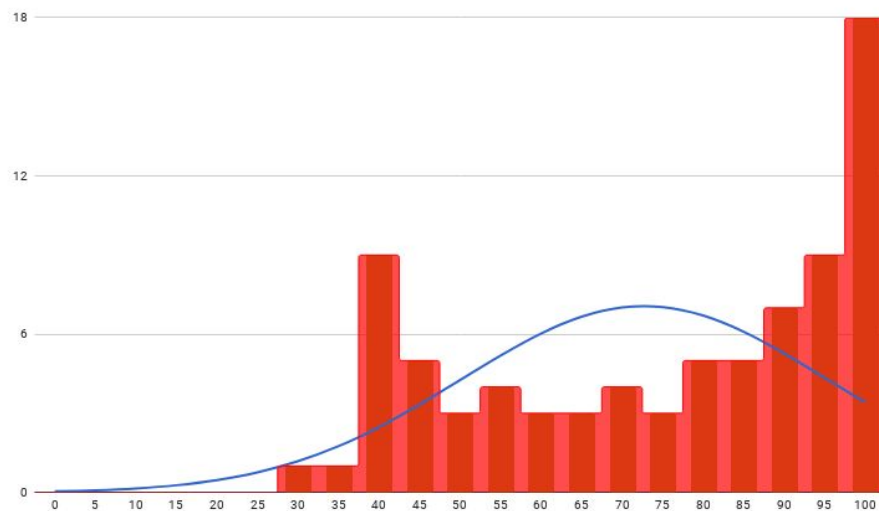


Figure 5.5 - Graph Displaying CPU Data is not normally distributed (Babylon)

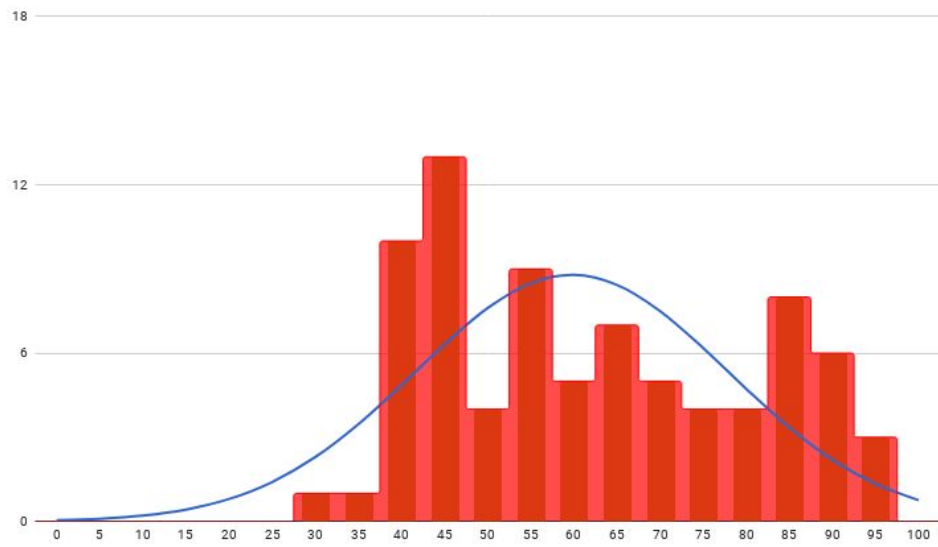


Figure 5.6 - Graph displaying CPU Data is approximately normally distributed (Three)

5.3 RAM

For the RAM data collected in the experiment, the same procedure was used as the previous metrics. When looking at the RAM data from the result section, one can see that there is a minimal difference between the two which makes it even more important to conduct a statistical test. The dataset that was included in the test can be found in the *appendix section* as appendix 2. As mentioned in the previous sections, a P-value is required to be extracted in order to find out if there is any significant difference between the two applications. The first approach that was meant to be used was a T-test. As mentioned in the GPU and CPU sections, the data must be normally distributed since it is a requirement for a T-test. Presented below, is two graphs that indicate that the data is not normally distributed. This means that this also requires the Wilcoxon test to determine the P-value. The JStat [22] application was used and the Wilcoxon Rank Sum Test. Below, *Figure 5.7* shows that there is no significant difference between the two which means that the difference presented in the result section is not particularly relevant to include in the decision when choosing between the two.

Wilcoxon Rank-Sum test

Dataset	N	Mean	Stdev
Three	50	3.76	1.379
Babylon	50	3.38	1.419

Note: Approximation using Normal Distribution

Significance level (P-value):	0.085 ($\alpha = 0.05$)
Result:	There is no significant difference between the datasets

Figure 5.7 - Wilcoxon Rank Sum test show that there is no significant difference

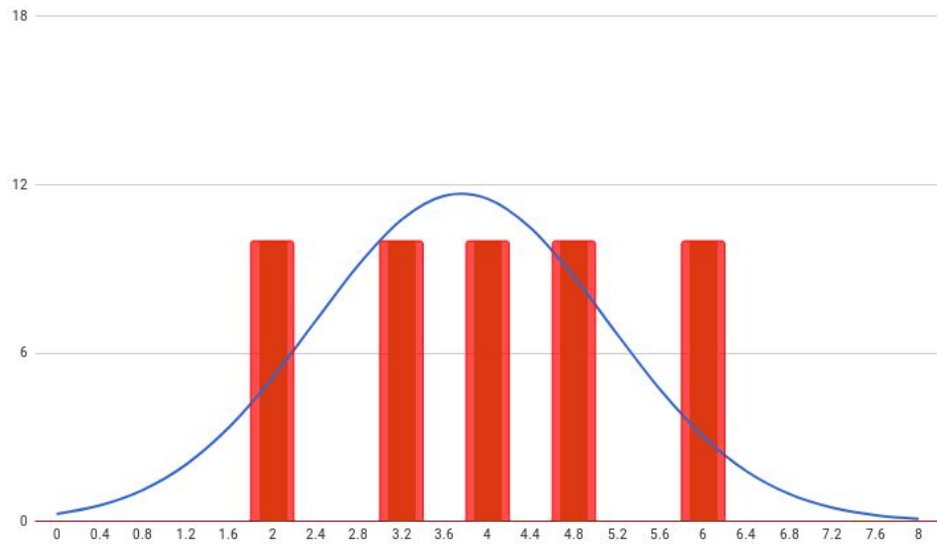


Figure 5.8 - RAM Data is not approximately normally distributed (Three)

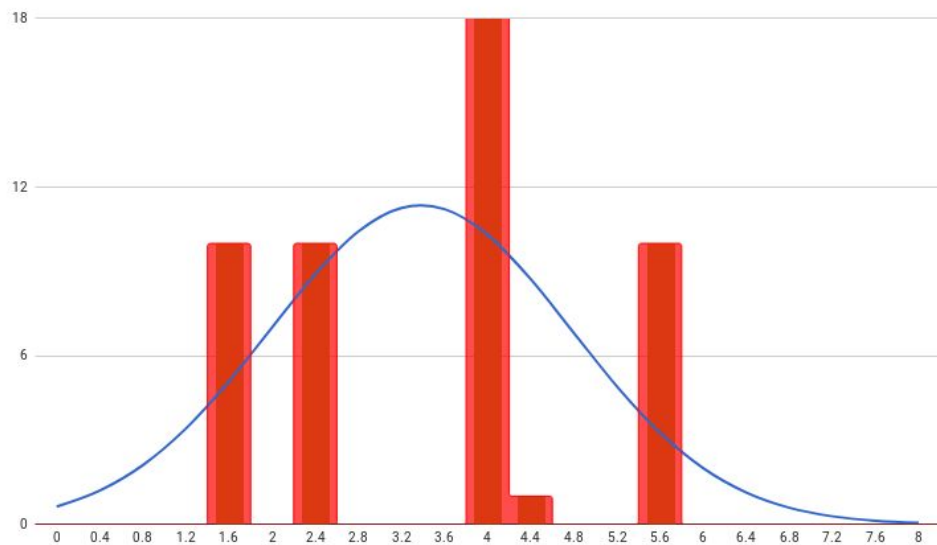


Figure 5.9 - CPU Data is not approximately normally distributed (Babylon)

5.4 Summary

The data that was gathered during the tests show that the library Three.js outperformed BabylonJS. The reason for this conclusion is the fact that Three.js used significantly less GPU- and CPU- power than BabylonJS with

the same performance. Three however used slightly more RAM than BabylonJS, but the difference is not statistically significant according to a Wilcoxon Rank Sum test. This was shown by comparing the collected data using the Wilcoxon Rank Sum test. The test showed whether or not the data collected was significant, which on the other hand proved that the GPU- and CPU-data was valuable.

6 Discussion

The data gathered and analyzed in the experiment in *chapter 5* proves the fact that Three.js outperforms BabylonJS significantly on all different tests except the RAM test. However, that specific part of the test was not statistically significant, which makes it obsolete. Therefore, Three.js is the library to choose if the only factor to take into consideration is performance. Since performance was important for the customer and the results showed that there was a significant difference between Babylon and Three, the choice between the two should be straightforward. Even though the results were straightforward and it is clear which library had the best performance, those facts could quickly change in a near future since both of the packages are still maintained and updated. The outcome could also be more varied in the future because of new and developed hardware which might not be adapted for one of the two libraries. That could result in either one of them to outperform the other in the future.

The previous research that is mentioned in *chapter 1.2* conforms to the results that were gathered during the experiment. Since the related works do not fully correlate with the contents of this thesis, some of the information in the mentioned works is redundant. *A. Muennoi* and *D. Hormde* states in their paper that WebGL has high performance for visualizing 3D and is a great tool for that purpose. The gathered data prove that both Three.js and BabylonJS performed well during the experiment without any stuttering and performance scaled linearly for more complex tasks.

M. Almashor and *I. Khalil* presented information about the performance impact regarding the Voronoi pattern. Since we had no performance issues with neither Three.js or BabylonJS, the results also further prove the point since our collected test data describes a linear performance. This means that their previous research also conformed with our results.

In short, Three.js outperformed BabylonJS significantly on all different metrics. This can, however, change at any time since the two libraries are still being maintained and updated. Different hardware might also favor one or the other library which could result in a different outcome. The data gathered conforms to the related research stated in *chapter 1.2*.

7 Conclusion

This section will present a short and informative conclusion about report overall.

Since the origin of this report derived from an idea that a user should be able to dive into a 3D world, our results and analysis are relevant for our customer. As mentioned in the discussion, Three.js outperformed BabylonJS on all measurements except RAM-usage. However, since the RAM difference was not statistically significant, this makes the RAM-usage metric obsolete. As mentioned in the *background* of this thesis, the main purpose of this report was to provide a educated opinion regarding WebGL libraries and frameworks if the customer chooses to use WebGL for the future vision of the application. Since this thesis was focused on performance, recommendations will be based on the results and conclusions presented in this thesis. Another interesting detail one might want to take into consideration even though it is not relevant for this report is the implementation for BabylonJS and Three.js. The authors of this report both come to the same conclusion that BabylonJS has a more beginner friendly API in relation to the more low-level Three.js. This might be of interest if there is a lack of implementation time.

Our results are not very varied since the testing was conducted on a single device only, and since the GPU was created by Intel. To make the results more interesting one might want to conduct the experiment on another GPU-platform, for example, AMD or Nvidia.

One aspect that could have been done better is the time planning regarding the implementation. As mentioned previously in this section, the Three implementation proved to be harder than expected. Even though Three proved to be the better candidate in regards to the performance metrics mentioned in this report, the difference could have possibly been even more significant if more time had been allocated to implement Three even more thoroughly.

To finalize this section, the results presented in the *Results* section have been analyzed, conclusions have been drawn based on the results with help of a statistical determination process in order to determine if there is any statistical significance. As presented in the analysis, there was a statistical significance in favor of Three.js.

7.1 Future work

As mentioned in the section above and in the *Method* section, only one machine was used to conduct the tests. To add additional validity to our results, additional experiments with more devices should be used for testing. Another interesting aspect that can be to test the difference frameworks but with focus on mobile devices instead of traditional computers. Since the framework and library selected for this essay were not the only two available, one can perform the same experiments but replace either BabylonJS and Three.js.

The most interesting future research is to replace the underlying worlds that were rendered and tested. The worlds used in these experiments were somewhat small, it can be of interest to for example double the size of the maps to understand if the size has any impacts on the results.

References

- [1] “The birth of the web”, home.cern, [Online] Available: <https://home.cern/topics/birth-web>, [Accessed Feb 12, 2018]
- [2] mfluehr, “transform”, developer.mozilla.org, Feb 5, 2018 [Online] Available: <https://developer.mozilla.org/en-US/docs/Web/CSS/transform> [Accessed Feb 12, 2018]
- [3] stephaniehobson, “Getting Started with WebGL”, developer.mozilla.org, Nov, 27, 2017 [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Getting_started_with_WebGL, [Accessed Feb 12, 2018]
- [4] K. Jang, Sangjin Han, Seungyeop Han, S. Moon and K S. Park, in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, 2011, pp. 1-14 [Accessed: Feb 12, 2018]
- [5] mcguffin, “The WebGL API: 2D and 3D graphics for the web”, developer.mozilla.org, Jan 27, 2018 [Online] Available: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API, [Accessed: Feb 12, 2018]
- [6] A. Dobrin, *A Review of Properties and Variations of Voronoi Diagrams*. whitman.edu. 1–43 (2005) <https://www.whitman.edu/Documents/Academics/Mathematics/dobrinat.pdf>, [Accessed: Feb 12, 2018]
- [7] D Sánchez-Gutiérrez, M. Tozluoglu, J D. Barry, A. Pascual, Y. Mao, L M. Escudero, “Fundamental physical cellular constraints drive self-organization of tissues,” *The EMBO Journal* vol 35, pp 77-88, 2016. [Accessed Feb 12, 2018]
- [8] F. Aurenhammer. “Voronoi diagrams - a survey of fundamental geometric data structure,” *ACM Computing Surveys (CSUR)* Vol 23, Issue 3, pp 345-405, Sept 1991. [Accessed: Feb 12, 2018]
- [9] “Intel Open Source HD Graphics, Intel Iris Graphics, and Intel Iris Pro

Graphics”, Volume 1: Preface. Intel. May 19, 2017. [Online]

Available:

<https://01.org/sites/default/files/documentation/intel-gfx-prm-osrc-skl-vol01-preface.pdf>, [Accessed: Feb 27, 2018]

[10] S. Junkins, “The Compute Architecture of Intel Processor Graphics Gen 7.5”, Version 1.0. January 8, 2014. [Online] Available: https://en.wikichip.org/w/images/f/f5/Compute_Architecture_of_Intel_Processor_Graphics_Gen7dot5_Aug4_2014.pdf, [Accessed: Feb 27, 2018]

[11] “Intel Open Source HD Graphics Programmers’ Reference Manual (PRM)”, Volume 5: Memory Views. Intel. June 2015. [Online] Available: <https://www.x.org/docs/intel/CHV/intel-gfx-prm-osrc-chv-bsw-vol05-memory-views.pdf>, [Accessed: Feb 27, 2018]

[12] “What is Vertex and Pixel Shading?” [Accessed: Feb 27, 2018] Available: <https://support.ubi.com/en-US/Faqs/000010031/What-is-Vertex-and-Pixel-Shading-1364550233142>

[13] A. Muennoi and D. Hormdee, “3D Web-based HMI with WebGL Rendering Performance” in *3rd International Conferences on Mechanics and Mechatronics Research 2016*, pp. 1-5 [Accessed: Mars 18, 2018]

[14] M. Almashor and I. Khalil, Fully Peer-to-peer virtual environments with 3D Voronoi diagrams, *I. Computing* (2012) 94: 679. [Accessed: Mars 18, 2018]

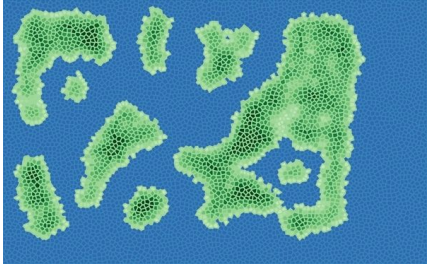
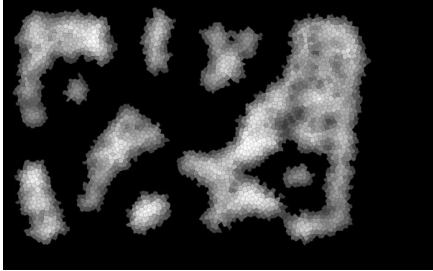
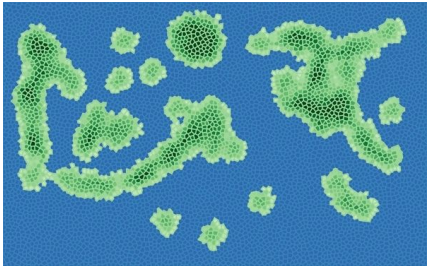
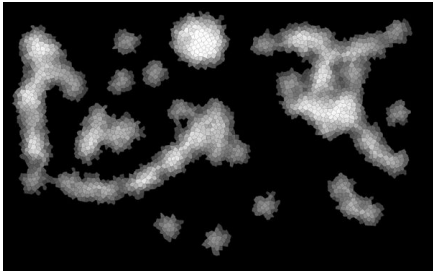
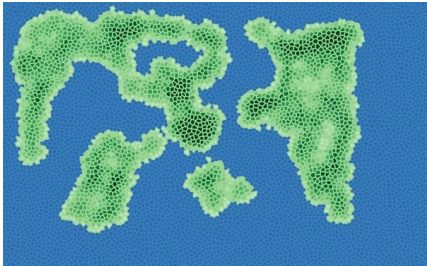
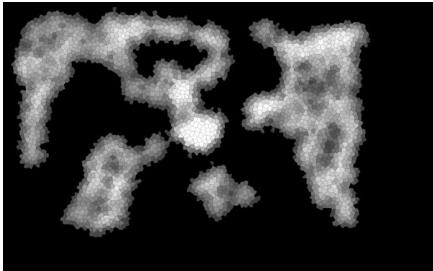
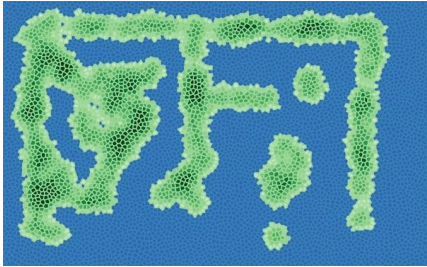

[15] sindresorhus, “Opn: A better node-open. Opens stuff like websites, files, executables. Cross-platform” sindresorhus 2014, [Online] Available <https://github.com/sindresorhus/opn>, [Accessed: Mars 18, 2018]

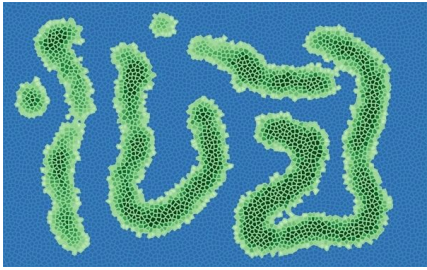

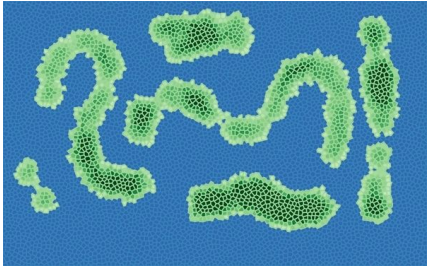
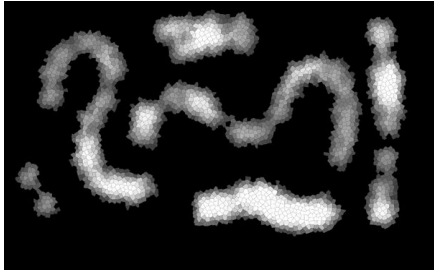
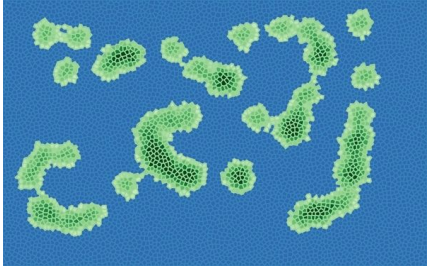

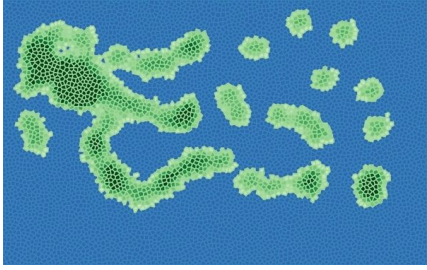

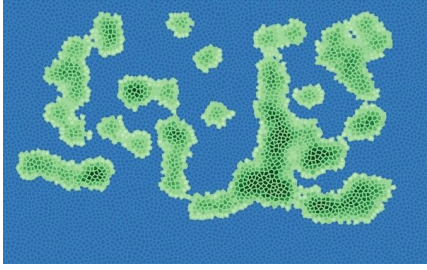

[16] sebhildebrandt, “Systeminformation: Simple system and OS information library for node.js”, 2014, [Online] Available: <https://github.com/sebhildebrandt/systeminformation>, [Accessed: Mars 18, 2018]

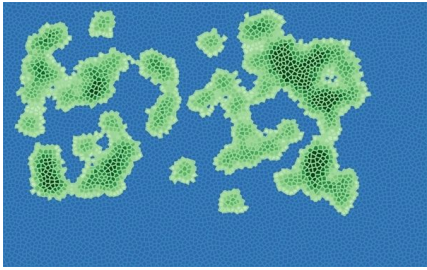
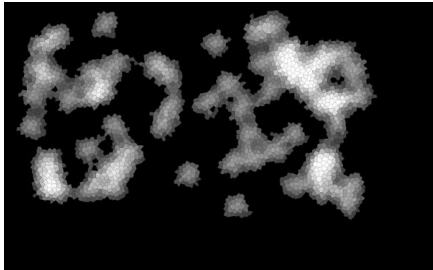
- [17] pkrumins, “Tree-kill: Kill all processes in the process tree, including the root process.” 2013, [Online] Available: <https://github.com/pkrumins/node-tree-kill>, [Accessed: Mars 18, 2018]
- [18] BabylonJS, “BabylonJS/Babylon.js”, 2018 [Online] Available: <https://github.com/BabylonJS/Babylon.js> [Accessed: Mars 18, 2018]
- [19] mrdoob, “Three.js: JavaScript 3D library”, 2018, [Online] Available: <https://github.com/BabylonJS/Babylon.js> [Accessed: Mars 18, 2018]
- [20] dmnsn, “WebGL-frameworks-libraries”, May 31, 2018 [Online] Available: <https://gist.github.com/dmnsn/76878ba6903cf15789b712464875cfdc> [Accessed: April 3, 2018]
- [21] Stephanie, “Wilcoxon Signed Rank Test: Definition, How to Run”, Sep 18, 2015, [Online] Available: <http://www.statisticshowto.com/wilcoxon-signed-rank-test/> [Accessed: April 26, 2018]
- [22] Dr. Johan Hagelbäck, “JStats”, [Online] Available: <http://aiguy.org/Statistics.html> [Accessed: April 26, 2018]
- [23] Dr. Johan Hagelbäck, “VoronoiDemo”, [Online] Available: <http://aiguy.org/voronoidemo.html> [Accessed: May 20, 2018]

□

Appendix 1

	Default	Heightmap
1		
2		
3		
4		

5		
6		
7		
8		
9		

10		
----	---	--

Appendix 2

All analysed raw data [gist](#).

Appendix 3

List containing explanation of acronyms.

- **CS** - Command Streamer
- **VS** - Vertex Shader
- **SDE** - South Display Engine
- **GAM** - GFX Page Walker
- **VF** - Vertex Fetcher
- **CL** - Clip Unit
- **SF** - Shared Function
- **TDG** - Thread Dispatcher Global
- **GAFM** - Graphics measuring unit
- **GAFS** -Data return policy from FFROB is a round-robin. For GPU tasks and motion planning.